ERNW NEWSLETTER 55/ SEPTEMBER 2016

THREAT ANALYSIS OF MALICIOUS APPLICATIONS ON MOBILE OPERATING SYSTEMS

Version:	1.0
Date:	19.09.2016
Author(s):	Benjamin Schwendemann

ERNW Enno Rey Netzwerke GmbH
Carl-Bosch-Str. 4
69115 Heidelberg

www.ernw.de
www.troopers.de
www.insinuator.net

Friedwart Kuhn
Digital unterschrieben von Friedwart Kuhn
Datum: 2016.09.19 23:45:51 +02'00'
Adobe Acrobat Reader DC-Version: 2015.017.20053

TABLE OF CONTENT

LIST OF TABLES

1 Abstract

In the following thesis a threat analysis of malware on mobile OSes will be performed. We use Windows Phone OS as an example.

One of the differences of mobile OSes to Desktop OSes is the enforcement of sandboxing techniques of applications. On mobile OSes these separate applications from each other but also from the OS itself. We will take a look at the security model of Windows Phone OS and see how sandboxing is implemented. With that we also analyze the structure of compiled application packages.

To perform the threat analysis, we use a methodology based on STRIDE. The acronym STRIDE stands for typical threats software is susceptible for, like tampering with data. For a structured approach we identify a data flow diagram consisting of typical parts of a mobile OS. Afterwards STRIDE is applied to the components of that diagram.

The result shows threats for each element of applications and OS from the perspective of a malicious application running on a mobile device. This can be used for analyzing a concrete application or as a basis for a risk analysis.

2 Introduction

Mobile phones have gone a long way, starting as quite simple devices with a two color LCD display and with buttons to dial and use it. Still the LCD display and buttons are there, though the number of buttons became fewer and the pixels and colors on the display more. But not only this. As technology evolved computers are becoming smaller and more powerful -- so small that one can even carry it with one hand and make phone calls. That is exactly what today's smart phones are: small computers with fully powered OSs, capable of surfing the Internet and running basically any kind of software. Similarly powered are tablets and also the first watches, e.g. like the Apple Watch or Samsung Galaxy Gear.

All those forms of computers along with laptops are mobile devices. Laptops are still a bit different as they typically run desktop OSs, while the other devices typically run special mobile OSs. The latter often have to meet more restrictive hardware requirements, as smart phones and tablets are still not as fast as desktop systems. The focus of this work will be on mobile devices with mobile OSs.

One key aspect in which desktop OSs and mobile OSs differ is the enforced sandbox in mobile OSs. That means, that every application runs in a sandbox, aiming to limit the permissions and capabilities to a minimum. Also this is applied to the user, as there is no administrative access for him. On one hand this helps to enforce security on different levels and also can help to introduce new security concepts. But also it can be frustrating for experienced users, which are used to have full control over their systems.

To gain administrative access, users jailbreak or root their devices. The motivation for this is quite different. Sometimes users want full access, because it is their device and they want to do with it, whatever they want. Sometimes enhanced access is needed to implement features, which will not work under normal circumstances, due to the limits of the sandbox. Also users may want to avoid the vendor's store, like on iOS and Windows Phone, to either install applications, which normally would not work, or pirated software.

The problem with jailbreaking or rooting is that they lower the security of the device by removing certain security features, e.g. allowing unsigned code to be run on the iOS. (Chell, Ersamus, & Jon, 2015)

The motivation for this work is to perform a threat analysis for malicious applications running on a mobile OSs. The base for this is an overview of typical sandboxing features and the attack surface of mobile devices, like smart phones, tablets and smart watches. The Windows Phone OS will be taken as an example.

2.1 Related work

Unfortunately Microsoft's own documentation about Windows Phone is quite vague, when it comes to technical details of their implementation, though MSDN (Microsoft Developer Network) is quite a good starting point for developing applications for Windows Phone.

The Mobile Application Hacker's Handbook by (Chell et al., 2015) presents a practical approach for analyzing applications on mobile OSs and can be used as guide for security researchers and forensic analysts. Also they present an overview of the respective underlying OS.

(Franz, 2015) presents a security analysis of Windows Phone's inter application communication techniques in his thesis and also discusses the exploitability of this communication.

For our threat analysis we will use an approach based on STRIDE. (Shostack, 2014) presents a comprehensive guide on using STRIDE for threat analysis.

2.2 Structure of this work

We will start by giving an overview over sandboxing mechanisms and security features of a mobile OS with the help of Windows Phone OS as an example. It is presented how applications are protected and isolated. A part of the protection relies on enforcing certain exploit mitigation features, which help to prevent attacks from malware on the phone. In chapter 4 the structure of a typical Windows Phone application is explained, which helps to further understand the mechanics of the underlying platform. To set Windows Phone in context, a comparison to Apple's iOS and Google's Android is made in chapter 5 with focus on the sandbox and security features.

Before starting the threat analysis in chapter 7, we explain our methodology. It is based on STRIDE and we will use a data flow diagram, which is presented in chapter 6. Also a list of attack vectors with examples of attacks and malware using them is presented.

The threat analysis will base on our data flow diagram to examine each key component of mobile OSs and is centered around a malicious application installed on the device. It is divided into the element types of the flow diagram, where in each section the respective elements are explained and analyzed based on STRIDE. We will also discuss the feasibility of performing a risk analysis in our work.

We will close this work with hardening recommendations for end users, developers and corporate environments in chapter 8 and present a conclusion in chapter 9.

3 Windows Phone Security Model

Similar to Apple's iOS, Windows Phone OS can be seen as a closed platform. This is due to the fact, that only applications verified and digitally signed by Apple or Microsoft respectively may be installed on the device. This is also one part the security of both platforms rely on, as during the submission process scanning for malware and suspicious behavior is taking place. So the user and vendor of the respective store can be quite sure, that provided applications are not bad. In contrast, applications which come from Google's Play Store are signed by the developer of the application, but not from Google, which means the application only needs a valid developer signature to install. This does not mean that Google does not scan applications before placing them to the Play Store, but applications can be installed from untrusted sources, such as a download from a website or a 3rd party application store.

While Windows Phone 7.x and the older Windows Mobile OSs are built on top of the CE kernel, Windows Phone 8.x and later are based on the NT kernel. The NT kernel is also running the desktop Windows OSs and as of (Cunningham, 2015) aligns with Microsoft's strategy to unify the mobile and the desktop OS versions. Because of that some of the following security concepts apply to both desktop and mobile Windows.

3.1 Application Protection and Control

As Windows Phone 8 is designed as a closed system, the only way to install new applications is by using Microsoft's Windows Store. With that Microsoft has also the possibility to perform formal checks as well as security checks on new applications and updates for existing applications. (Microsoft, n.d.-b) offers a short overview of the submission and certification process in the MSDN.

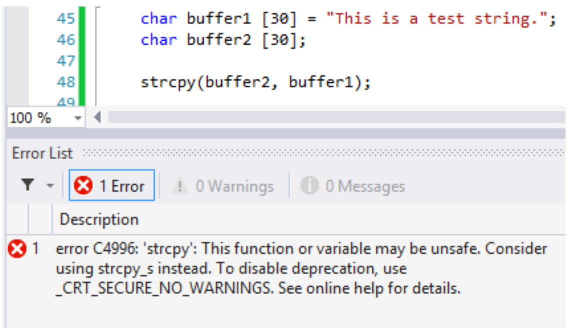
3.1.1 Store Submission

When a new application or an update for an application is submitted by the developer, Microsoft will perform multiple steps to ensure that the submission contains no malware and also to enforce a certain security and usability level.

Upon uploading the application package, a formal check is run on the package. Further details on the structure of an application package are covered in chapter 4.

One security check is of course a scan for malicious code. But even if the scanner does not find malicious looking code, there still can be suspicious behavior in the application. This could be an attempt to access files outside of the application's sandbox or the Windows registry, although this should be blocked by the sandbox anyway. Questionable behavior like this may lead to rejection of the application. Also altering the behavior of the application after certification is prohibited. This can be achieved by downloading additional code after installation on the device, e.g. additional JavaScript for a JavaScript based application. It seems that detailed information about the prohibited behavior is not made publicly available by Microsoft.

Keeping in mind that Windows Phone 8 and 8.1 support native code like C and C++, it is rather interesting that there are seemingly no restrictions regarding unsafe API-calls like `strcpy()` or `puts()` (Chell et al., 2015). At least Visual Studio marks them as unsafe upon compilation with default settings (see Figure 1).



```
char buffer1 [30] = "This is a test string.";
char buffer2 [30];

strcpy(buffer2, buffer1);
```

```
error C4996: 'strcpy': This function or variable may be unsafe. Consider
using strcpy_s instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS. See online help for details.
```

Figure 1: Visual Studio 2013 Community Edition showing an error upon compilation that the usage of `strcpy()` *may be unsafe. This is a default installation and to compile the project either those type of errors must be explicitly disabled or an alternative must be used.*

Altogether there are essentially no checks for insecure coding practices, like storing user passwords in plain text on the device or simply checking for usage of the before mentioned unsafe methods. Windows Phone 8.1 applications, in contrast to Windows Phone 8 applications, must pass the Windows App Certification Kit (Chell et al., 2015). The Windows App Certification Kit is a software that can be used to test the technical compliance of an application on the developer's local machine before the application is submitted to Microsoft. The application is then checked against formal tests like the correctness of the applications manifest file, but also performance tests, which focus on user experience (Microsoft, n.d.-e). (Microsoft, n.d.-f) provides details in the MSDN.

3.1.2 Code Signing and DRM

After an application or update successfully went through the security and compliance checks mentioned before, it will be signed by Microsoft Marketplace Certification Authority. This signature is verified before installation. It must be created by Microsoft to ensure that the application went through Microsoft's submission process.

As of (Chell et al., 2015) Microsoft uses its PlayReady-DRM to protect applications from modifications and to verify the application package before installation.

3.2 Application Sandbox

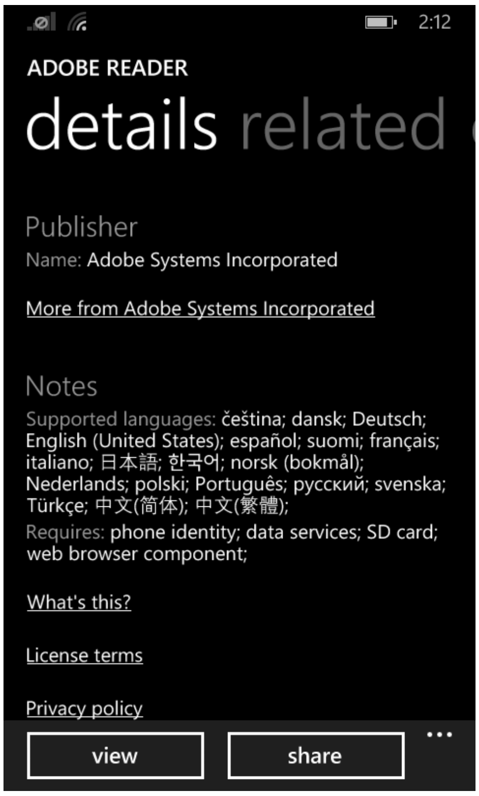
Although Microsoft may perform comprehensive checks on submitted applications and updates, there are application sandboxes in place to further protect the user, the system and even other applications from each other. It is also in line with the closed system design. Application sandboxing is achieved with Security Identifiers (SIDs) and integrity levels to form the *AppContainer*.

In Windows SIDs are used to identify users, groups or other security principals, like services or processes. SIDs are then used in Access Control Lists (ACLs) to check which security principal has access to which resources. In Windows versions prior to Windows Vista only users and groups had an SID assigned, while Windows Vista introduced SIDs for services. Since Windows Phone 8 each AppContainer gets its own SID. The SID is then used to evaluate what level of access the application in that AppContainer has.

An independent concept are the integrity levels, which provide a Mandatory Access Control (MAC) for write access. Each process, or more precisely each SID, has an integrity level assigned and has only access to resources with the same or lower integrity level. Windows has five integrity levels:

System, High, Medium, Low and Untrusted. For example if a process belongs to the SYSTEM group, but has low integrity level, it actually will not be able to make changes to the system. Though the ACL may allow access to the system, the integrity level blocks the write access, as the system has a higher integrity level that the said process has. (Comvalius, 2013)

Following the least privilege principle by default every application is running in an own AppContainer and is only able to access its own folder structure. If more permissions are needed, an application may request certain *capabilities* at installation time. Those have to be defined by the developer in the `Package.appxmanifest` file, which is read and handled by the installer. When a user likes to install an application from the Windows Store, the capabilities are handled similar as on Android. Until Windows Phone 8.1 Update 2, the user cannot choose which capability he wants to grant, but just has an all-or-nothing choice, which means that he has to choose to accept all requested capabilities on installation or to not install the application at all. With Windows Phone 8.1 Update 2 the user is able modify the access for applications on user data.


2:12

ADOBE READER

details related

Publisher

Name: Adobe Systems Incorporated

More from Adobe Systems Incorporated

Notes

Supported languages: čeština; dansk; Deutsch; English (United States); español; suomi; français; italiano; 日本語; 한국어; norsk (bokmål); Nederlands; polski; Português; русский; svenska; Türkçe; 中文(简体); 中文(繁體);

Requires: phone identity; data services; SD card; web browser component;

What's this?

License terms

Privacy policy

Figure 2: A screenshot showing the requested capabilities of the Adobe Reader application in the Windows Phone Store. The user has to switch to the detail view to see them. The user can follow the "What's this?"-link to find descriptions for the capabilities.

Looking at the OS-level, the capabilities are handled with SIDs and groups. There exists a group for each capability and the AppContainer's SID is added to the respective groups by the application installer. The access on the resources then is handled by ACLs. A few example capabilities noted in XML are:

- `<m3:Capability Name="appointments"/>` -- access to the calendar
- `<Capability Name="internetClientServer"/>` -- access to the Internet, acting as client or server
- `<DeviceCapability Name="location"/>` -- access to the location service
- `<Capability Name="musicLibrary"/>` -- access to the music library

Additionally some capabilities are device capabilities, which also act as a prerequisite, like having a camera or a GPS sensor. Devices which do not fulfill these requirements will not be able to install that application as the installer blocks them.

Actually there do not exist capabilities for some resources, like opening a file from the internal storage. For example if a user wants to view a downloaded PDF, the PDF viewer application does not load the file directly, but calls a trusted broker process, which belongs to the OS. This process lets the user pick the file via a dialog on the screen and handles the chosen file back to the calling process. (Chell et al., 2015; Franz, 2015)

3.3 Data Encryption

If a phone is lost or stolen, not only the value of the device itself is lost, but also the data on the phone. The data may be accessed by the thief or finder, which is especially critical in targeted attacks against companies. Even when it's a private device the user may not want the thief or finder to be able to access private data such as photos or the address book.

To prevent extracting data from the flash storage, it should be encrypted "at rest", which means that either a file based encryption or a full disk encryption should be in place. When full disk encryption is enabled, the whole disk is encrypted, except at least a small part for booting and decryption. Typically the user is asked for the password to decrypt the device during the boot process. After that the key will reside in the memory, so that data can be encrypted when written to the disk and decrypted when read from the disk. Mostly this process is transparent to the user and processes, after the disk is mounted properly. File based encryption is working in a similar way, though on each file separately. Though Windows Phone OS offers encryption using Microsoft's BitLocker technology, the status quo is quite surprising.

3.3.1 Internal

By default the internal storage is not encrypted. And there is a problem with that: private users simply have no option to enable (or disable) encryption on the device. To enable encryption the user must either connect to a Microsoft Exchange server, which requires encryption, or connect to an enterprise device management server, which pushes an appropriate policy down to the phone.

Also there is no option to disable the device encryption, neither by the Microsoft Exchange server nor by policy. (Microsoft, n.d.-i)

3.3.2 SD-Card

Removable storage like SD-Cards, whose usage many Windows Phone models offer, are not encrypted at all. That means that personal information like videos or photos are stored on the SD-Card on an unencrypted partition. So the user (or company) has no possibility to protect the data on the external storage. As of (Microsoft, 2014) with Windows Phone 8.1 the user can choose to move applications and their data on the SD-Card on an encrypted and hidden partition. So there are two partitions on the SD-card: an encrypted partition for the applications and their data and an unprotected partition for the user data, like photos and music. The advantage of this is, that the photos and other user data on the SD-Card can be read from other devices, while the installed applications are protected.

3.4 Exploit Mitigation

To attack a system an attacker may want to execute his own code -- at best with high privileges. Typically processes with high privileges or even the kernel itself are attacked with exploits. A group of techniques to achieve this, are buffer overflows, which aim to write arbitrary code into memory and bring it to execution in the attacked process' context, which means the code executes with the same privileges.

One famous example for an unsafe and exploitable function is `gets()` from the `stdio` c-library, which reads data from an input stream to a buffer until a termination character is found. The problem here is, that it never checks the size of that buffer or the data from the input. Therefore the read data may exceed the size of the buffer and partially is written into other parts of the memory.

In the past years different mitigation techniques came up to harden OSs and programs against this kind of attacks and simple buffer overflow attacks, as just explained, mostly will not work anymore. Though the paper from (Sotirov & Dowd, 2008) focuses mainly on bypassing the following exploit mitigation techniques, they also explain them in detail. So only a short overview of the techniques implemented in Windows Phone 8 and 8.1 is given here.

3.4.1 Data Execution Prevention (DEP)

When a program is running, the different memory regions of it can roughly be separated into two parts: code memory and data memory. The code segment is loaded by the OS when the program is started and is read-only. Data segments are readable and writable, but as code should not reside there, there is generally no reason for them to be executable. This is exactly what DEP does: it prohibits execution of data segments by setting the so called NX-bit (**n**o e**x**ecute)[1]. DEP is not only a compiler and kernel option, but also must be supported by the processor.

[1] *Depending on the hardware vendor, names like XD (**e**xecute **d**isable) or NX (**n**ever e**x**ecute) may also be common.*

For an attacker this means, that code he injected into (data) memory is not executable. To circumvent DEP one can rely on return-oriented programming, which uses already present code to build a chain of small code fragments to execute a task.

(Fan, 2009) stated in a blog post on the Microsoft Developer Network, that the compiler option `/NXCOMPAT` to enable DEP was introduced in Visual Studio 2005 and is enabled by default for processors with x86 architecture. Windows for non-x86 architectures enforces this.

3.4.2 Address Space Layout Randomization (ASRL)

The basic idea behind ASLR is to randomize the layout of a program in memory, i.e. its executable, stack, heap and libraries. So between distinct runs the base addresses of the before mentioned program parts are changed randomly. The goal of this technique is to make such key addresses unpredictable to attackers and make exploits basing on memory corruption less successful.

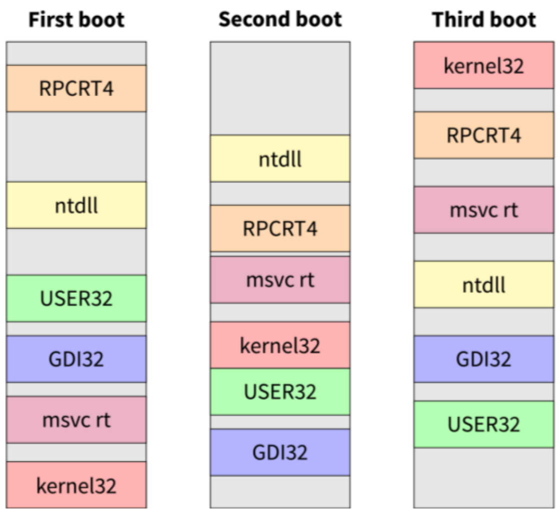


Figure 3: ASLR is on. Libraries and processes are loaded at random memory locations between runs.

Taking a stack based buffer-overflow attack as an example, the attacker normally wants to overwrite a function pointer or the return address of an exploited function with a constant predetermined address. This can be an address of a loaded standard library or some arbitrary code, which was injected by the attacker beforehand.

When ASLR is in place (see Figure 3), those addresses will not be constant anymore and exploits with fixed addresses are likely to fail. But there are advanced exploitation techniques to find the needed addresses or at least make it more likely to bring arbitrary code to execution, like "heap spraying". With that technique the attacker tries to "spray" his code all over the heap to make it more likely that it gets hit and executed. (Chell et al., 2015)

The compiler option `/DYNAMICBASE` for ASLR was introduced in Visual Studio 2005 and is supported since Windows Vista. It is default on since Visual Studio 2008. (Fan, 2009) For further information on ASLR the reader can check: Address-space randomization for windows systems; (Li, Just, & Sekar, 2006).

3.4.3 Stack Canaries

Stack Canaries are also known as Stack Cookies. Those are random values placed prior to critical data on the stack e.g. the return pointer. When data is popped from the stack again and the canary is reached, its value will be checked. If it changed, the program will be terminated, as memory corruption took place, like a buffer overflow. In order to overwrite data on the stack, which is protected in this way, the attacker needs to find out the canary's value beforehand.

There is also a variant which uses string termination values like the null byte, carriage return or line feed. Though in this case the value is known to an attacker, it protects functions like `gets()` or `strcpy()` as they both work on string data. As strings are terminated by null bytes both of them will stop when they reach a null byte and return. For the attacker this means he also has to include such a string termination value at the same location as the canary in the data he likes to inject, which in return means that the data afterwards will not be read by said functions.

The compiler option `/GS` for Stack Canaries was first introduced in Visual Studio 2002 and is on by default. (Chell et al., 2015)

3.4.4 Safe Structured Exception Handling

With the introduction of Stack Canaries and ASLR, attacks based on overwriting return addresses were no longer reliable. But soon another way to execute arbitrary code was found by exploiting Structured Exception Handling (SEH).

Windows uses this to handle exceptions like invalid memory accesses or divisions by zero. Therefore a chain of `EXCEPTION_REGISTRATION_RECORD`s (ERR) is build, which basically contain the function pointer of the exception handler and the pointer to the next ERR. When an exception occurs each handler has the chance to handle the exception or pass it on. The ERRs are stored on the stack. (Pietrek, 1997)

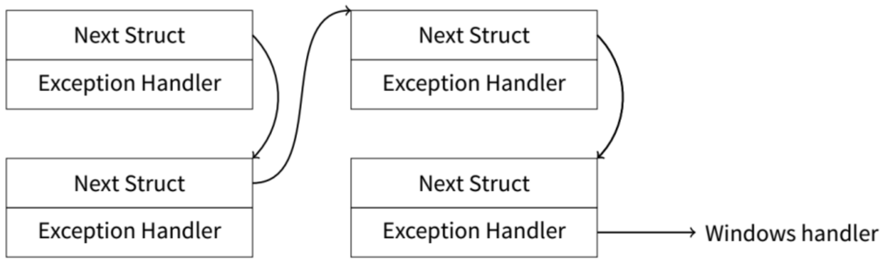


Figure 4: Structured Exception Handling. Each `EXCEPTION_REGISTRATION_RECORD` *holds a pointer to the next record and a pointer to the exception handler. When an exception occurs, each handler has the chance to handle the exception or to pass it. The last handler is set by the OS and will always handle the exception by terminating the process.*

(Litchfield, 2003) describes an exploiting technique, which overwrites ERRs with an arbitrary pointer. So when an exception occurs, e.g. by accessing invalid memory space, the attacker's code will be executed.

Microsoft introduced SafeSEH as a mitigation to this problem. When an exception occurs each exception handler is checked against a table of known exception handlers before execution. So if the pointer is tampered this check will fail, though (Litchfield, 2003) was also able to circumvent this. To further mitigate this, *Structured Exception Handling Overwrite Protection* (SEHOP) was developed which places a cookie at the end of the SEH chain. Before the exception handling is executed SEHOP walks the whole chain of ERRs, validates that cookie and stops execution if that check fails. So when an attacker overwrites the function pointer of the execution handler, he necessarily also overwrites the *Next Struct* pointer as this is in front of the function pointer. Then the chain is broken as *Next Struct* points to somewhere else and SEHOP's check will fail.

The compiler flag `/SAFESEH` is enabled by default on all versions of Visual Studio and SEHOP is implemented in Windows Phone 8 and 8.1. (Chell et al., 2015)

3.4.5 Heap Exploit Mitigation

While the before mentioned techniques are a bit more stack related, there are also protection techniques for the heap. One basic difference here is that the data on the stack is located in a linear order in memory, which does not hold for the heap, although it is organized as a double linked list.

In their paper (Valasek & Mandt, 2012) give a detailed overview over the data structures and allocation algorithms for the heap and also exploit mitigation techniques in Windows 8. They conclude that "this version of Windows probably has more mitigations added than all other versions combined" for the user land heap, which also includes new data structures and improved algorithms for allocation and freeing of memory. Though in kernel land there are basically no improvements on the structures and algorithms, there are still a lot mitigation techniques in place, so generic attacks on the kernel heap will not work anymore.

Going into details on each mitigation technique for the heap is not in the scope of this work, but as an example *Guard Pages* are explained in the following. Like on the stack there exist also buffer overflow attacks on the heap. When a buffer on the heap is overflown, data will be written into another block on the heap. When certain heuristics are triggered, such as allocation of large numbers of blocks with the same size, Guard Pages are placed between those blocks, which are inaccessible in any way. So in the case of an overflow attack a segmentation fault is caused, when the overflow reaches the Guard Page.

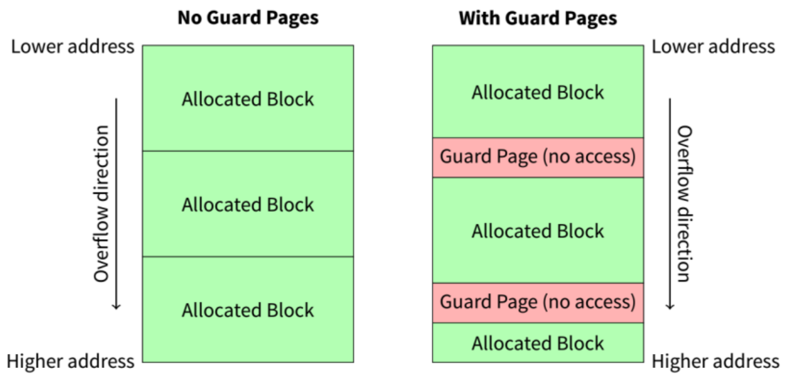


Figure 5: Guard Pages are placed in between allocated data blocks on the heap and are inaccessible in any way. Overflow attacks will cause segmentation faults when they reach the Guard Page.

3.4.6 Exploit Mitigation in kernel space

As already stated in the previous section, there are exploit mitigation techniques in place, which quite raise the bar to successfully run an exploit on the kernel heap. One example here is safe linking and unlinking, yet again details can be found in the paper from (Valasek & Mandt, 2012).

In addition similar techniques like the above are also used in kernel space (Chell et al., 2015):

- DEP / NX (for non-paged pools)
- ASLR
- Stack Canaries
- NULL pointer dereference protection

3.4.7 Exploit Mitigation on Windows Phone

Though both desktop and mobile Windows run on the same NT kernel, the active exploit mitigation techniques may differ, especially for the user programs. For the kernel space it is easier to apply such techniques as the kernel is built by Microsoft and user programs do not need to be recompiled. As (Chell et al., 2015) state, the before mentioned techniques are all active on Windows Phone.

Regarding user land, enforcing exploit mitigations is not always that easy. Some techniques like DEP are easier to apply, when the kernel (and processor) supports this. In executables mostly the code and data part are distinguishable and DEP can be applied on that[2]. On the other hand to activate ASLR the binary needs additional information. (Li et al., 2006) call this allocation information, which makes it possible to place the program parts and libraries in different regions of the memory and change pointers accordingly. Without this information ASLR cannot be fully applied, as for a binary scan integer values and pointers both are constant values and look the same.

With that in mind the user cannot be sure, if each exploit mitigation technique is active for each program on a desktop system, especially when it comes to older programs. But the setting is a bit different on Windows Phone as each application in the store is scanned by Microsoft and applications which are not compiled accordingly can be rejected, which is the reason each of the previously shown technique is ensured to be active on Windows Phone. (Chell et al., 2015)

3.5 Secure Boot

The above exploit mitigations and security features would be useless if the kernel is corrupted. One way to do this, is to actually boot another kernel. An attacker might boot his own, modified kernel, which for example logs passwords the user enters or dumps data on the SD-Card, which can be read later. To ensure that the real, uncorrupted kernel and components from Microsoft are running on the device, Windows Phones utilizes *Secure Boot* which is part of the UEFI specification. UEFI stands for Unified Extensible Firmware Interface, which is a replacement for the BIOS (Basic Input/Output System).

Secure Boot uses public key cryptography to verify the integrity of the OS and also the UEFI software itself. That means that each component is digitally signed and when booting, this signature is checked:

1. When powering on the device, the pre-UEFI boot loader signature is verified against the root of trust and loaded if this succeeds. This is done on hardware level.

[2] *It might happen, that some programs are not DEP compatible and do not run stable anymore.*

2. The UEFI boot loader is verified and loaded.
3. The UEFI boot loader verifies each UEFI application or driver (including the Windows Phone Boot Manager). If any of the checks fail, the binary will not be loaded and the boot fails.
4. The Windows Phone Boot Manager checks if certain settings are intact (or resets them to default) and checks the integrity of the Windows Phone OS boot loader.
5. The Windows Phone OS boot loader validates and loads the OS kernel and boot-critical drivers. Then the responsibility for checking and loading further drivers and applications is handed over to the Windows kernel.

If any of the steps fail, the booting fails. (Microsoft, n.d.-i)

3.6 Summary

Due to Microsoft's closed system approach, they are able to enforce developers to compile binaries with exploit mitigation techniques enabled, which can improve the security of the system. This could be further improved by scanning applications for insecure coding practices, which can be as simple as banning functions like `strcpy()` or `puts()` from C-code.

To enforce the sandboxing of application Windows Phone uses separate SIDs for the applications and projects capabilities on memberships of different groups. Also MAC with different integrity levels further helps to protect the OS from malicious applications. Unfortunately normal users are not able to encrypt their devices, although it seems that all Windows Phones are delivered with Microsoft's BitLocker-technology.

To ensure that neither the boot-loader nor the OS is tampered with, Microsoft uses Secure Boot on Windows Phones.

4 Structure of Windows Phone 8 Applications

As discussed in the last chapter, there are formal checks on applications or updates during the submission process. This helps to enforce certain security features, but also is needed to verify the package, so the application can be properly installed and configured.

4.1 Application Package

Windows Phone uses either XAP-files or APPX-files for the distribution of applications. Similar to iOS's IPA-files and Android's APK-files those are essentially zip files, though due to the DRM-protection they are encrypted and signed by Microsoft.

Essentially an application package of either format contains the needed binary files or .NET assemblies, resources, like images and sounds, and a manifest file, along with some other files. The package contents of our simple test application is shown in the following listing:

```
Archive:  TestApp.appx
Length      Date    Time    Name
---------  ---------- -----   ----
     2516  2015-09-25 16:08   Assets/Logo.scale-240.png
      753  2015-09-25 16:08   Assets/SmallLogo.scale-240.png
    14715  2015-09-25 16:08   Assets/SplashScreen.scale-240.png
     1122  2015-09-25 16:08   Assets/Square71x71Logo.scale-240.png
     2200  2015-09-25 16:08   Assets/StoreLogo.scale-240.png
     4530  2015-09-25 16:08   Assets/WideLogo.scale-240.png
     3150  2015-09-25 16:08   MalAppTest.xr.xml
     8192  2015-09-25 16:08   MalBackgroundTask.winmd
     2808  2015-09-25 16:08   MainPage.xbf
     4080  2015-09-25 16:08   MapPage.xbf
     3704  2015-09-25 16:08   CalendarPage.xbf
      690  2015-09-25 16:08   App.xbf
     3430  2015-09-25 16:08   ContactPage.xbf
     3872  2015-09-25 16:08   MalTask.xbf
    35328  2015-09-25 16:08   MalAppTest.exe
     2928  2015-09-25 16:08   resources.pri
     3705  2015-09-25 16:08   AppxManifest.xml
     2455  2015-09-25 16:08   AppxBlockMap.xml
      673  2015-09-25 16:08   [Content_Types].xml
---------                     -------
   100851                     19 files
```

Listing 1: The listing of contents of an APPX-file from a simple example application.

It depends on the Windows Phone OS version if XAP- or APPX-files can be used. XAP was the format for Windows Phone 7 and 8 and APPX was introduced exclusively for Windows Phone 8.1, as Windows Phone 8.1 has new features and APIs, but also to unify the package distribution of Windows Phone and Windows Desktop. XAP-files for Windows Phone 8 can still be installed on Windows Phone 8.1 and XAP-packages for Windows Phone 7 can be installed on Windows Phone 8, but not the other way around. (Chell et al., 2015)

4.2 Application Manifest

The application manifest describes the details of the application, like its name, author, description and a unique application ID. In the case of our test application it is named `AppxManifest.xml`, though this may vary depending on the target OS version. In any case it is an XML-file, which also contains different sections describing the possible interactions of the application.

One of them is the capabilities section, which introduces the requested capabilities to the installer. As described before, the installer then uses them to assign the application to the corresponding groups. The following listing shows a snipped of our test application requesting most capabilities.

```
<Capabilities>
  <Capability Name="internetClientServer" />
  <m3:Capability Name="contacts" />
  <m3:Capability Name="appointments" />
  <Capability Name="musicLibrary" />
  <Capability Name="picturesLibrary" />
  <Capability Name="removableStorage" />
  <Capability Name="videosLibrary" />
  <DeviceCapability Name="location" />
  <DeviceCapability Name="microphone" />
  <DeviceCapability Name="webcam" />
</Capabilities>
```

Listing 2: Part of AppxManifest.xml describing requested capabilities.

After installation this test application would be able to access the Internet, read and write both contacts and calendar entries and gets access to any media library and the SD-card. Also it would be able to activate the microphone and the camera and read the user's location. Some of those capabilities are device capabilities, like the last three in the listing. Those also act as requirements, which means that devices not fulfilling them cannot install the application, even if the user would be willing to grant the capabilities.

The notation and naming of the capabilities may vary depending on the type of developed application, e.g. for Silverlight applications the capability for accessing contacts would be named `ID_CAP_CONTACTS`.

Another section of the manifest describes background tasks, file type associations and protocols.

```
<Extensions>
  <Extension Category="windows.backgroundTasks"
EntryPoint="MalBackgroundTask.MalTask">
    <BackgroundTasks>
      <Task Type="systemEvent" />
      <Task Type="timer" />
    </BackgroundTasks>
  </Extension>
  <Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="mft">
      <DisplayName>MyFileType</DisplayName>
      <SupportedFileTypes>
        <FileType ContentType="text/plain">.mft</FileType>
        <FileType ContentType="text/plain">.foo</FileType>
        <FileType ContentType="text/plain">.bar</FileType>
      </SupportedFileTypes>
    </FileTypeAssociation>
  </Extension>
  <Extension Category="windows.protocol">
    <Protocol Name="mp">
      <DisplayName>MyProtocol</DisplayName>
    </Protocol>
  </Extension>
</Extensions>
```

Listing 3: Part of AppxManifest.xml describing the application's background tasks, file type associations and protocols.

Our example application would be able to run one or more background tasks of the class called `MalBackgroundTask.MalTask`, where the first part is the namespace and the second is the name of the class. It would be able to register them with a system event or a timer. For example the background task could be activated when an Internet connection is available or every 15 minutes. As (Microsoft, n.d.-d) details in the MSDN, it has to be noted, that background tasks encounter certain limits regarding CPU time, memory space and transferred data, like 0.625 MB every two hours and not more than 7.5 MB a day for an application that is not on the lock screen. The following listing shows a code snippet for registering a background task with certain events.

```
await BackgroundExecutionManager.RequestAccessAsync();
// Register task with system event
var builder = new BackgroundTaskBuilder();
builder.Name = "BackgroundTask-SystemEvent";
builder.TaskEntryPoint = "MalBackgroundTask.MalTask";
builder.SetTrigger(new SystemTrigger(SystemTriggerType.InternetAvailable, false));
BackgroundTaskRegistration task = builder.Register();
// Register task with time trigger
builder = new BackgroundTaskBuilder();
builder.Name = "BackgroundTask-TimeTrigger";
builder.TaskEntryPoint = "MalBackgroundTask.MalTask";
builder.SetTrigger(new TimeTrigger(15, false));
task = builder.Register();
```

Listing 4: Code snippet for registering one background task with a system event and one with a timer. The first will be executed when Internet is available, the other every 15 minutes.

Going back to the XML-snippet above, there are also file type associations and a protocol defined. If one or more of them are in place, the application attempts to register with the respective file type or protocol. The latter is meant for URLs. If a user clicks a URL or a file, the application will appear in the "Open with" dialog or it will be opened directly with the application, if it is the only one registered with the file type or URL. So in our example files with the endings `.mft`, `.foo` and `.bar` can be opened with the application as well as URLs starting with `mp:`. (Franz, 2015) analyses the security of those mechanisms and gives examples of attacks, e.g. by registering with the same protocols like other applications to intercept URL-calls.

Also there are other interesting sections of the manifest, like activatable classes and interface. They give hints which interfaces and libraries the application uses and which public interfaces it offers.

Altogether the application's manifest file does not only present meta information like the application's name and author, but also gives some insights on how the applications works and interacts with the system. This can be helpful for security analyses and also may reveal suspicious or malicious behavior. (Chell et al., 2015)

4.3 Application Directories

On Windows Phone the applications are installed in `C:\Data\Programs\[GUID]\Install`, where the GUID is the application's ID. This means that after the application packet's signature is verified, it is unpacked and decrypted. The binaries, assemblies, assets and other files belonging to the application will be placed there.

As stated in the previous chapter, each application owns a separate folder structure, which can be used to store data from the application, e.g. configuration files or a message database. This folder structure is located at `C:\Data\Users\DefApps\APPDATA\[GUID]`, where the GUID again is the application's ID. There is a set of predefined folders in it, where some of them might be familiar to Windows desktop users, like `Local`, `LocalLow`, `Roaming` and `Temp`. And also the meaning of each folder is similar. (Microsoft, n.d.-a, n.d.-h)

- **Local/LocalLow** This folder is used for data existing only on the device, which will not be synced to other devices. Typically this is used for browser caches or data, which is too big for synchronization. `LocalLow` has the low integrity level (see Section 3.2) and is used by applications with the same integrity level.
- **Roaming** Data which should by synced to other devices, should be placed here. Typically this is used for settings, profiles or bookmarks in browsers, so that users can access them on other devices, too.
- **Temp** The data in this folder is meant to be temporary and it could be removed by the OS at any time.

There is also another set of folders, which is used by a WebView, if used in the application: `INetCache`, `INetCookies` and `INetHistory`. (Chell et al., 2015)

4.4 Programming Languages

Windows Phone 8 and newer offers quite a variety of languages, which can be used to build applications. With that developers could choose the language they like most, but it also offers the ability to choose the language depending on the requirements. The following typical language combinations can be chosen:

- C# or VisualBasic with XAML
- C++ with XAML or DirectX
- JavaScript with HTML and CSS

With that a web developer has quite easy access to developing applications for Windows as they can use JavaScript, HTML and CSS. In that case WinRT is exposing a whole API, so JavaScript has about the same possibilities as the other languages have.

Typically applications are written in C# with the use of XAML (e**X**tensible **A**pplication **M**arkup **L**anguage), where the latter is a XML-like language used by the .NET-framework to design and layout the GUI of applications. In the same manner XAML can be used by applications written with VisualBasic or C++.

When high performance is a requirement for applications, they can be written in C++ to run natively on the device. Also DirectX can be used for GUIs, which is typically used when creating games.

Generally it is possible to mix those languages up a little bit in order to create hybrid applications. So one might use C# for the main programming logic and JavaScript and HTML/CSS for the UI. Also it is possible to call native libraries with the `P/Invoke` interface, if more performance is needed.

(Chell et al., 2015; Microsoft, n.d.-c)

5 Comparison to iOS and Android

In preparation to the threat analysis later on, it makes sense to put Windows Phone into context with the other two major mobile OSs iOS and Android[3]. Table 1 gives an overview over the security features implemented in the respective OS with focus on the sandbox.

Feature	Windows Phone 8.1	iOS 9	Android 5.1
Device Encryption	Full disk, only by policy	Full disk and file based	Full disk
External Storage Encryption	Applications and their data only	No external storage available	No, by 3rd party application
Code Signing	By Microsoft	By Apple	By developer
Application sources	Only Windows store	Only Apple store	Any store or download
Per application permissions	Since 8.1 Update 2, mostly user data related	Mostly user data related	No, by 3rd party application
Sandboxing Mechanisms	Separate user per application, ACLs, MAC	Same user per application, process-level MAC	Separate user per application, UNIX file based permissions, SELinux for MAC

Table 1: Overview over security features of Android, iOS and Windows Phone with focus on sandboxing.

Though all OS offer full disk encryption, they do it in a different way. iOS uses encryption since quite a few major releases and also enables application developers to additionally protect their files with a second

[3] *For comparison see market share reported by (comScore, n.d.)*

encryption layer in addition to the disk encryption. The developer can decide if a file should be decrypted and accessible when the phone is unlocked, after first unlock or always and also if the file should be bound to the device. On Android users can enable encryption in the settings, which causes the phone to be fully encrypted. This is enabled by default since Android 5.0. Though natively Android does not offer encryption for external storage, some hardware vendors do so with their customized versions. Unfortunately Windows Phone falls a bit short here, as Microsoft's BitLocker technology is built in, but can only be enabled by a policy, which has to be pushed on the phone, so regular users cannot encrypt their devices. Like Android, Windows Phone does not offer encryption of external storages, but will store applications and their data in a separate encrypted partition, if the user transfers them to external storage.

Due to their closed system philosophy both iOS and Windows Phone only install applications from their respective application store, where also all applications are digitally signed by Apple and Microsoft respectively. Of course Apple and Microsoft have quite a good control over the applications as they can be scanned for malware and malicious code during submission. Though Google does the same for its application store, Android also allows the installation of applications from other sources like 3rd party application stores or websites, but the application still must be signed by the developer.

With Update 2 on Windows Phone 8.1 users are now able to control the permissions applications have, though mostly regarding privacy related data as calendar, contacts, messages, microphone and camera (Vasiliu, n.d.). iOS has similar controls, as users are asked for permission when an application wants to access certain data. Android is left behind here, although there are 3rd party tools which may offer even better and finer grained controls than iOS and Windows Phone do.

To enforce its sandbox Android uses the user and group system from Linux. With that, each application gets an own user on installation with its respective directory. Also access to other directories is controlled by adding or removing the application's user to the respective group. Additionally Android has SELinux build in since version 4.3, which enables Mandatory Access Control on Android (Google, n.d.-b). Similarly Windows uses SIDs and group memberships to build its sandbox. Also integrity levels are used for MAC. While iOS is also a UNIX-like system, every applications runs with the same user. The sandbox is enforced with MAC on process-level. (Chell et al., 2015)

6 Methodology

Before going into the threat analysis in chapter 7, we first need to define our approach. There are different ways of how to approach a threat analysis, which could lead to different results, but the approach should also be depending on the scope to look at. To help understand the difference between attack vectors and threats, a list of common attack vectors is given later in this chapter, but this should also raise the reader's awareness of the attack surface today's smart phones expose.

6.1 Definitions

When talking about threat analysis clarification of some terms might be needed, as some of them might be used with a different meaning in daily usage. For better understanding the terms defined here are applied to the example of a fictive house. (Meier et al., 2003)

6.1.1 Asset

An asset is something of value. Assets can also contain or be made of other assets. This can be a house itself, but also the jewelry in it. A technical example can be a server, the data on it or even a complete network. The definition of an asset is depending on the scope and scale to look at.

6.1.2 Attack vector

An attack vector is a method or way an attacker gets access to a system. For a house this would be the doors and windows, but also the owner itself, who may the key be stolen from. These are all different vectors. In computer systems or networks this could by any protocol or open ports on a system.

6.1.3 Attack surface

The attack surface is the sum of attack vectors and tells how exposed a system is. It could be as simple as counting the doors and windows of a house, but it will become complex in computer systems. It is not only the number of open ports, but also which protocols run on them and which services are offered.

6.1.4 Vulnerability

To actually use an attack vector the attacker exploits a vulnerability. It can be seen as a weak spot of a system. When one thinks of a door, a vulnerability could be a weak lock or the door being open. This means an attacker can more easily lock pick it or just walk in. In computer systems this can be any kind of software bug, weak cryptographic protocols or bad configuration of systems.

6.1.5 Threat

Threats are independent of attack vectors. Threats describe possible events, which for example lead to loss or damage. When an attacker breaks open a door of a house, without destroying it, this may not be a threat. The threat is, that he steals objects of value or gathers information about the people living in the house. Examples for computer systems are data disclosure or taking over computer systems.

6.1.6 Risk

The term risk is quite abstract. It covers the threat itself, but a very important part is also the chance that such an event happens and what consequences it has. So the threat of an attacker successfully breaking in to one's house exists for everyone. But depending on the neighborhood and the "attractiveness" of the house, the possibility that this event occurs can be quite low. The term attractiveness here does not only include the potential goods an attacker might steal, but how well or bad secured the house is. Also the consequences of a break-in need to be considered: are there objects of value and what potential losses can occur? This can be applied to computer systems similarly, though the "neighborhood" might be the whole Internet.

6.2 Approaches for threat analysis

Before actually starting to threat model, it must be clear how to do it. (Shostack, 2014) identified four major threat modeling strategies in his book. One of them is brainstorming, which could lead to quite unconventional threats and results, but also lead to out-of-scope threats due to its unstructured nature. Though a moderator can help to keep focus on the scope, the quality of the results can differ a lot. One reason is, that a lot depends on the experience of the experts and how good they are at brainstorming. But there is no real exit criteria, which means it is unclear when the brainstorming is really finished and also if everything is covered. This also leads to problems in time planning, as it is hard to estimate the time needed for the brainstorming.

The other three strategies identified by (Shostack, 2014) are all structured approaches, which means they have a predefined way to model and find threats. When choosing one of those, it is rather a question of usefulness, than a question if the approach is "right" or "wrong". (Shostack, 2014) states that focusing on one strategy is preferable, as they are hard to combine.

Focusing on assets is in some way an intuitive way to model an environment with assets. This approach divides assets into three groups:

- Things attackers want
- Things you want to protect
- Stepping stones to either of these

Actually those groups are likely to overlap, which means an asset can be in more than one group. As an example a company wants to protect its reputation. But as this is a quite intangible asset, one might reduce it to things that are important to customers, which could typically be user credentials and credit card data. On the other hand those are things attackers want and are also tangible. Stepping stones can be things like firewalls or VPNs, which increase the effort an attacker has to take.

Focusing on attackers uses the attacker's perspective for threat modelling. It uses attacker lists and personas to find what can or will be attacked and also what efforts are taken for this. But with this it is often hard to define the resources of the attacker and also easily leads to discussions about that. This approach can also be less technical, as human aspects come into scope which are sometimes easy to miss, e.g. what if the attacker can corrupt any employee of the target company? What if this is a system administrator?

Focusing on software focuses on the software or system being deployed. If available the software documentation can be used as a starting point, like UML-diagrams, architecture overviews or API documentation. It also can be useful to include the developers of the software and let them explain the software and what it does. The latter can also lead to a common understanding of the software itself, which may differ on large projects or old software. Eventually any type of diagram can be used to find threats.

As we model a mobile OS, we will use the software-centered approach just described. The interested reader finds further details on each approach in (Shostack, 2014).

6.2.1 STRIDE threat model

When using a software-centered approach for threat modeling it might still be useful to have additional guides. This is where STRIDE can help. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges. This framework can be used to find typical threats or attacks a software might experience. Table 2 presents a definition and an example for each threat. (Meier et al., 2003; Shostack, 2014)

Threat	Definition	Example
Spoofing	Pretending to be something or someone else	Falsely claiming to be ernw.de or a technician from an ISP, who wants to check the Internet connection
Tampering	Modify data in memory, in transit or at rest	Modifying the response of an HTTP-Request to load arbitrary JavaScript-Code in the clients browser or change a spreadsheet of a colleague
Repudiation	Claiming to have not done something or to be not responsible for that	"I did not delete that file!" -- be it from a human or a process
Information Disclosure	Providing information to someone, who is not authorized to see it	Opening files or e-mails, but also filenames or capturing network packets
Denial of Service	Denying functionality or usage of a service or resource	Slowing down or making unavailable an Internet server, application or device with an overload of packets or by exploiting a vulnerability to crash a process
Elevation of Privilege	Performing actions without proper authorization	Allowing a normal user to perform commands as an admin or allow an unauthenticated user to perform any command

Table 2: The STRIDE threats with definitions and examples

Each of those terms are the opposite of security properties a system should have: Authentication, Integrity, Non-Repudiation, Confidentiality, Availability and Authorization respectively. While STRIDE might be misunderstood as categories in which threats should fit (which sometimes will not work) it is more a help to find and enumerate typical threats. The idea is to look at the software or components of it and try to find each of the threats. Any other threat that is found during that process, which maybe does not fit to any of STRIDE, is also valid and should be documented. Also some software components may be more vulnerable to some threats than to others. For example the data in a database can be tampered, so that some data is modified, added or deleted. Though it might look like there must be some kind of Elevation of Privilege threat to the database itself, it is just data. In this example the Elevation of Privilege threat actually applies to the process that controls access to the database.

6.2.2 Our approach

We will model a generic mobile OS with a data flow diagram (see Figure 7 below), which is centered around a malicious application installed on the device. Before we come to finding threats itself, we will at first take a look at common attack vectors for smart phones and mobile OSs.

To find threats, we will use STRIDE as a framework, though in a modified way, with respect to prevalence of certain threats to certain elements of the diagram. This helps to focus on important or at least most likely threats for each component. (Shostack, 2014) calls this *STRIDE-per-element*. Table 2 shows which threat applies to which component of a data flow diagram. It is slightly modified as we found, that external entities are also likely to encounter denial-of-service- and reputation-attacks.

	S	T	R	I	D	E
External Entity	×		×	×	×	
Process	×	×	×	×	×	×
Data flow		×		×	×	
Data store		×	×	×	×	

Table 3: Modified STRIDE-per-element table from (Shostack, 2014, f. 3–9)

Similar in the example before, it is quite unlikely that a data store, like a database or a file, is spoofed about the process or user accessing the data. But on the other side the process might be confused about the database it's accessing. One might argue that data stores are typically not vulnerable against reputation attacks, but it might be a logging data store and therefore its log potentially is a target of a reputation attack. Each component has to be seen as a victim. So modifying a network packet to fake the sender address is actually a spoofing attack against the receiving endpoint with a tampering attack on the data packet. This does not mean there cannot exist threats, if there is no mark for a component, but it helps to look at the right things at the right places.

Though STRIDE helps to find generic threats on each component, some limitations have to be considered on the types of threats to be found. It is assumed that there already exists a malicious application on the phone, while the way how it got there is out-of-scope in our threat analysis. This could happen by either a regular download from a legit application store, where the application is already infected or the user may be tricked to download and install an application, e.g. from an attackers website. The success of the latter attack of course depends on the considered OS, as Windows Mobile and iOS do not allow to install applications, which are not from their application store. As Apple, Microsoft and Google scan applications submitted to their stores, an attacker has to evade those scans at first to actually reach its victims. On Google's Android applications can be installed from any source, which could be a custom application store or a download from a website. There a user cannot be sure how comprehensive scans for malware are, if even in place. On Android the user just has to change his settings to allow installation of applications from untrusted sources as shown in Figure 6. There is no similar option for iOS and Windows Phone, but a user can activate developer mode to push and install applications to the device.

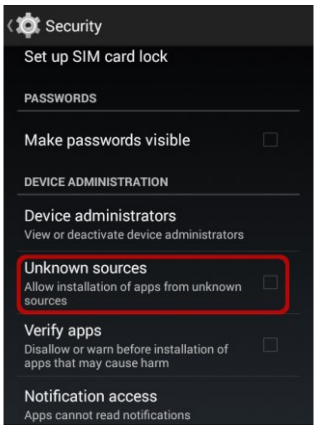
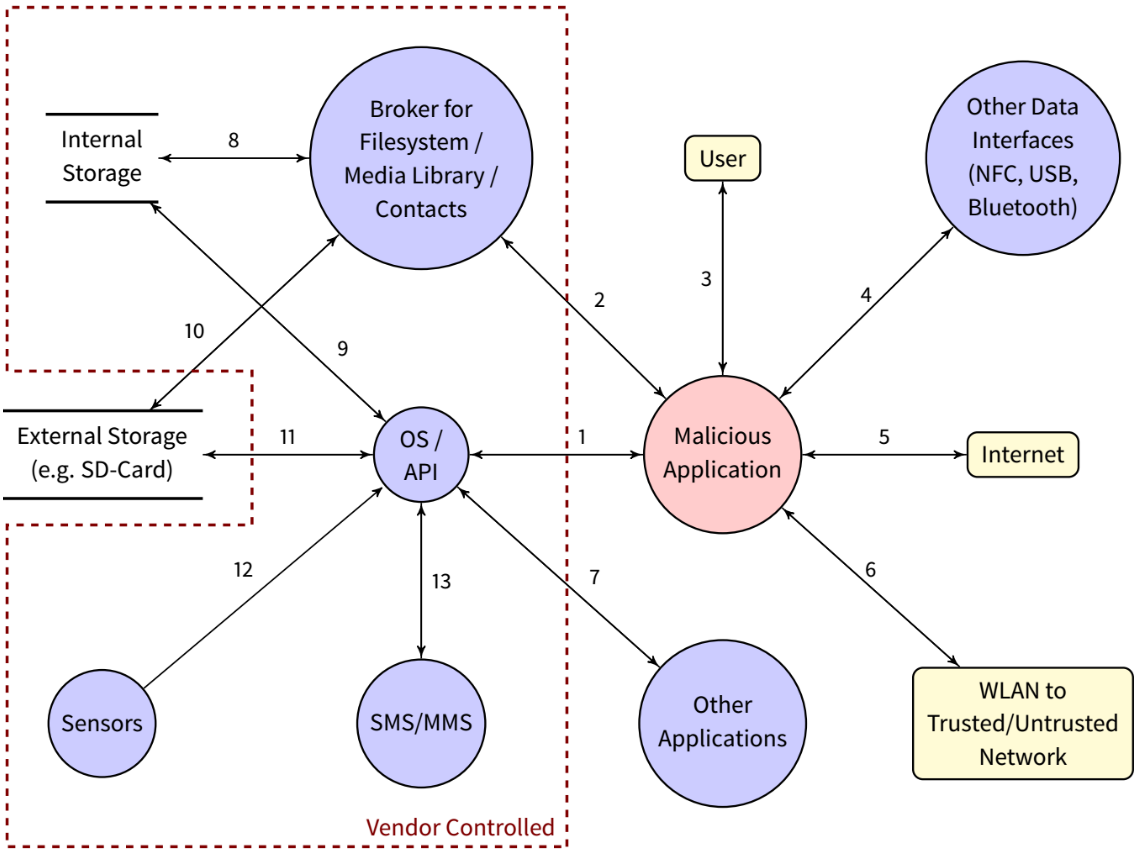

Security
Set up SIM card lock
PASSWORDS
Make passwords visible
DEVICE ADMINISTRATION
Device administrators
View or deactivate device administrators
Unknown sources
Allow installation of apps from unknown sources
Verify apps
Disallow or warn before installation of apps that may cause harm
Notification access
Apps cannot read notifications

Figure 6: Users can easily allow the installation of applications from unknown sources by enabling it in the settings.

6.3 Data flow model for smart phones

In the threat analysis we consider mobile OSs and therefore it makes sense to use a software-based approach. We decided to use a data flow model because data flows are modeled explicitly and also STRIDE can be easily applied to it with the STRIDE-per-element strategy described above. Figure 7 shows our data flow model.



Internal Storage
8
Broker for Filesystem / Media Library / Contacts
User
Other Data Interfaces (NFC, USB, Bluetooth)
2
3
4
10
9
External Storage (e.g. SD-Card)
11
OS / API
1
Malicious Application
5
Internet
12
13
7
6
Sensors
SMS/MMS
Other Applications
WLAN to Trusted/Untrusted Network
Vendor Controlled

Figure 7: Data flow model for mobile devices. The process Sensors *includes sensors like cameras, microphones, GPS, gyroscopes and magnetometers. While Internet may be accessible by WLAN, WLAN also includes other (accessible) devices inside the local network, which may not be accessible globally.*

The software on a mobile device can be split in two parts: a trusted and untrusted area (from a vendor's perspective), where the trusted area is vendor controlled and essentially the OS itself. One might argue that some vendor developed applications should be inside that trust boundary. This makes sense in a way, that when one trusts the vendor of the OS, one should trust the applications of that vendor. On the other hand it makes also sense to assume those applications to be untrustworthy from a security perspective, which means that they should have less permissions than core systems of the OS and handled as any other application.

Of course the file system is handled by the OS, which also enforces the access rights of users, groups and processes on files. But often abstraction layers are in place, like a media library. Though an application might have access to images, it might never see the full file-paths of the images. There also is a difference in the trustworthiness of the internal and external storage, where the latter typically is an SD-Card. This difference is due to the fact, that the SD-Card can be removed and accessed by other devices, so in that moment it is out of the mobile OS's control. Also often there is a single application permission for full SD-Card access, like the `ID_CAP_REMOVABLE_STORAGE` capability on Windows Phone 8, which means that files on the SD-card can easily be modified by other applications.

Though the data read from sensors like GPS or a magnetometers may be spoofed or simply wrong, the drivers reading that data are under the vendors control and are assumed trustworthy here. Also sensors are assumed as read-only devices.

Mobile OSs typically offer different ways of app-to-app-communication, like sending files or links to another application or providing a direct communication channel. While on Windows Phone 8 this type of communication is always handled by the OS, Android offers a direct app-to-app-communication (Franz, 2015). Therefore it is assumed that at least the establishment of the channel is handled by the OS's API.

Though applications need a permission to access WLAN and Internet on Windows Phone and Android, it is not shown on Android on installation time. So the user can never be sure if an application can "call home" or not. Of course there exist applications which need Internet access to work properly just like any social media application, but for other applications, like a QR-scanner, there might be no need at all to access the Internet. In the data flow model there is a differentiation between Internet and WLAN. Internet connection can of course be established by WLAN, but also by mobile Internet technologies like GSM, UMTS or LTE. Actually it is also possible to have WLAN without an Internet connection. On the other hand there are often other accessible devices on the WLAN.

Also a smartphone often has built-in NFC and Bluetooth modules, which can be used by applications. Also the USB interface provides another kind of data-connection. Typically these interfaces need some user interaction to work properly, like having the device close enough to another NFC- or Bluetooth-enabled device.

Applications can interact with the user, typically with the build-in touchscreen to either show information to the user or receive user input. Also the usage of the speakers, microphone or vibrations is possible.

Our data flow model is in some ways simplified, as any application has to use OS APIs to do anything. This means there are special calls to provide a GUI or to establish an Internet connection. For example the OS is the one to establish a TCP connection to a server, which then can be used for communication with the HTTP protocol. But when an application has the correct permissions, those types of connections are seen as "direct" for our purpose. Also we omitted a few data connections (e.g. from other applications to the broker for the gallery or the Internet) as this model is centered around the malicious application. So our model should keep the balance between the reduction of complexity and having enough details for a comprehensive threat analysis.

6.4 Attack vectors for smart phones

Again it is important to clarify the difference between attack vectors and threats. Attack vectors are essentially entry points to an asset, where an asset can be anything like a building, network, device or software. Often it also makes sense to break one asset down to its parts, where each sub-asset can be analyzed individually.

Attack vectors also still exist, even if that entry point is secured in some way. As an example a door is still there, regardless of the quality of the door or its lock, though those measures might make it harder to break in. The same applies to an SSH port on a server. Assuming that the SSH server is properly configured and uses strong cryptography, it still exists and even may face the public Internet.

On the other hand threats are events that could occur, often with unwanted consequences. Thinking again of a door with a proper lock, there still exists the threat that someone can break it. With that there exist other threats. What can be done when the door is open? What can be stolen? What information can be gathered?

With that in mind, essentially any way to send data or commands to the phone or any way to let the phone read data is an attack vector. So the following list might not be complete, especially as there will be new technologies for sure in the future. But also it could happen that attack vectors disappear in the future as technologies become deprecated.

- **Internet** When using the Internet with a smart phone, this typically is done with a browser, like Safari on iOS, Internet Explorer on Windows Phone and Chrome on Android. But this is not the only way, as for example a news reader gets the news from the Internet and can also have an integrated webview. Of course websites can always be used for social engineering attacks against a user, like faking login sites of banks or payment providers. But it was also possible to jailbreak an iPhone just by visiting the website *www.jailbreakme.com*, though that does not work anymore on recent devices with updated software. This example still gives some insight of how far a browser exploit can lead, as it essentially means that by visiting this site, someone was able to gain full access on the device and install arbitrary software with the same high-level privileges.

- **E-mail** Of course Internet does not only mean to use a browser to surf. Another typical use-case is the usage of e-mail on smart phones. Again they can be used for social engineering of users, but also spamming. Another point here is also the fact that e-mail clients sync with the users e-mail provider on a regular base. This means that the user does not have to perform any action to get his latest e-mails, but it also could mean that an exploit is auto-delivered to the client, when it downloads the according e-mail.
- **Applications** As today most applications use the Internet in some way, typically to provide their functionality or to load advertisements, they expose a certain attack surface. For example an attacker could use a man-in-the-middle-attack to modify packets sent to the application and deliver an exploit for it. Also when thinking about instant messengers and other social networking applications, they also potentially provide a way to send messages to other users, which can contain an exploit or again can be used for social engineering.
  But not only the fact that "good" applications may contain exploitable vulnerabilities has to be considered. It is also possible that an attacker was able to circumvent the protection mechanisms of an application store and submit his malicious application, which then may be downloaded by the user. But also the user might be tricked to download the application from a website and install it. With that the attacker is able to run code on the device which can lower the barrier to run further exploits and infect the device.
- **WLAN** The main difference here compared to mobile Internet access, is that the smart phone is part of a local network, in which also other devices may be participating. The phone also gets an IP-address to participate in the network and so packets can be sent to it by other devices. This again can lead to sniffing or replay attacks when insecure protocols are used, but an attacker could also flood the phone with packets to deny or slow down functionality.
  Also there is another thing to consider: is the WLAN trustworthy at all? Due to the fact that any traffic goes through the access point, it can read and manipulate the packets, when they're unencrypted. This even gets more dangerous, when devices were connected to an unencrypted network, like a public hotspot. Mobile devices like smart phones, tablets and also laptops probe for access points to connect to and normally auto-connect to those they remember. This is done by using the SSID of the access point and "asking" whether there is such an access point nearby. While for a protected WLAN the credentials will not fit, an access point could impersonate any other unprotected access point. (Hunt, 2013) demonstrates in his blog post how easy this fact can be exploited with a Pineapple, which automatically impersonates any unprotected access points it is asked for.
- **SMS, MMS** Even technologies like SMS can be used to attack phones. As an example in 2003 there was a vulnerability in a series of Siemens devices, which lead to unstable behavior of the firmware, just by sending some special characters (SecurityFocus, 2003). A recent DoS-attack caused iPhones to

crash and reboot just by sending a special non-Latin string in a message. This worked not only with SMS but with any text messaging application (Gibbs, 2015).

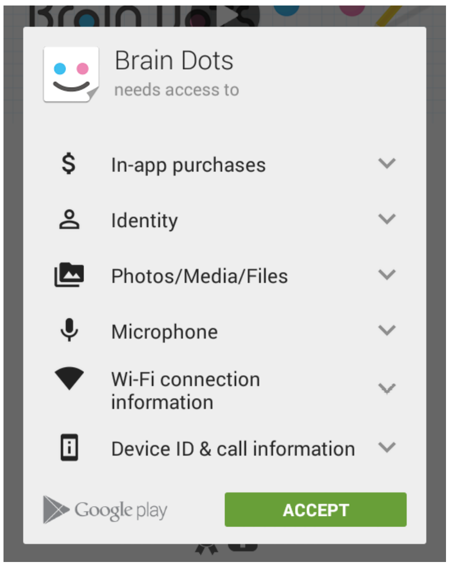
- NFC, Bluetooth, Infrared   Close proximity technologies of course do not allow to attack any phone from any distance, but still they employ a communication channel to other devices. This could be used to connect to the hands-free equipment of a car over Bluetooth or to implement mobile payment systems, where one uses his phone to authorize a payment over NFC. Although an attacker must be nearby his victim, the attack can be performed nearly unseen, for example by standing or sitting close to someone in a train.
- Sensors   Cameras, microphones, GPS, magnetometers and similar can hardly be forced to produce arbitrary input, as they essentially just read their environment. But still it may be possible to produce data which may be misinterpreted or spoof the sensors, e.g. by placing a magnet near the phone to interfere with the magnetometer.
- SD-Card   External or removable storage provide essentially two ways to attack the phone. On one hand the SD-Card can be read by other devices, especially if the data is not encrypted. Typically there is media like pictures and videos stored on the SD-Card, but it is also possible that applications may store some internal data on it.
  On the other hand this data can be tampered with or malicious files like an infected PDF-file can be placed on the storage, which then can be used to exploit vulnerabilities of the PDF viewer application.
- USB   Using the USB interface on modern smart phones serves multiple purposes, mainly to charge the device and to transfer data from and to it. But often the USB interface offers debug interfaces or low level interfaces, e.g. to flash a new firmware. Also it is used commonly for developing new applications, which can easily be transferred and installed on the device for testing. But that is exactly the problem, as the USB interface exposes protocols and interfaces to the "outside", which can have a deep impact on the phone or its software. The researchers Billy Lau, Yeongjin Jang and Chengyu Song presented at Black Hat USA 2013 a self-crafted malicious charger, which is able to infect an iPhone within 60 seconds if hooked up. (Lau, Jang, & Song, 2013)
- Hardware   Though attacks directly on hardware are quite low level, they still exist. For example an attacker could just remove the internal storage to read it, but this is not always necessary. Some devices offer a JTAG interface, which main purpose is for debugging, but maybe was not deactivated properly by the vendor. For example it is possible to extract the unlock pattern from an HTC Wildfire S with JTAG. (Forensics Wiki, 2012)
- SIM   One might not be aware of it, but SIM cards are capable to do more than just storing key material or subscriber IDs. For example they also can run Java applets or perform encryption operations. So when an attacker is able run arbitrary code on the SIM he can even bypass the OS or actually modify it. German security researcher Karsten Nohl of Security Research Labs was able to break the encryption on SIMs of many vendors as they used weak encryption and then was able to redirect calls, send premium SMS in behalf of the user or read the whole 2G, 3G and 4G traffic. (Donohue, 2013)

This list should make clear that smart phones and tablets expose a big attack surface and also that for different parts and technologies different parties are responsible.

Again to differentiate attack vectors and threats, the before mentioned DoS-attack against iPhones is taken as an example here. The attack vectors were any kind of messages, which caused a pop-up with a message preview on the screen, like an SMS or instant message. In turn causing the iPhone to reboot is a denying of service threat against it.

7 Threat Analysis

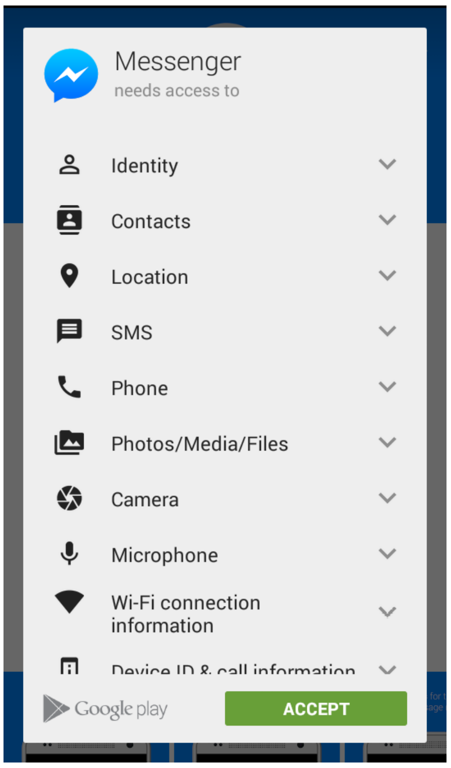
When analyzing the impact of a malicious application, we'll make a difference between applications, which run "legal" code, and those which contain exploit code. This means the former type of applications access the allowed APIs and will not try to actively circumvent any security mechanisms, although typically this type requests an extensive amount of permissions upon installation. The latter type will try to exploit vulnerabilities of other applications or the OS itself. In the worst case the application will get root-access to the system. Therefore we will separate the threats below into low privilege and high privilege. This is intended to give an idea, whether or not the application has to elevate its privileges (e.g. with exploit code) for that threat. Of course there might be differences in the privileges an application can legally get between the different OSs or even versions of the same OS, so a low privilege threat might actually be a high privilege threat or vice-versa.



Brain Dots
needs access to

In-app purchases
Identity
Photos/Media/Files
Microphone
Wi-Fi connection information
Device ID & call information

Figure 8: Messenger likes to have access to essentially all private data on the phone.

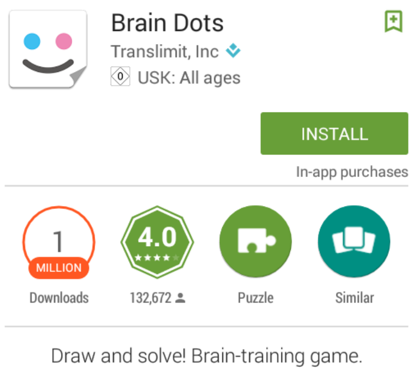


Messenger
needs access to

Identity
Contacts
Location
SMS
Phone
Photos/Media/Files
Camera
Microphone
Wi-Fi connection information
Device ID & call information

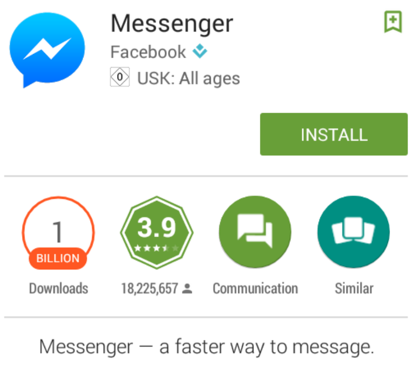
Figure 9: Brain Dots likes to have access to the photo gallery and the microphone.

We also assume that an application can get all its permissions granted from the user upon installation. This will not work for all users, but for many. One might think that users get suspicious regarding applications requesting many permissions, it seems that this is not always the case. Figure 8 to Figure 11 show screenshots of two applications in Google's PlayStore. Both *Brain Dots* and *Messenger* are one of the top applications at the time of writing and have 1 million and 1 billion downloads respectively. When a user likes to install the applications, a list of permissions demanded by the application is presented before it is downloaded and installed. While Messenger seems to request all permissions it can get, especially privacy related, Brain Dots still wants access to the microphone and media gallery. Maybe there are legit reasons for each of the requested permissions, so the applications can do their job, but in most cases there is no explanation for them in the application's description.



Draw and solve! Brain-training game.

Figure 10: Brain Dots: a game with about 1 million downloads



Messenger — a faster way to message.

Figure 11: Messenger: Facebook's instant messenger with 1 billion downloads.

7.1 Processes

Processes are for sure the most threatened types of entities in data flow diagrams, as they typically have a data input and output, but also work on data. There is also the danger of memory corruption to either change the behavior or the data of a process. There is always the threat of elevating privileges to the root level by an exploit, so this will not be mentioned anymore in the following.

7.1.1 Data Interfaces

We have modeled the data interfaces like NFC or Bluetooth as processes, as there is some kind of driver on the device to communicate with. Of course there typically is a receiving end, which can be modeled as an external entity, though this was omitted here, as the application interacts with the driver. Also it should be noted that applications under normal circumstances have very limited or no capabilities to interact with USB.

Of course the OS is sitting between the application and the driver or the driver is part of the OS, so we see them as merged together.

Data interfaces can be used to attack other devices, via Bluetooth, NFC or USB. If the malicious application can use appropriate exploits, it might be able to eavesdrop connections from other applications or even modify the data.

Spoofing

- Low Privilege Assuming the application can freely form the data sent to the interfaces, it could spoof the receiver of the message and for example impersonate a reader for NFC enabled credit cards.
- High Privilege The application could spoof the OS/driver to impersonate another application.

Tampering

- High Privilege The data in the buffers or the code of drivers could be modified to change their behavior.

Repudiation

- Low Privilege When the receiver can be spoofed, it logs a wrong sender identity.
- High Privilege Logs from the driver or OS on the device can be changed or erased.

Information Disclosure

- High Privilege Buffers can be read directly or the drivers modified to dump data on the disk, e.g. to read communication from other applications.

Denial of Service

- Low Privilege The application can repeatedly send lots of data over an interface to either block the interface on the device or the receiver. The data rate could be limited by the OS.
- High Privilege Send data without rate limitation. An exploit in the driver could force the device to freeze or reboot.

Elevation of Privileges

- Low Privilege   When the receiver is spoofed, the application might elevate its privileges there.
- High Privilege   Bypass limitations or gain low level access to the hardware. Also run commands or processes in the driver's context.

7.1.2 Other Applications

Depending on the OS, applications have different ways to communicate with each other. For example on Android exists the possibility to directly communicate with other applications, while on Windows Phone the OS acts as a proxy and handles the communication. It is also important to note, that parts of the OS can also be applications just as downloaded or pre-installed ones.

Attackers might be interested in data other applications hold, like chat logs, contacts or even passwords or private keys. If the user reuses passwords on other websites or services, they are susceptible for an attack. Also data tampering could be used for further attacks, like social engineering against the user.

Spoofing

- Low Privilege   Communicate with other application with a fake identity. Also register for links to open with malicious application, e.g. for `fb://*` to impersonate the Facebook application.
- High Privilege   Modify the OS/API-Calls to fake the malicious application's identity, e.g. when establishing an app-to-app-communication.

Tampering

- Low Privilege   Files in public folders could be changed, like on the SD-Card or in the gallery.
- High Privilege   Files in another application's folder structure can be modified, but also the memory of processes can be corrupted. Also the malicious application might inject code or data into other applications to change behavior, which can be used for further attacks, e.g. against the user.

Repudiation

- High Privilege   Modifying logs from the application or the OS.

Information Disclosure

- Low Privilege   Files of other applications in public folders can be read, like on the SD-Card or in the gallery. Depending on the OS: enumeration of other installed applications. Also those information can be sent to the Internet.
- High Privilege   Files in other application's folder structure or even the processes' memory can be read, which could lead to disclosure of passwords or security certificates.

Denial of Service

- **Low Privilege** If not limited by the OS: intense usage of CPU or memory to slow down other applications.
- **High Privilege** Run exploits on other applications to freeze or crash them.

Elevation of Privileges

- **Low Privilege** Elevate privileges on other applications by spoofing the malicious applications identity.
- **High Privilege** Run exploits to elevate privileges on other applications.

7.1.3 OS / API

In our diagram this process is very condensed, although it is quite an integral part. A few typical tasks the OS has to fulfill is to manage system resources like CPU and memory and also acting as layer between software and hardware, while offering an API. Also runtime environments like the Dalvik VM on Android or Windows' .NET-Framework can be seen as a part of the OS. This means that the OS and programs working closely together with it, exposure a certain attack surface, which could lead to critical vulnerabilities.

In general the OS is involved in most threats and attacks on the device. By attacking the OS, the malicious application might overcome the sandboxes or gain advanced permissions in any other way. In the worst case it gets administrative access. Also the malicious application may circumvent exploit mitigation techniques and is able to infect the device with malware, like viruses or worms.

Spoofing

- **Low Privilege** The malicious application could be named like another one to spoof its identity, although the developer is known because of either the submission process or the signature of the developer certificate.
- **High Privilege** If the credentials or certificate of the faked application's developer can be stolen, it is possible to submit an application in his behalf. Though this is mainly an attack against the developer, it could lead to an automatic update of the application, because the OS thinks it comes from the attacked developer.

Tampering

- **High Privilege** Changing buffers, code or other data in memory, typically by using an exploit.

Repudiation

- **Low Privilege** Depending on the detail of the log, the application can fake its name and therefore produces false logs.

- High Privilege Further access to logs, like changing or deleting logs.

Information Disclosure

- Low Privilege The following data can often be accessed, but maybe needs the user's permission: OS- and API-versions, battery-state, connectivity-information, device ID, International Mobile Equipment Identity (IMEI), and accounts on the device.
- High Privilege Cached credentials, contents of buffers or other memory sections may be accessed.

Denial of Service

- Low Privilege If not limited by the OS: intense usage of CPU or memory to slow down the OS or drain the battery.
- High Privilege Run exploits on OS to freeze or crash it, circumvent CPU and memory limits for the malicious application or reduce memory and CPU time available for other applications.

Elevation of Privileges

- High Privilege Change user of the malicious application to gain more privileges. Run commands or other processes with elevated privileges.

Download and installation of software

- Low Privilege The malicious application may be connected to a command-and-control-server and download and install additional software. Depending on the OS the application may not be allowed to install other applications or software or at least the user is asked for permission.
- High Privilege The malicious application may use an exploit to perform installation without notifying the user.

7.1.4 Sensors

Sensors like microphones, cameras, gyroscopes and GPS are very common in today's smart phones. Again like with the data interfaces, we assume that the access to these sensors is done by a driver, which is mostly accessible via an API. In the model the data flow is also just one way, as the sensors just output data, although it might be possible to pass tuning parameters to the driver for some sensors. The latter should be handled by the OS and would require elevated privileges to do so for sandboxed applications.

The microphone and camera as well as GPS offer the possibility to spy on the user, especially if the application is able to activate them without letting the user know.

Spoofing

Faking the malicious application's identity shouldn't be much of a point here, as we assume there is no kind of authentication by the driver and also again the driver has typically no input. Controlling the access to the drivers API is also a task of the OS.

Tampering

- **High Privilege** The data read from the sensors could be modified while in a buffer or the tuning parameters might be changed to falsify the data. This could be used to mislead other applications relying on the data, like a navigator application.

Repudiation

- **Low Privilege** Depending on the detail of the log, the application can fake its name and therefore produces false logs.
- **High Privilege** Further access to logs, like changing or deleting logs.

Information Disclosure

- **Low Privilege** Taking photos with the camera or recording sound from the microphone. Reading of the devices position and movement. This could also be used to track the user and derive further information from that, like his working place and home. Accessing those sensors mostly is notified to the user.
- **High Privilege** Accessing the sensors without notifying the user.

Denial of Service

- **Low Privilege** If not limited by the OS: intense usage of sensors to drain the battery.
- **High Privilege** Run exploits on the driver to freeze or crash it. This could force the device to reboot.

Elevation of Privileges

- **High Privilege** Bypass limitations or gain low level access to the hardware. Also run commands or processes in the driver's context.

7.1.5 Broker for file system / media library / contacts

Although brokers which control access to the file system are a part of the OS, they are separated here. There are different types of brokers, controlling direct access to the file system like the SD-card or offering interfaces for contacts-database, media galleries or user dialogs to open a file. With that they often offer some kind of abstraction level, like hiding the real file location in the media gallery.

As applications can get access to certain parts of the file system, media libraries and contacts database by requesting the according permissions, those may be subject to information leakage threats. The user's contacts can be used to attack further users.

Spoofing

- **Low Privilege** The malicious application can use a fake name to partially fake its identity. Again the developer is known through the submission process or the developer's signature.
- **High Privilege** Further abilities to fake identity, like modifying function calls to change parameters, like the application ID or similar.

Tampering

- **Low Privilege** Change, delete or add files in the gallery, on SD-card or other publicly writable directories. Change, delete or add contacts.
- **High Privilege** Change, delete or add files which are normally not writable or without the application requesting the proper permission. Modify buffered files.

Repudiation

- **Low Privilege** Depending on the detail of the log, the application can fake its name and therefore produces false logs.
- **High Privilege** Further access to logs, like changing or deleting logs.

Information Disclosure

- **Low Privilege** Reading files, filenames or file metadata from the gallery, SD-card or other publicly readable directories. Read contacts.
- **High Privilege** Reading files, filenames or file metadata of normally unreadable files or without properly requested permission.

Denial of Service

- **Low Privilege** Deletion of files or writing huge files on the disk to slow down the system and fill space on the disk. This could be limited by the OS.
- **High Privilege** Bypass limitations for writing files. Delete protected files, which could lead to system crashes.

Elevation of Privileges

- **High Privilege** Exploit broker to gain access to protected files or get direct access to the disk.

7.1.6 SMS/MMS services

SMS- and MMS-services on smart phones consists mainly of two parts: a part of the OS, which handles the low level communication and is responsible for actually sending and receiving SMS- and MMS-messages and an application which the user interacts with to view and compose new messages. For our purpose we assume that the malicious application is communicating directly with the OS.

One of the most costly threats for the user is the fact, that a malicious application can send SMS messages to premium services. Sometimes this can be blocked or limited by either the OS or the mobile communication provider. Also access to the SMS database can be used by the malicious application to intercept messages containing one time authentication codes, such as transaction numbers (TANs) for online banking.

Spoofing

- Low Privilege The application can use a fake name to partially fake its identity.
- High Privilege Further abilities to fake identity, like modifying function calls to change parameters, like the application ID or similar.

Tampering

- Low Privilege Change messages in the database.
- High Privilege Change or add messages in the database or in buffers.

Repudiation

- Low Privilege By deleting messages from the database, the user may not notice incoming or outgoing messages.
- High Privilege Further access to logs, like changing or deleting logs.

Information Disclosure

- Low Privilege Reading messages from the database.

Denial of Service

- Low Privilege Sending a lot of messages to prevent or slow down sending messages from other applications and the user. This could be limited by the OS.
- High Privilege Bypass limitations for sending messages.

Elevation of Privileges

- **High Privilege** Exploit services to gain low level access to carrier.

Sending of Premium SMS

- **Low Privilege** Sending of SMS messages to premium numbers could cause high costs for the owner of the SIM-card. The OS could limit these or ask the user for permission.
- **High Privilege** Bypass limitations for sending messages or asking for the user's permission.

7.2 Data Store

Data stores have to be seen as a "raw" medium. Taking a database as an example, the database itself is typically just a file on a disk, so it is just data, without any authentication or authorization. Enforcing file permissions on file system level is typically a task of the kernel. Also there might be a proper access control implemented in brokers or similar processes. This can be problematic if the file systems permissions are inconsistent to the permissions such a process enforces. On one hand this can lead to instabilities and problems, when the file system permissions are too restrictive and denies access to files the process needs to work properly. On the other hand, if the file system permissions are too loose, an attacker might exploit a broker process or circumvent it for direct access and may get access to files neither of them should have. There might be file systems, which maintain an access-log, so repudiation attacks might also appear.

7.2.1 Internal Storage

Typically the boot-loader, the OS and most applications are installed on the internal storage. Also there are often public accessible folders like the gallery, but most of the files and folders are protected, due to the criticality of the containing data.

As the internal storage holds the OS and the boot-loader, it imposes an interesting target for attackers. Full access on the internal storage might lead to further infection of the system. Additionally the internal storage can hold user data, which are subject for information disclosure threats.

Tampering

- **Low Privilege** Change or add files in the gallery or other public writable directories.
- **High Privilege** Change or add files anywhere. This could also include temporary files, logs, passwords or keys and files protected by obscurity.

Repudiation

- **High Privilege** Change logs if it is a logging file system.

Information Disclosure

- **Low Privilege** Reading files, filenames or file metadata from the gallery or other publicly readable directories.
- **High Privilege** Reading files, filenames or file metadata from anywhere. This could also include temporary files, logs, passwords or keys and files protected by obscurity.

Denial of Service

- **Low Privilege** Deletion of files or writing huge files on the disk to slow down the system and fill space on the disk. This could be limited by the OS.
- **High Privilege** Bypass limitations for writing files.

7.2.2 External Storage

Often the external storage has a weaker protection, like on Windows Phone and Android, where applications can get full access to it via one permission. In the case of Windows Phone applications can be installed on the external storage to a hidden and encrypted partition. Similarly Android encrypts the APK-file on the external storage, but keeps the applications data on the internal storage. (Android Developers, n.d.) The external storage can hold user data, which typically is not protected. Also on the major OSs applications are able to get full access on the SD-card with just one permission. This can easily lead to information leakage or tampering of the data.

Tampering

- **Low Privilege** Change or add files on the SD-card.
- **High Privilege** Change or add files anywhere on the external storage, which could also include protected partitions.

Repudiation

- **High Privilege** Change logs if it is a logging file system.

Information Disclosure

- **Low Privilege** Reading files, filenames or file metadata from the SD-card.

- **High Privilege** Reading files, filenames or file metadata from anywhere of the external storage, which could also include protected partitions.

Denial of Service

- **Low Privilege** Deletion of files or writing huge files on the disk to slow down the system and fill space on the disk. This could be limited by the OS.
- **High Privilege** Bypass limitations for writing files.

7.3 External Entities

External entities see the device as a whole and not each particular process, as in our scenario communication is done via the network interface or the user interface. We found that in our case external entities are also subject to denial-of-service- and repudiation-attacks, as opposed to the suggestion from (Shostack, 2014).

7.3.1 Internet

From our perspective Internet as a general term stands for devices, which are reachable from the Internet. Also those devices may not only be victims of attacks, but maybe are also owned by the attacker, like a command-and-control-server. It has to be noted, that Windows Phone applications can use sockets to implement own communication protocols on top of UDP or TCP. (Microsoft, n.d.-g)

In general the device can be used to take part in a botnet, which can be used to run Denial of Service attacks against Internet facing servers. Also the malicious application may run exploits on servers and web applications.

Spoofing

- **Low Privilege** Spoofing of the user id, when communicating with servers and services. The IP may be hard to spoof, but may also be obfuscated if the device is behind a router with NAT (Network Address Translation).
- **High Privilege** With low level access to the network interfaces packets may be formed arbitrary and the IP-address may be spoofed. Also a proxy may be used to spoof or hide the device's real IP-address.

Repudiation

With the above mentioned spoofing techniques (or any other), the target may create false logs.

Information Disclosure

- **Low Privilege** The Internet may be used to disclose any information to a command-and-control-server. Also Internet-accessible devices may be scanned.
- **High Privilege** With low level access to the network interfaces targets may be scanned more comprehensively with different techniques.

Denial of Service

- **Low Privilege** The malicious application may participate in a distributed denial-of-service-attack. Bandwidth may be limited by the OS.
- **High Privilege** Bypass bandwidth limitations.

Threatening of other devices

The malicious application may use different techniques to run exploits against other devices. While spreading of the malware is a typical goal, any of the STRIDE-threats may be imposed to them. As an example exploits against a forum software may be run to gain administrative access and leak or change information or even shut down the service completely. Whether the malicious application needs low or high privileges on the local device depends on the exploit it is trying to run. SQL-injection attacks may be performed with simple HTTP-requests, while others may need specially crafted IP-packets.

7.3.2 WLAN

There is quite some similarity to the Internet, though in a local network there might be devices which would normally not be accessible from the Internet, e.g. due to a firewall. Especially in corporate networks this can lead to attacks against internal servers or other critical devices, which normally are protected by firewalls. Of course the impact here depends on the implementation of the network, but even in a guest network other devices may be attacked.

Spoofing

- **Low Privilege** Spoofing of the user id, when communicating with servers and services.
- **High Privilege** With low level access to the network interfaces packets may be formed arbitrary and the IP-address may be spoofed.

Repudiation

With the above mentioned spoofing techniques (or any other), the target may create false logs.

Information Disclosure

- **Low Privilege** Locally accessible devices may be scanned. If the attacker is an insider, the malicious application might disclose information to a command-and-control-server located in the local network.
- **High Privilege** With low level access to the network interfaces targets may be scanned more comprehensively with different techniques.

Denial of Service

- **Low Privilege** The malicious application may participate in a distributed denial-of-service-attack or flood other local devices with packets. Bandwidth may be limited by the OS.
- **High Privilege** Bypass bandwidth limitations.

Threatening of other devices

Again similar to the Internet the malicious application may attack other devices to spread the malware or impose other threats. Depending on the implementation of the network, more critical devices and services may be reachable and maybe are less secured against attacks from the internal network.

7.3.3 User

As the communication with the user takes place via (touch-)screen and buttons, the malicious application has to have a UI to interact with.

Social engineering attacks against the user often have the goal to disclose sensitive information, like login credentials for payment providers or credit card data. Also the user may be hold to ransom by denying access to the device or the data on the device.

Spoofing

- **Low Privilege** The malicious application could fake another applications UI and name. Also the user may be tricked to perform certain actions, e.g. installing an application due to "security reasons".
- **High Privilege** The application may change links, so that the fake application is opened instead of the real one.

Information Disclosure

- **Low Privilege** The user can be tricked to pass sensitive information to the application. The user may be spied on with the device's camera, microphone or even GPS-tracking.
- **High Privilege** Activation of the camera, microphone or GPS is not notified to the user.

Holding to ransom

- **Low Privilege** Showing pop-ups to trick the user to pay ransom, e.g. because the device is/will be encrypted. Encrypting of files in public folders may actually happen, which could hold pictures and other documents. Also the user may be blackmailed with private information collected beforehand, like secretly taken pictures.
- **High Privilege** Files in non-public folders may be encrypted and also the screen may be locked to hold the user to ransom.

Inexperienced Users

We want to mention inexperienced users as a separate threat here, although the threats do not really differ from the above. But typically inexperienced users are more susceptible to threats and attacks including social engineering. So they may be tricked more easily into entering sensitive information or installing Trojan Horses.

7.4 Data Flows

In our model the data flows are mostly internal, so data in transit is hard to manipulate or eavesdrop. To do this an attacker must either hook with functions to manipulate input and output or directly access the data in memory. We assume data on the device's wires to be not accessible by software. With that we will not go through any connection, but take a look the general internal data flow.

Tampering

- **High Privilege** Data can be changed directly in memory or by hooking a function.

Information Disclosure

- **High Privilege** Data can be read directly from memory or by hooking a function.

Denial of Service

- **High Privilege** Such an attack might work if the RAM is accessed in parallel e.g. with a multi-core processor. So when the malicious application is running on one core it might flood the RAM with cache-inefficient accesses. This could slow down the memory access for the other cores, i.e. other processes. The same principle could also work even better for permanent memory like hard disks or solid state drives, as they are generally slower than the main memory.

7.5 Sandboxing techniques

We were talking about sandboxing techniques in chapter 3 and chapter 5, but they are not yet reflected in the threat analysis. Regarding our data flow diagram, of course those techniques belong to the OS, which is responsible to enforce the sandbox. So any of the STRIDE threats would put the sandboxes in danger, but we want to take a look at the mechanisms separately.

7.5.1 Full disk encryption

When full disk encryption is enabled, this means that either complete disks or partitions are encrypted on block level. At least a small part must be unencrypted for the boot and decryption process of the other partitions, though. The user provides the password for en- and decryption during boot. While the device is on, the key resides in memory and the encryption typically is transparent to processes.

Although full disk encryption does not help to enforce sandboxes on the running device, an attacker with physical access to the disk will not be able to read the data. So a malicious application running on the system has the same access, as if the full disk encryption is not in place.

Tampering

As the en- and decryption is transparent to processes, there is no difference for them, when accessing files. Files can easily be changed, when an application has access to it.

Information Disclosure

Reading of files also only depends on the access rights of a process, due to the transparent encryption. An advanced attack could extract the encryption key from memory.

Denial of Service

As en- and decryption needs a certain amount of processing power, a malicious application, which reads and writes a lot of data, might slow down the file access for other applications. The effectiveness may be different if a separate crypto-processor is in place.

An advanced attack could be directly on the encryption key in memory. If this key would be changed, any reading and decryption of data would fail, while writing and encryption of data would happen with a wrong key. It is very likely that the system would crash in this case, as critical data cannot be read anymore. Also this can leave the file system in an inconsistent state, as some data would be unreadable due to the encryption with a different key.

7.5.2 File based encryption

Encrypting files separately is possible independently from full disk encryption, as this just adds another encryption layer. It is possible to use different keys for different files, similar as it happens in iOS. With that the respective keys do not have to be in memory all the time and the user may be asked for the password when needed.

To support the sandboxing of applications, the data of each application could be encrypted with a separate key and only made readable by the respective application.

Of course file based encryption will not protect a file from destructive tampering, as an attacker could overwrite the file with random data. If an attacker is able to extract the keys for the respective files, he can decrypt them and also re-encrypt data to modify files, but keep them readable.

Tampering

To write meaningful data to the file, the attacker either may extract the key of that file or exploit the process handling the keys and encryption to write data to the file.

Repudiation

If the process handling the keys and encryption produces logs, they may be a subject for attacks.

Information Disclosure

Depending on the implementation, the filename may be not encrypted and readable. To extract information from the file, the attacker must either obtain the encryption key or exploit the process handling the keys and encryption to read the file.

Denial of Service

Encrypted files can directly be overwritten with different data, which would destroy the file, as the decryption of the file would produce random looking data. As en- and decryption needs a certain amount of processing power, a malicious application which reads and writes a lot of data might slow down the file access for other applications. The effectiveness may be different if a separate crypto-processor is in place.

Elevation of privileges

The process for handling keys and encryption might be exploited to gain further access to protected files.

7.5.3 File based permissions

This is quite a simple, but also powerful concept. The access to each file is controlled by either ACLs or Unix-like with owner and groups the file belongs to. As stated in Chapter 5 Android creates a new user for each application and enforces file access with users and groups.

Inconsistent file permissions can lead to further access than an attacker should have, if he is able to run own code. Also attackers might gain access by either changing the file permissions or run code in a different user context.

Tampering

A malicious application could change the permissions a file has set. This could be an allowed operation, e.g. when the file belongs to the application, but it may use an exploit to run a forbidden operation.

Repudiation

Depending whether or not there is a file access log created, it might be a subject for attacks.

Information Disclosure

File permissions can be an indicator for the criticality of files and folders. Also it may be possible to enumerate other users of the system.

Denial of Service

By changing file permissions, processes might lose access to files, which where belonging to them. This could cause instability on the system.

Elevation of Privileges

The malicious application may try to run itself as a different user to elevate privileges on file access.

7.5.4 Mandatory access control

MAC is an independent security control. Essentially MAC adds another layer of access control, which can be implemented in different ways. We will use *security attribute* as a general term in the following, which is used for Mandatory Access Control. For example Windows uses integrity levels as security attribute and processes can only access files or services with the same or lower integrity level, even if it would have access by file based permissions. iOS uses MAC to separate applications from each other, although all applications run with the same user.

Sometimes MAC may look similar to file based permissions, as in the iOS example the result might be similar, if each process runs with a separate user and proper file permissions. But it is also possible that MAC will not help to isolate applications from each other, but more trustworthy system parts from less trustworthy. Assuming that each application on Windows Phone runs with a low integrity level, MAC will not prevent them from access each other's files, as they all have the low integrity level, but it keeps applications from accessing files with a higher integrity level.

As stated MAC might not necessarily help to enforce the sandbox between the applications. Changing security attributes of resources or running code with a different security context can be a goal of an attack to elevate privileges.

Tampering

Security attributes of resources like files or services, may be changed. This can be an allowed operation, if the malicious application has the permission to do so.

Repudiation

Depending whether or not there is an access log created, it might be a subject for attacks.

Information Disclosure

Security attributes of resources like files or services can be an indicator for their criticality.

Denial of Service

Security attributes of resources could be changed to restrict access for processes needing access to those resources. This could cause instability on the system.

Elevation of Privileges

The malicious application may try to run itself with different security attributes to elevate privileges.

7.6 Risk analysis

After the threats have been evaluated, the next step would be a risk analysis. It takes the environment, the security measures and potential losses into account to find some value to describe the risk. This could be a monetary value, but often it is hard to measure everything with money. So often a scale is applied, which for example could go from 0 to 10, where 0 means "no impact" and 10 means "damage seriously threatening the company's existence".

An example could be an Internet company, where the customer database has been leaked. There are some costs which are quite easy to express, like paying an external incident response team or maybe paying a fine due to law violations. But there are also factors which are quite uncertain and hard to express a monetary value for, like the company's reputation, which could lead to customers leaving or making it harder to acquire new customers.

Threats always exist, but for a risk analysis the likelihood of them appearing have to be taken into account. And that is very dependent on the company's environment. What are the attack vectors? How far will an attacker come with what effort? And again: what are the implications of an attacker having success?

Sometimes it is even hard to correctly estimate the value of a certain asset. A good example is the PowerSpy attack developed by (Michalevsky, Nakibly, Schulman, & Boneh, 2015). The basic idea is to analyze the power usage of the phone to estimate the phone's location. The power usage is influenced by the distance to the base station the phone is connected to, but also by obstacles between them, such as buildings. The problem is, that the battery state and power usage of a phone would mostly be considered as uncritical data, but it seems otherwise.

The point here for us is, that we simply cannot perform a full risk analysis as this has to be done for each environment separately. Also a good risk analysis would need a defined approach and structure, which maybe needs some adaption to the case. Still there are threats which might have a bigger influence than others. We will consider a mobile device used privately and mobile devices used in a corporate environment in the following and take a look at a few threats we found earlier.

7.6.1 Threats against privately used devices

It makes sense to take a look at the assets a private smart phone has, which is mostly user data. Interesting data are photos, movies, messages and contacts, which may also be derived from any social media application. But also data like the location or records from the microphone or camera. The phone itself might also be an interesting target for usage in a bot network.

- **Information disclosure from sensors** Typically it is not much of a problem to get the permission to activate the camera or microphone or use location services, as the mobile OSs typically offer according permissions. But the OS might inform the user when the camera is activated or GPS tracking is on. This will get dangerous if an application can circumvent this notification by the OS and activate those features without letting the user know.
- **Information disclosure of media and contacts** As stated at the beginning of this chapter, users sometimes do not seem to care much about the permissions applications ask for. An attacker could

easily build an application, which maybe seems to have some legit reasons to access certain data, and will surely have a good chance that users may install it. Such an application would be able to read and write media files, but also contacts. The latter can also be used to social engineer the user's family and friends, as the attacked user certainly has a higher level of trust than a stranger.

- **Information disclosure against the user** A prominent target of phishing attacks is payment information like login credentials for payment providers or credit card data. Such data could then be sold by the attacker on black markets or directly used by himself.
- **Threats against local networks** The impact of such threats is dependent on the goal and capability of the attacker. It might be possible to compromise more private devices, which for example could support phishing attacks.

In general most direct threats against users are privacy related and as mentioned, payment data is also a common goal of attacks. But there are also indirect threats against a person. For example when devices take part in a botnet, it might happen that the owner is brought to justice.

7.6.2 Threats against devices used in corporate environments

We assume that on devices in corporate environments there is a minimal amount of private user data. This might not be realistic, but we considered private user data in the previous section. It might also be fair to say, that private contacts in addition to business contacts are stored on the phone.

- **Threats against internal networks** Mobile devices like phones or tablets with a connection to the internal network, pose an increased threat against that. They maybe sit behind the firewall and are able to access internal servers. Those servers could be inaccessible at all from the outside or there are fewer security controls enabled for the internal network, than from the Internet. So an attacker might have fewer security controls to overcome.
- **Tampering and information leakage of corporate data** If the phone holds corporate data, it could leak information to an attacker and also the data may be tampered with. Depending on the implementation, this data may be synced back to internal data storages.
- **Threats against corporate applications** If the corporate uses special mobile applications to offer services to the user, this applications could be subjects for attacks. Those applications may be attacked directly to leak information or tamper with data. But they also could be manipulated to gain further access to internal systems.
- **Information disclosure from sensors** Similar to the private case, camera and microphone could be used for espionage, like overhearing confidential conversations.

To actually get a proper overview of threats a compromised phone imposes, it makes sense to analyze the corporate environment. Assuming that at least one device is compromised and an attacker has full control over it, can help to find threatened assets in a corporate environment and find possible mitigations. It is important to not only take a look at the phone of a random employee, who might have reduced access rights, but also consider the case that the phone from an administrator or the CEO is compromised.

8 Hardening

Hardening devices and their software can help to reduce their attack surface and mitigate risks. Typical hardening steps are the proper configuration of software and also removal of unneeded software to remove attack vectors. The following hardening steps and practices mainly focus on mitigating the risks from malicious applications.

8.1 End user

Some of the most profitable attacks against users are the extraction of payment information and credentials for payment services and also holding the user for ransom. The latter can either be done by locking down the device until a fee is payed or maybe extract private information and blackmail the user.

Unfortunately at the time of writing mobile OSs offer very few security relevant settings to change, so it is hard for advanced users to harden the device in depth.

- **Usage of the vendor's application store** As seen in Chapter 3 Microsoft is reviewing and scanning applications during the submission process, but also Google and Apple have a similar procedure. While those scanners might not be perfect, they at least offer some level of security, so users can be quite sure that applications in the respective stores contain no malware. Even if malware gets into one of the stores, it can be reported and is typically removed quickly be the vendor. In contrast third party applications store may not have such controls and some are specially targeted at hosting pirated software, which also often contains malware.
- **Rooting/Jailbreaking** Due to code-signing in iOS and Windows Phone OS the user cannot install applications from untrusted sources. So sometimes users jailbreak or root their devices to either install such applications or to get full control over the device. To achieve this security mechanisms like code signing are disabled and also potentially malicious applications can break out of their sandbox.
- **Minimal-system-principle** The idea here is to run a system with a minimum set of software to reduce the attack surface. This also may increase performance as fewer background processes may be running and also disk space is kept free. On Android devices this is often not so easy, as hardware vendors include already installed applications, which cannot be fully removed. So once in a while end users should go through their applications and remove them if not needed anymore.
- **Data reduction** This principle is quite similar to the minimal system, but with focus on data. So users should reduce the data, which could be leaked from the device. Moving private pictures and videos to another storage, or deleting old messages in chats could be such measures. Some messaging applications offer an option to automatically delete messages older than a chosen time span.

- **Review permissions** Spyware and other malicious applications often try to fake their appearance to trick the user to install it. But they often request more permissions than the original application would need. So a user can use this as an indicator. But also in general applications with many permissions can lead to information leakage.
  iOS and Windows Phone 8.1 (with Update 2) enable the user to control the access on privacy related data. This can be changed in the settings.

- **Installing updates** Keeping a system up-to-date helps to reduce the attack surface and mitigate threats. So at least security updates should be applied as soon as possible.

- **Disable debugging and development features** Mobile OSs typically offer some kind of development features, which for example enable the developer to install applications via the USB connection and gather debug information. Although those features are disabled by default, they should be kept disabled.

- **Antivirus software** Due to the closed nature of Windows Phone and iOS an antivirus software cannot work, as the sandbox permits to scan the whole system. They may be able to scan public directories or the SD-card or maybe retrieve the list of installed applications to find malicious applications by name, but the latter is quite unreliable as valid software may have such a name and malware can be named differently each time. Android has a special interface, that allows antivirus software to scan the APK-file even before installation or updates, but it still cannot scan the other parts of the system or folders from other applications. So it may prevent the user from installing a malicious application.
  Technically developers can use `Intents` to get a notification when applications get installed or updated and are able to scan the according APK-file (Google, n.d.-a).
  Although the capabilities of antivirus software is somewhat limited on mobile OSs, they also often bring some advisories and checklists for security and privacy with them, which may be helpful for the user.

- **Review settings** There are privacy and security related settings, which should be reviewed. For example Google's Chrome can send “Do Not Track”-requests to ask websites to not track the user. In Windows Phone the user can disable the advertising ID, though this might only work for ad-providers using that ID.
  Also device encryption should be enabled. This can be found in the security settings on Android, while -- at the time of writing -- it is not possible for the user on Windows Phone (see also Section 3.3). On iOS encryption is always on, but the key can and should be protected by a PIN. (Mobile Device Security, n.d.)
  To further control access to the devices storage via USB, on Windows Phone the user can set an “Ask before USB-connection”-option. On Android mounting the device as a storage can be fully disabled and enabled only when needed.

8.2 Application developer

Developing secure applications is a separate and considerably large topic, so here only a few brief hints and suggestions can be given. Also the focus here is on developing an application on a mobile OS and to protect it against other malicious software on the device.

- **Secure Coding** Practical guidelines strongly depend on the used programming language and in some parts also on the used environment. So a developer should read and follow appropriate guidelines, to create more robust and secure applications.
- **Input Validation** A developer should always keep in mind, that data coming outside of a system may be malformed. The definition of a system might vary here, as it could be a single program, a server or even a network. The question is about the boundary or the trust border. For example a program may receive input from the user or other programs, but an application server may have an isolated communication line with a database server.
  The point here is to always check (user) input. In some cases it makes sense to sanitize the input, e.g. when storing an HTML-document in a database, but when possible a stricter approach should be chosen: if the input is different than expected, an appropriate error should be presented and the input be ignored. This of course can be quite complex, e.g. when the input may be a (valid) xml-document, but it can also be as simple as ignoring anything that does not look like a number.
- **Enabling of Exploit Mitigations Features** As we saw in Chapter 3 Windows Phone brings a lot of exploit mitigation features, which help to mitigate security bugs. And also the other mobile, desktop and server OSs bring exploit mitigation features with them, which should be used. Typically there is rarely any impact to the code, but some compiler options must be enabled. It makes sense to explicitly enable those features, as they may not be always be enabled by default on different OSs or compilers.
- **Encryption** Encrypting the applications data, especially user data, may help to protect the user from harm. So in addition to getting access to the application's directory, the attacker must also be able to decrypt data. If the user must provide a PIN or password to access the application, this can be used to protect the encryption key, though this may have a negative usability impact. If the user cannot be asked for a password or does not want to set one, it becomes quite hard to hide the encryption key, as it can be found in the code or the file system by a sophisticated attacker.

8.3 Corporate environment

While privately used mobile devices only impact the user's security and privacy, they may have a security impact in a corporate environment. They have to be treated different than workstations as they may leave the corporate network and do not sit behind firewalls and other security appliances all the time.

- **Mobile devices management** The usage of an MDM-solution is highly recommended as it allows the administration to roll out policies to the phones. With that the user can be enforced to use a PIN or password on the lock screen or the device encryption can be enabled, to name a few examples. The available policies and settings depend on the mobile OS and on the MDM-solution.
- **Bring-your-own-device** When employees are allowed to use their private phones for business purposes in the corporate environment it is a good idea to use a solution, which installs an isolated environment on the phone. Applications like an e-mail-client or web browser can be run in that secured environment to separate corporate and private data. This helps both to prevent information leakage and to mitigate malware spreading as typically there is no communication allowed between the private and corporate section on the devices.
- **Untrusted network for mobile devices** Sometimes employees are allowed to use the corporates WLAN on their private phones for Internet access. This should be an isolated and untrusted network, with no connection to the internal network.
  If mobile devices are used for business, they still should be kept in a separate network segment, so the access to internal systems can be limited more easily, e.g. to only make the e-mail-server and MDM-server reachable from that segment.
- **VPN for mobile devices** Similar to the point above mobile devices might have access to a separate network segment via VPN. With that internal services and applications do not need to be exposed to the public Internet, but still users can reach them from home or with mobile Internet.
  The major mobile OS offer VPN connections rolled out with policies from MDM-solutions.

9 Conclusion

For a general understanding of sandboxing and protection mechanisms for applications we used Windows Phone OS as an example. We investigated the structure of applications for Windows Phone 8.1 and put Windows Phone into context by comparing it with Apple's iOS and Google's Android.

Afterwards we described our methodology for the threat analysis. It is based on STRIDE, with a modification to apply it on each element of a data flow diagram, which is called STRIDE-per-element. Both is described by (Shostack, 2014). The idea of STRIDE-per-element is to focus on typical threats each type of element in the diagram may encounter, though similar to STRIDE, it should not limit the threat analysis to the given threat types. Also our generic data flow diagram for mobile OSs is presented and explained in this chapter. We reduced the complexity of our model compared to a real system, but kept enough detail to use it as a base for a comprehensive threat analysis. With that in mind, it may be necessary to adjust the model to specific scenarios, e.g. when analyzing a specific application. We also presented a list of attack vectors on mobile OSs and also provided some examples of malware and attacks using those attack vectors.

According to STRIDE-per-element our threat analysis is divided into the types of elements appearing in a data flow diagram. In the analysis we assumed that the malicious application does not have root access on the device and considered the threats regarding low and high privileges. High privileges mean those kind of privileges an application cannot get under normal circumstances and has to exploit a vulnerability, as opposed to low privileges, which an application can get after requesting them, e.g. by specifying it in their manifest file. Also typical sandboxing techniques were analyzed. We also discussed the feasibility of performing a full risk analysis. As there is no specific environment given, e.g. a corporate environment, we are not able to perform a full risk analysis. But we discussed a few threats in the context of a privately used phone and in the context of phones used in a corporate environment.

We also presented hints for hardening mobile devices. In general mobile devices have to be treated as a fully capable computers, with a twist that they will not always sit behind a corporate firewall as desktop systems maybe do. Depending on the use case, they may hold both corporate and personal data of the user, which means the user may literally carry sensitive corporate data to the outside. Also non-corporate devices still may hold more private and more intimate data of its user, like chats, videos and pictures, but also movement data via GPS or even medical data, like the heart rate in the case of a smart watch.

10 References

Android Developers. (n.d.). App Install Location. Retrieved from https://developer.android.com/guide/topics/data/install-location.html

Chell, D., Ersamus, T., & Jon, L. (2015). *The Mobile Application Hacker's Handbook*. John Wiley & Sons.

comScore. (n.d.). comScore Reports June 2015 U.S. Smartphone Subscriber Market Share. Retrieved from https://www.comscore.com/Insights/Market-Rankings/comScore-Reports-June-2015-US-Smartphone-Subscriber-Market-Share

Comvalius, R. P. L. (2013). Demystifying AppContainers in Windows 8 (Part I). Retrieved from http://blog.nextxpert.com/2013/01/31/demystifying-appcontainers-in-windows-8-part-i

Cunningham, A. (2015). Our first look at Windows 10 on phones, and Universal Apps for touchscreens. Retrieved from http://arstechnica.com/gadgets/2015/01/our-first-look-at-windows-10-on-phones-and-universal-apps-for-touchscreens/

Donohue, B. (2013, August). Weak Encryption Enables SIM Card Root Attack. Retrieved from https://threatpost.com/weak-encryption-enables-sim-card-root-attack/101557

Fan, X. (2009). {/DYNAMICBASE} and {/NXCOMPAT}. Retrieved from http://blogs.msdn.com/b/vcblog/archive/2009/05/21/dynamicbase-and-nxcompat.aspx

Forensics Wiki. (2012). JTAG HTC Wildfire S. Retrieved from http://forensicswiki.org/wiki/JTAG_HTC_Wildfire_S

Franz, F. (2015). Analysis of App-Communication Mechanisms in Windows Phone 8.1.

Gibbs, S. (2015). iOS bug lets anyone crash your iPhone with a text message. Retrieved from http://www.theguardian.com/technology/2015/may/27/iphone-crash-bug-text-imessage-ios

Google. (n.d.-a). Android API Documentation -- Intent. Retrieved from https://developer.android.com/reference/android/content/Intent.html

Google. (n.d.-b). Security-Enhanced Linux in Android. Retrieved from https://source.android.com/devices/tech/security/selinux/index.html

Hunt, T. (2013, April). The beginners guide to breaking website security with nothing more than a Pineapple. Retrieved from http://www.troyhunt.com/2013/04/the-beginners-guide-to-breaking-website.html

Lau, B., Jang, Y., & Song, C. (2013). Mactans: Injecting Malware into iOS Devices via Malicious Chargers. Retrieved from https://www.blackhat.com/us-13/briefings.html#Lau

Li, L., Just, J. E., & Sekar, R. (2006). Address-space randomization for windows systems. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual* (pp. 329–338).

Litchfield, D. (2003). Defeating the stack based buffer overflow prevention mechanism of microsoft windows 2003 server.

Meier, J. D., Mackman, A., Dunner, M., Vasireddy, S., Escamilla, R., & Murukan, A. (2003). *Improving web application security: threats and countermeasures*. Microsoft Redmond, WA.

Michalevsky, Y., Nakibly, G., Schulman, A., & Boneh, D. (2015). PowerSpy: Location Tracking using Mobile Device Power Analysis. *arXiv Preprint arXiv:1502.03182*.

Microsoft. (n.d.-a). MSDN -- App data storage. Retrieved from https://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh464917.aspx

Microsoft. (n.d.-b). MSDN -- Certify your app. Retrieved from https://msdn.microsoft.com/en-us/library/windows/apps/hh694079.aspx

Microsoft. (n.d.-c). MSDN -- Create your first app. Retrieved from https://msdn.microsoft.com/library/windows/apps/bg124288.aspx

Microsoft. (n.d.-d). MSDN -- Supporting your app with background tasks (XAML). Retrieved from https://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh977056.aspx

Microsoft. (n.d.-e). MSDN -- Technical certification requirements for Windows Phone. Retrieved from https://msdn.microsoft.com/library/windows/apps/hh184840%28v=vs.105%29.aspx

Microsoft. (n.d.-f). MSDN -- Windows App Certification Kit tests for Windows Phone. Retrieved from https://msdn.microsoft.com/library/windows/apps/dn629257.aspx

Microsoft. (n.d.-g). MSDN -- Windows.Networking.Sockets namespace. Retrieved from https://msdn.microsoft.com/en-us/library/windows/apps/windows.networking.sockets.aspx

Microsoft. (n.d.-h). What is the AppData folder? Retrieved from http://windows.microsoft.com/en-us/windows-8/what-appdata-folder

Microsoft. (n.d.-i). Windows Dev Center -- Secure boot and device encryption overview. Retrieved from https://dev.windowsphone.com/en-US/OEM/docs/Phone_Bring-Up/Secure_boot_and_device_encryption_overview

Microsoft. (2014, April). Windows Phone 8.1 Security Overview.

Mobile Device Security. (n.d.). iOS -- Encryption. Retrieved from http://www.mobilepolicies.com/ios/ios-encryption/

Pietrek, M. (1997). A {C}rash {C}ourse on the {D}epths of {W}in32 {S}tructured {E}xception {H}andling. Retrieved from https://www.microsoft.com/msj/0197/exception/exception.aspx

SecurityFocus. (2003). Siemens M Series SMS DoS Vulnerability. Retrieved from http://www.securityfocus.com/bid/7004/info

Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.

Sotirov, A., & Dowd, M. (2008). Bypassing browser memory protections. *Black Hat USA*.

Valasek, C., & Mandt, T. (2012). Windows 8 heap internals. *Black Hat USA*.

Vasiliu, A. (n.d.). Windows Phone 8.1 Update 2 new features uncovered. Retrieved from http://www.pocketmeta.com/windows-phone-8-1-update-2-new-features-uncovered-21731/