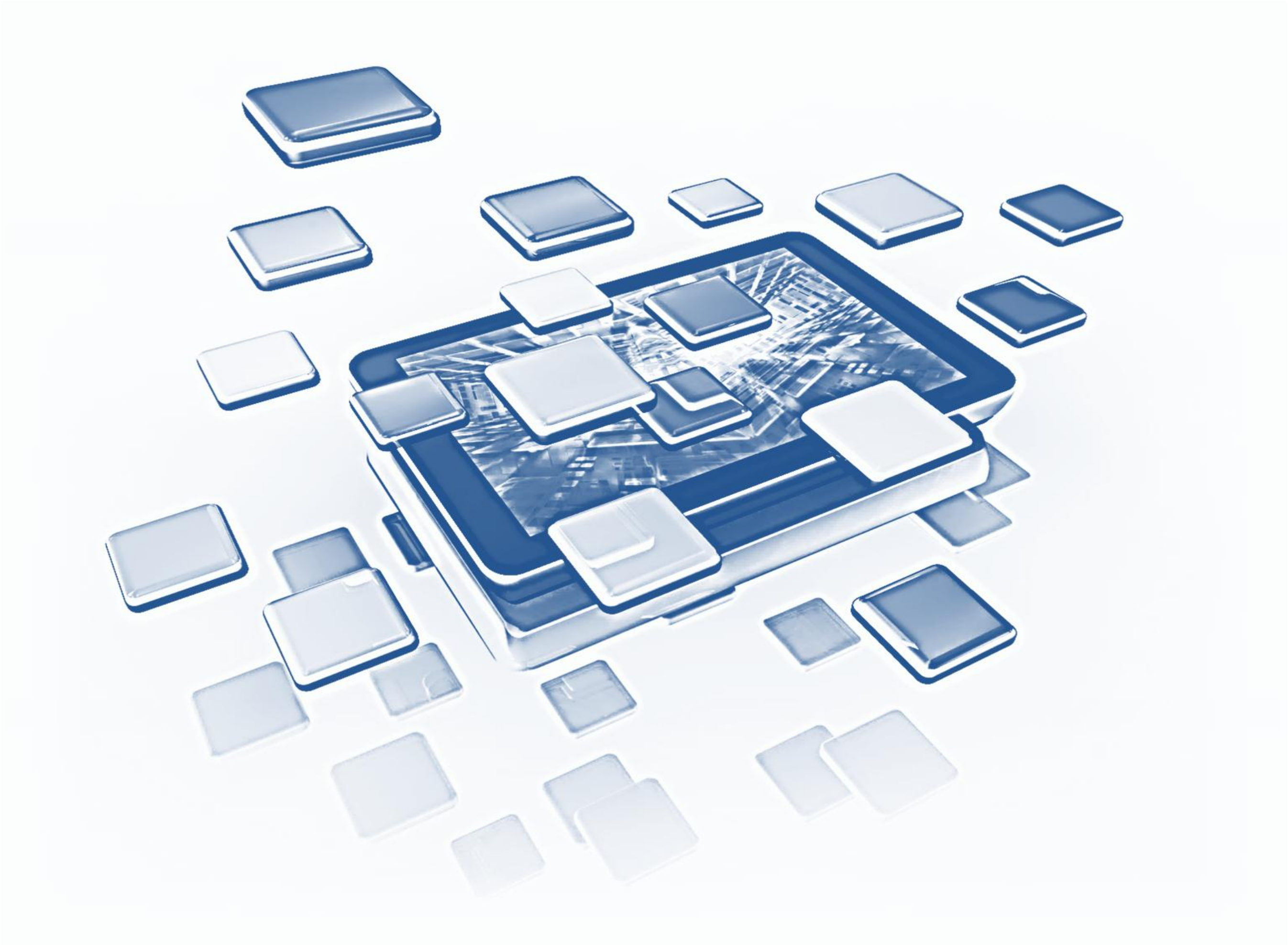
Appstore security

5 lines of defence against malware

September 2011

Appstore security

5 lines of defence against malware



enisa

European Network and Information Security Agency

About ENISA

The European Network and Information Security Agency (ENISA) is an EU agency created to advance the functioning of the internal market. ENISA is a centre of excellence for the European Member States and European institutions in network and information security, giving advice and recommendations and acting as a switchboard of information for good practices. Moreover, the agency facilitates contacts between the European institutions, the Member States and private business and industry actors.

Contact details

Authors:
Dr. Marnix Dekker, CISA (marnix.dekker@enisa.europa.eu) and
Dr. Giles Hogben (giles.hogben@enisa.europa.eu).

Press and inquiries:
Ulf Bergstrom (ulf.bergstrom@enisa.europa.eu).

Credits

The threat analysis was performed in collaboration with the DistriNet Research Group, K.U.Leuven, Belgium (as part of ENISA tender P/29/10/TCD), in particular: Prof. dr. Frank Piessens, Dr. Lieven Desmet, Dr. Pieter Philippaerts, and Philippe De Ryck.

We consulted with industry experts while drafting this paper and the malware defences in particular. We are grateful for their valuable input and comments.

- Peter Dickman, Nick Kralevich (Google)
- Nader Henein (Research in Motion)
- Kari Ti. Kostiainen, Niall Odonoghue, Timo J. Heikkinen, Mikko Saario (Nokia)
- Vinay Bansal (Cisco Systems)

Legal notice

Notice must be taken that this publication represents the views and interpretations of the authors and editors, unless stated otherwise. This publication should not be construed to be an action of ENISA or the ENISA bodies unless adopted pursuant to the ENISA Regulation (EC) No 460/2004 as lastly amended by Regulation (EU) No 580/2011. This publication does not necessarily represent state-of the-art and ENISA may update it from time to time.

Third-party sources are quoted as appropriate. ENISA is not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for educational and information purposes only. Neither ENISA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this publication.

Executive Summary

Smartphones will outnumber PC's by 2013 and they will be the most common device for accessing the internet (Gartner, 2010). A key feature of smartphones is the use of *appstores*: managed repositories of third party software. Apple's appstore and Google's Android market have hundreds of thousands of apps, and claim billions of app downloads. Where in the past mobile phone users could just change ringtone or wall paper, apps turn the smartphone into the digital equivalent of the Swiss army knife, offering anything from sonic mosquito repellent to point-of-sale credit card payments.

Apps have not escaped the attention of cyber attackers. For example, in 2010 diallerware was found in appstores for Windows Mobile phones and in 2011 malware was disguised as a popular app on the Android appstore infecting thousands of smartphones. Still, the number of malware attacks on smartphones pales in comparison with PCs. The large market share of PCs plays a role, but we believe that security design choices have been instrumental in preventing smartphone malware. To consolidate this head start, in this paper we analyse malware threats in *app ecosystems* and we identify five lines of defence that protect end-users[1] from malware and insecure apps:

- **App review:** Appstores should review apps before admitting them to the appstore. While app review cannot be perfect, it limits the possibilities for app developers to introduce malicious, or legitimate but insecure apps in appstores. Appstores can check apps with automatic (static and dynamic) analysis tools. Additionally human (manual) review can be used. While scalability is a problem with human review, this could be addressed by focussing on sensitive functionality and by using escalation procedures.
- **Reputation mechanism:** Reputation of apps and app developers can help users avoid malware. Appstores should show the reputation of apps and app developers. Second-order mechanisms can increase reputation quality. Appstores could take into account the reputation of the same app in other appstores. A point of concern is that most users rate apps for their functionality and not for their security, so there should be a separate channel for security and privacy issues (e.g. "this app works, but asks for excessive privileges at install").
- **App revocation (aka kill-switch):** Smartphone platforms should support remote removal of installed apps by appstores. Appstores should have an app revocation mechanism for malware and insecure apps. In special cases, e.g. when malware breaks out of the app sandbox, it may be necessary to use customized removal tools.
- **Device security:** Appstore defences rely on the security of the devices running the apps. The device should install and run apps in sandboxes, to reduce the impact of malware. In the

[1] *This paper is part of a number of ENISA activities around smartphone security. We have previously published a full overview of information security risks for smartphone users. We are also working with OWASP to draft best practices for app developers.*

sandbox, apps should get only a minimal set of privileges (the principle of least privilege). The sandbox should monitor the app inside it and allow the user to see the app's past activity. App revocation should uninstall the app and return the device to a pre-install state.

- **Jails (or walled gardens):** Smartphone (platform) vendors can restrict smartphones to apps from one or more designated appstores only and in this way prevent drive-by download attacks. This is commonly referred to as a jail or a walled garden. The smartphone should either be blocked from using untrusted appstores or, for expert users, present clear warnings about installing from untrusted sources. The approach to this issue is crucial – if users can easily install from untrusted appstores, then it is easy for attackers to bypass the defences of the good appstores (with drive-by download attacks e.g.). On the other hand, overly-restrictive jails encourage users to break the jail, possibly introducing higher risks than those originally mitigated by the jail. Jails should, for example, not be used to stifle legitimate competition.

We refer the reader to the body of this paper for details about the threat analysis (a combination of STRIDE threat modelling and attack trees) and the five lines of defence against malware.

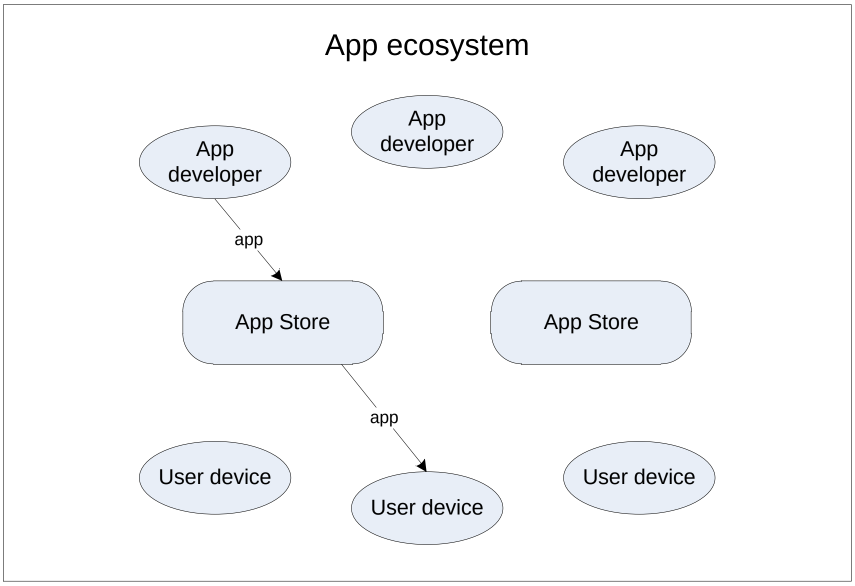
In conclusion, on the positive side, appstores can offer important opportunities to prevent, or reduce the impact, of malware and insecure apps. They can provide customers with 'vetted' software distribution channels, showing the reputation of apps, and operating a revocation mechanism for malware and insecure apps.

At the same time, cyber attackers are focussing more on smartphones. They will try to sell malicious apps directly or go after software vulnerabilities in popular apps. The stakes are high: Consumers, government and business professionals use smartphones to store and process large amounts of confidential and personal data. Different smartphone platforms and different app stores currently address malware and insecure apps differently, which for consumers can be confusing. Without overlooking the differences between the various smartphones models and appstores, we recommend an industry-wide approach to addressing malware and insecure apps. Security teams should exchange information about apps (analyses, reputation) and share security practices. Consumers should be presented with clear security information about app developers and the apps they sell. A solution could be a distributed reputation system for apps and app developers across the different appstores.

By analysing the threats and showing a baseline set of malware defences in this paper, we hope to contribute to a more common and standardized approach to app store security, across the (booming) smartphone industry.

1. Introduction

Besides the well-known Apple Appstore and Google Android Market there are many other appstores. For instance, Amazon opened an appstore for Android smartphones, Microsoft has one for Windows Mobile phones, Nokia for Symbian-based smartphones, CISCO has an appstore for its tablet, and some enterprises even created appstores for their employees. Besides appstores for smartphones there are also other appstores, for example for social media networks (Facebook apps), for business cloud services (Google apps), for browsers (Mozilla's addons), and so forth. In this paper we call a set of appstores, targeted at a specific platform, an *app ecosystem* (see figure below).



In an app ecosystem, *app developers* create apps, and sell or distribute them to users. An *app* is a piece of software that extends the functionality of the *user device* (a smartphone or a browser platform for example). The appstores receive apps from app developers and sell (or distribute) them to users, acting as a broker. Appstores generally show the reputation of each app, for example the number of downloads per app, the user reviews and the user votes.

In this paper we focus on the threat from malware or insecure apps in app ecosystems. We first build a dataflow model of an app ecosystem in Section 2. The app ecosystem model is used for a STRIDE threat analysis which is presented in Section 4. The scope of our threat analysis, assumptions about the attackers in scope, is explained in Section 3. In Section 5 we analyse the threats using attack trees and we identify five lines of defence.

2. Dataflow diagram of an app ecosystem

In this section we construct a model of an app ecosystem, which we will use in Section 4 as the basis for a STRIDE threat analysis.

2.1 Introduction to dataflow diagrams

We describe our model using a data-flow diagram[2]. The diagram has the following elements:

- **Interactors** (rectangles) generate input and consume output (to a process), e.g. humans.
- **Processes** (circles) perform some specific function. They take one or more inputs, and generate one or more outputs, e.g. computer system functionality.
- **Datastores** (two parallel lines) are used for storing data temporarily or permanently, e.g. a file system or a database.
- **Trust boundaries**[3] (red dashed lines) indicate the edges of control. For example, we draw a trust boundary between the smartphone and the appstore because the smartphone is under control of the user, and the appstore under control of the appstore owner.

2.2 Modelling an app ecosystem

The full diagram is shown below:

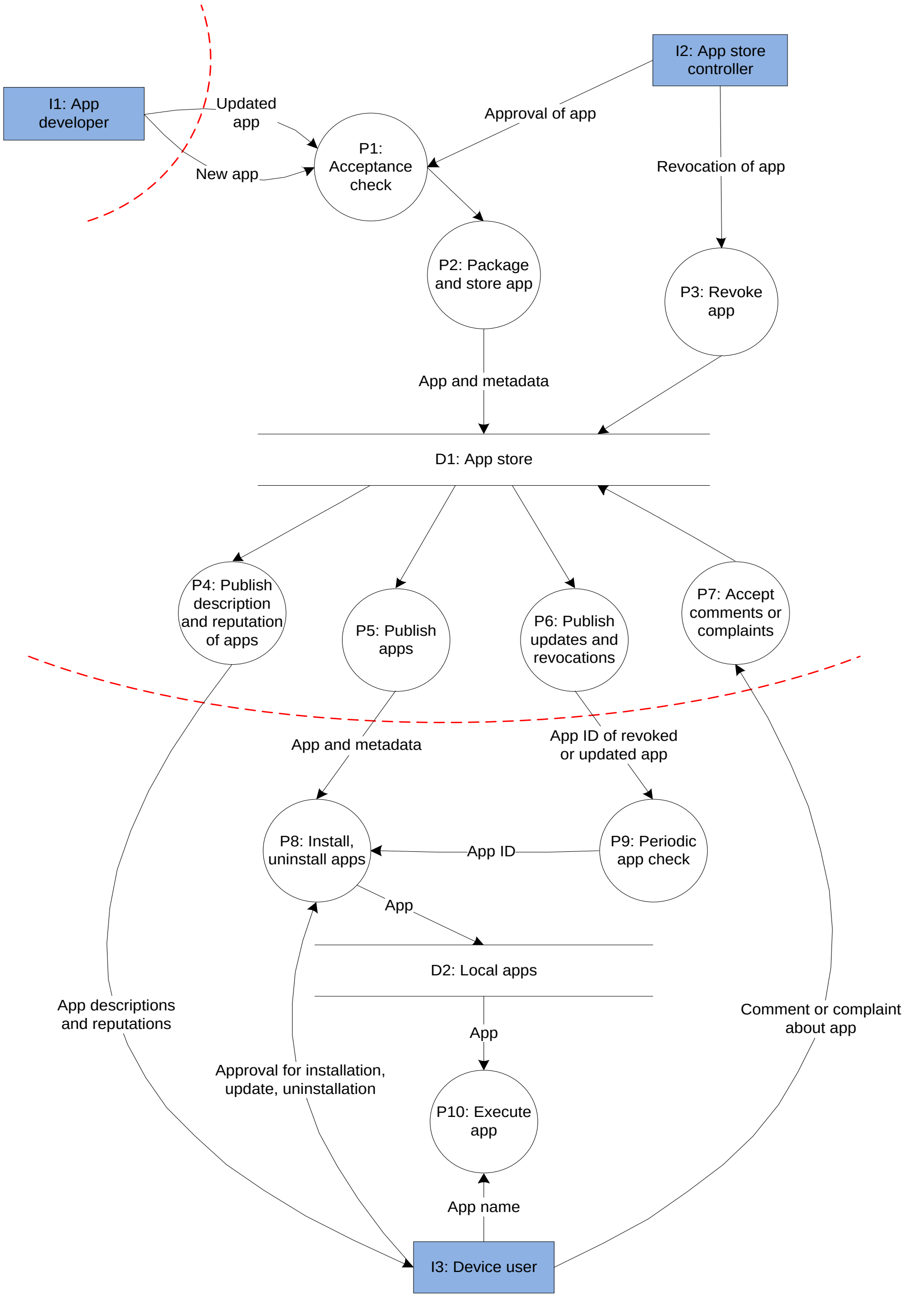
- The upper left corner shows the main use cases for the *app developer* (I1). The app developer can send a new app, or an update, to the *acceptance check* (P1), which checks whether the app is suitable for inclusion in the appstore.
- The upper right corner shows the main use cases for the *appstore controller* (I2). The appstore controller can approve apps for inclusion in the *appstore* (P1). After approval the app is packaged for inclusion in the appstore (P2). *Metadata* is added to the app such as a description and a list of permissions that the app needs on the user device (sometimes called a ‘manifest’). Additionally the appstore controller can *revoke* bad apps (P3), based on complaints for example. Revocation removes apps from the user devices (sometimes called a ‘kill switch’).
- The lower part of the diagram shows the main use cases for the *device user* (I3). The device user can browse descriptions and reputation of apps (P4). The installer (P8) requires as input the actual app published by the appstore (P5) and approval from the device user. On installation the apps are stored in the datastore with *installed apps* (D2). After installation the device user can execute an app (P10). The device user can also submit comments and complaints about apps, which are processed and stored in the appstore (P7).

[2] *One could use STRIDE with other modelling techniques, for example UML, but data flow diagrams are most common and support the use of tools such as the Microsoft SDL Threat Modeling Tool.*

[3] *Trust boundaries are not usually included in data flow diagrams, but they are used in STRIDE threat analyses; in a STRIDE analysis the focus is on the data flows crossing trust boundaries.*

- To ensure that the user devices receive timely updates and revocations, there is a periodic check (P9) for updated or revoked apps. Updates and revocations trigger the installer to install an updated app or uninstall a revoked app.

Note that our model is only a general, simplified description of how app ecosystems work, based on an analysis of the Android marketplace, the iPhone appstore, and the Mozilla Addons store.


I1: App developer
Updated app
New app
I2: App store controller
Approval of app
P1: Acceptance check
Revocation of app
P2: Package and store app
P3: Revoke app
App and metadata
D1: App store
P4: Publish description and reputation of apps
P5: Publish apps
P6: Publish updates and revocations
P7: Accept comments or complaints
App and metadata
App ID of revoked or updated app
P8: Install, uninstall apps
App ID
P9: Periodic app check
App
D2: Local apps
App descriptions and reputations
App
Comment or complaint about app
Approval for installation, update, uninstallation
P10: Execute app
App name
I3: Device user

3 Attacker model

Here we set the scope of the threat analysis and define which attacks and attackers are in scope.

In this paper we assume the target of cyber attackers to be users, consumers or professionals in public or private sector organizations, who download and install apps. We focus on attacks that introduce malware on the device via apps or appstores[4]. We don't distinguish between stand-alone malware and malware that relies on other apps. The attackers have two intermediate technical goals:

- to get malicious code on the user device, and (if that works)
- to keep malicious code on the user device

We ignore attacks directed at app developers or appstores which have no impact on the end-user (e.g. click-fraud, plagiarism, unfair competition). We also ignore social engineering attacks in which the user is tricked to configure insecurely an otherwise secure app.

Remark: There are 2 layers of software on a smartphone: the OS and the apps. So there are 4 categories of malware attacks:

1. *Sell or distribute a malicious app.*
2. *Exploit vulnerability in an existing app.*
3. *Sell or distribute a malicious OS*
4. *Exploit vulnerability in an existing OS*

In this paper we mostly focus on categories more on 1 and 2. The Gemini and DroidDream attacks, seen in 2011, are an example of the first category. The ZitMo malware is an example of the second category[5]*.*

Selling or distributing a malicious OS (category 3) may seem to be a relatively minor threat, but there are device users who install a custom, unofficial OS on their smartphone, for added functionality. There is a risk that malicious operating systems are distributed: users should review the reputation of such operating systems carefully. Exploits of OS vulnerabilities (category 4) have been the dominant vector for attacks on PC's in the past decade. For protecting against OS exploits we refer the reader to a literature survey by D. Gollmann.

[4] *It should be stressed that information security risks do not only come from malware alone. Even authentic, vetted apps can lead to data leaks, for example voice to text conversion software that uploads audio for processing. For corporate users such apps may expose sensitive customer data.*

[5] *Exploits of third-party software have become the dominant problem for PCs as well: For example, according to Symantec more than half of the web-based attacks in 2010 exploit flaws in Adobe software (either the plugin or supporting libraries).*

4. STRIDE Threat analysis

In this section we show the result of the STRIDE threat analysis on the app ecosystem model.

4.1 Introduction to STRIDE

STRIDE categorizes threats into six types[6]:

- **S**poofing threats: a process or an interactor pretends to be someone/thing else.
- **T**ampering threats: a process, a data flow, or a datastore is changed.
- **R**epudiation threats: evidence of an action by a process or an interactor disappears.
- **I**nformation disclosure threats: a process, a dataflow, or a datastore reveals sensitive data.
- **D**enial of service threats: a data flow, a datastore, or a process is overloaded, rendering normal use impossible.
- **E**levation of privilege threats: a process is used to perform unauthorized actions.

The 6 threat types apply to different elements in a dataflow model, as follows.

	Spoofing	**T**ampering	**R**epudiation	**I**nformation disclosure	**D**enial of service	**E**levation of privilege
Data flows		x		x	x	
Datastores		x		x	x	
Processes	x	x	x	x	x	x
Interactors	x		x			

The focus of the STRIDE analysis is on threats which cross trust-boundaries in the dataflow model.

4.2 STRIDE threat analysis on the trust boundaries

Our model has 3 interactors, 10 processes, 2 datastores, and 20 dataflows, so the full STRIDE sweep gives 132 threats. We enumerate all the (61) threats on the trust boundaries first. Threats considered marginal to the scope of this paper are included for the sake of completeness, but typeset in grey.

Element	Type	Threat description
App developer (I1)	**S** **R**	T1 App developer is impersonated (spoofed) by an attacker and (in his name) submits a malicious app. T2 Attacker submits a malicious app, and denies this later.
Dataflows beteen app developer (I1) and the appstore (P1).	**T** **I**	T3 Attacker tampers with an app or an update, adding malicious code to it. . T4 Attacker learns sensitive information, for example authentication credentials of the app developer. T5 Attacker denies app developers to submit apps or updates by overloading

[6] *STRIDE includes only technical threats, resulting from design flaws or bugs, and not non-technical threats such as bribery.*

	D		the acceptance check with many apps.
Acceptance check (P1)	**S**	T6	Attacker spoofs the acceptance check, and gets the developer to reveal authentication credentials.
	T	T7	Attacker gets a malicious app past the acceptance check.
	R	T8	Appstore denies having received an app or an update for acceptance.
	I	T9	Attacker learns sensitive information about the appstore or other app developers from the acceptance check.
	D	T10	Attacker denies app developers to submit apps or updates by overloading the acceptance check with many apps.
	E	T11	Attacker approves a malicious app or submits an app on behalf of someone else.
Interactor I3 (Device user)	**S**	T12	Attacker impersonates a device user and posts false feedback for an app or skews download numbers for apps.
	R	T13	Attacker repudiates having given false feedback.
Publish description and reputation of apps (P4)	**S**	T14	Attacker spoofs the appstore interface, so users see false descriptions and reputations of apps.
	T	T15	Attacker tampers with the appstore interface, changing the descriptions and reputations of apps.
	R	T16	Attacker browses descriptions and reputations but denies this later.
	I	T17	Attacker learns sensitive information, for example which users want to install which apps.
	D	T18	Attacker prevents users from browsing app descriptions by overloading the appstore.
	E	T19	Attacker carries out unauthorized actions, such as changing the descriptions and reputations of apps.
Dataflow between P4 and I3: App descriptions and reputations	**T**	T20	Attacker changes the descriptions and reputations of apps.
	I	T21	Attacker learns which users have installed which apps.
	D	T22	Attacker prevents users from browsing app descriptions by overloading.
Publish apps (P5)	**S**	T23	Attacker spoofs the appstore, so the device installs malicious apps.
	T	T24	Attacker tampers with the appstore, so the device installs malicious apps.
	R	T25	Attacker downloads an app for installation and denies this later.
	I	T26	Attacker learns sensitive information from the appstore, such as which apps are being installed on which devices.
	D	T27	Attacker denies access to the appstore, for example to prevent devices from downloading apps or updates for installation.
	E	T28	Attacker carries out unauthorized actions, such as inserting additional (malicious) apps in the store.
Dataflow between P5 and P8: App and metadata	**T**	T29	Attacker tampers with the app, so the device installs a malicious app.
	I	T30	Attacker learns which apps are installed on which devices.
	D	T31	Attacker prevents devices from downloading apps or updates for installation.
Publish updates and revocations (P6)	**S**	T32	Attacker spoofs the appstore, so the device does not get all the revocations or updates.
	T	T33	Attacker tampers with the appstore interface, so the device does not get all the revocations or updates.
	R	T34	Attacker receives an update or revocation notice and denies this later.
	I	T35	Attacker learns sensitive information from the appstore interface, such as which apps are installed on which devices.

	D	T36 Attacker denies access to the appstore interface, for example to prevent users from receiving updates and revocations.
	E	T37 Attacker carries out unauthorized actions, such as changing or removing updates or revocations.
Dataflow between P6 and P9: App ID's of revoked or updated apps.	T	T38 Attacker tampers with the updates and revocations from the device, so that the device does not receive all the updates and revocations.
	I	T39 Attacker learns which apps are installed on which devices.
	D	T40 Attacker prevents devices from receiving updates or revocations for installation.
Accept comments or complaints (P7)	S	T41 Attacker spoofs the appstore, so the user submits comments and complaints in the wrong place.
	T	T42 Attacker tampers with the appstore interface, changing or removing complaints.
	R	T43 Attacker submits positive feedback information and denies this later.
	I	T44 Attacker learns sensitive information, for example which apps are installed on which devices.
	D	T45 Attacker prevents device users from submitting comments and complaints by overloading the appstore with comments.
	E	T46 Attacker carries out unauthorized actions, such as removing comments or complaints about an app.
Dataflow between P7 and I3: Comment or complaint about app	T	T47 Attacker changes comments and complaints from the device user.
	I	T48 Attacker learns sensitive information about the device users, for example which apps are installed by which users, or personal details about device users.
	D	T49 Attacker prevents device users from submitting comments and complaints by overloading the appstore with comments.
Install, uninstall apps (P8)	S	T50 Attacker spoofs the device, for example to skew the statistics on which users have installed which apps or updates.
	T	T51 Attacker tampers with the installer, installing malicious apps.
	R	T52 Attacker installs an app, denying this later on, for example to skew the statistics on which users have installed which apps or updates.
	I	T53 Attacker learns sensitive information about the user, for example which device the user is carrying.
	D	T54 Attacker overloads the installer, preventing the user from installing updates, or uninstalling revoked apps.
	E	T55 Attacker carries out unauthorized actions, such as installing malicious apps and rootkits.
Periodic app check (P9)	S	T56 Attacker spoofs a user device, for example to skew the statistics on which users have received update or revocation notices.
	T	T57 Attacker prevents a user device from receiving updates or revocations.
	R	T58 Attacker receives an update or revocation notice, denying this later on, for example to skew the statistics on how many users received updates or revocations.
	I	T59 Attacker learns sensitive information from the appstore user interface (for example which apps are installed on which user devices).
	D	T60 Attacker overloads the process, preventing the user device from receiving updates and revocation information.
	E	T61 Attacker carries out unauthorized actions, removing revocations.

4.3 STRIDE analysis inside the trust boundaries

A full STRIDE threat analysis would also cover the elements and data flows *inside* the trust boundaries. For example, the full STRIDE analysis also yields a spoofing attack on the Appstore controller (I2) - an attacker impersonating the Appstore controller. For the sake of brevity, we do not elaborate all these threats and just discuss the internal threats (within the trust boundary) to the process Execute app (P10) because often malicious behaviour only emerges at runtime.

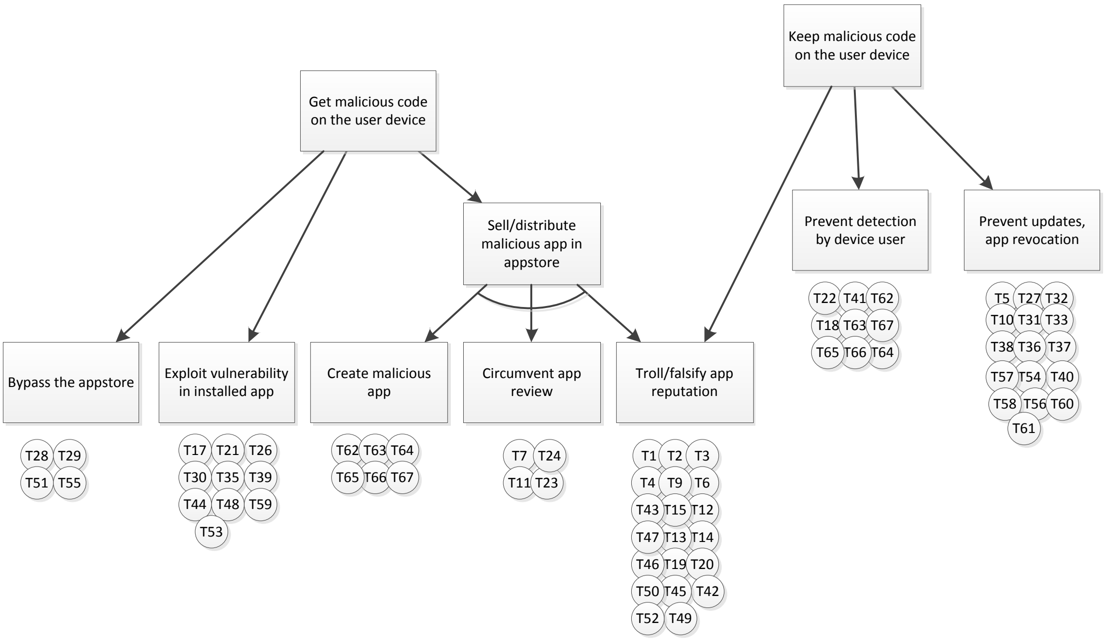
Element	Type	Threat description
Execute app (P10)	**S**	T62 Attacker spoofs the execution process, so the user thinks (wrongly) that an app is executing.
	T	T63 Attacker tampers with the execution process, to perform malicious functions.
	R	T64 Attacker can run an app without leaving evidence (such as logs).
	I	T65 Attacker learns sensitive information from the execution process, for example about the device user, or sensitive information on the device.
	D	T66 Attacker overloads the execution process, to prevent the user from executing other apps.
	E	T67 Attacker carries out unauthorized actions, such as tampering with other apps, reading data stored by other apps, or reading sensitive information.

5. Defending against the threats

The STRIDE analysis yields many different threats. To address them systematically we use attack trees - a technique introduced by Bruce Schneier to analyse all the different ways to attack a system[7].

5.1 Attack tree

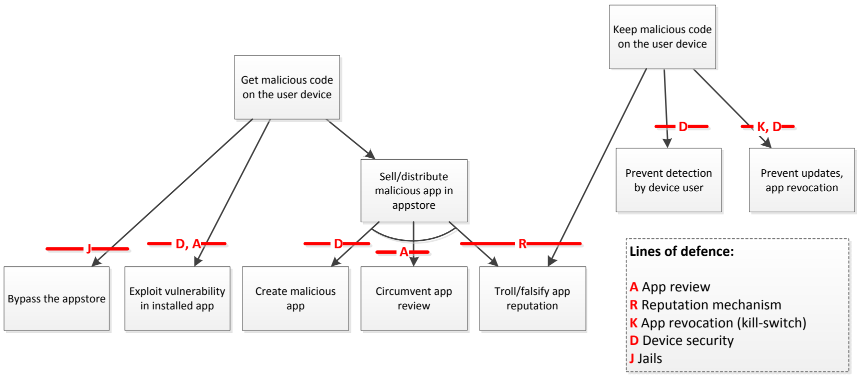
An attack tree for attacking the device user with malware is shown below. The top nodes are the high-level technical attacker goals: to get malicious code on the user device, and to keep malicious code on the user device. In this tree we take into account both malicious apps and exploits of vulnerable apps. We do not refine further to practical goals, such as stealing money or sensitive data. An arc denotes conjunction (meaning that both sub goals are required to execute the attack). The threats from the STRIDE analysis, showing how they can be used in attacks, are on the bottom of the diagram.



[7] *Attack trees are similar to fault trees and dependency trees which are used in industrial engineering to analyse how a system or a structure deals with incidents or disaster.*

5.2 Lines of defence

In this section we identify 5 lines of defence which can be deployed to prevent malware attacks on end-users (see below). The defence mechanisms are shown in the attack tree below.


Keep malicious code on the user device
Get malicious code on the user device
D
K, D
Prevent detection by device user
Prevent updates, app revocation
Sell/distribute malicious app in appstore
J
D, A
D
A
R
Bypass the appstore
Exploit vulnerability in installed app
Create malicious app
Circumvent app review
Troll/falsify app reputation
Lines of defence:
A App review
R Reputation mechanism
K App revocation (kill-switch)
D Device security
J Jails

App review (A)

Appstores can check apps for security issues before they are distributed to end-users. As mentioned earlier, app reviews cannot give a 100% security guarantee, but they make it harder for attackers to introduce malware in app ecosystems.

- **Automated software analysis tools:** Static analysis tools can be used to inspect the source or binary code of an app to ensure that it does not use unauthorized functionality and that it adheres to (some of) the developer guidelines. An example would be the tool in Apple's submission process that checks for forbidden API calls. Alternatively, such tools can scan for viruses or known malicious code fragments. In addition to static analysis tools, dynamic analysis tools can be used, that will actually run the app against a number of test cases so it can be checked and monitored for unwanted or unauthorized behaviour. An example is Microsoft's 'Hopper' tool that runs a submitted application for a number of hours and monitors memory usage, performance, and stability. Automated tools could be set up to trigger escalation procedures to manage resource-intensive analysis such as manual analysis (see next bullet).
- **Manual analysis**: While research in software analysis is slowly pushing the boundaries of what can be analysed automatically, there are a number of aspects that can only be checked by a human, for example if an app is trying to spoof another app to fool the user. Scalability of manual analysis is a point of concern for appstores. Escalation procedures and a focus on critical functionality can be used to make manual analysis more efficient.
- **Sharing analysis results:** When possible, results of security analysis should be shared with other appstores and security researchers. Appstores could also leverage the expertise of 3rd party researchers and security companies by allowing them to bulk-download and analyse apps.

- **Authentication of app developers:** App developers should be securely authenticated so that rogue app developers cannot piggy-back on the (good) reputation of other app developers. In app ecosystems often single individuals create apps rather than large software vendors[8]. Apps are not only developed in well-protected development environments. This means that authentication of app developers becomes more important. Phishing attacks or XSS attacks to get the credentials of app developers are a significant risk.
- **Risk profiling of app developers:** The appstore should monitor and create risk profiles of app developers. New app developers, or app developers who submit unsafe or malicious apps, should be treated with special care. Anomalous behaviour from well-known app developers could be a sign of an attack (phishing for example).
- **Continuous process**: App review should be a continuous process, and appstores should analyse apps even after they have been admitted to the appstore, for example by using more resource-intensive analysis (manual analysis e.g.). This type of ex-post review could be triggered by the popularity of an app (download numbers) or the reputation mechanism (see below).
- **Priority for updates:** Appstores should consider priority vetting for updates to existing (and popular) apps to allow app developers to patch vulnerabilities quickly.

Reputation mechanism (R)

A reputation mechanism for apps and app developers is important for device users to be able to avoid insecure apps. A reputation mechanism does not give a 100% guarantee, because attackers could invest in building up a good reputation, but it does make it harder for attackers to distribute malicious apps.

- **App track record:** To allow users to make a good choice, the reputation of an app should show the history and track record of app developers and apps, download statistics of apps, user votes, and detailed comments and complaints from users.
- **Separate security and privacy reputation**: An important issue with current app reputation systems is that users often only rate apps for their functionality. Security and privacy information about apps should be treated separately in app store reputation mechanisms.
- **Sybil attack resistance:** Mechanisms should be in place to avoid attackers from creating multiple pseudonymous identities and thereby gaining excessive influence to be able to skew the reputation of apps (aka Sybil attack). Second-order mechanisms can be used to foil Sybil attacks (see next bullet).

[8] *Consumerisation of software development also means that software developers can more easily change names to escape a bad reputation, and that security processes may be absent, for example patcbing processes.*

- **Second-order reputation:** Second-order reputation can prevent attacks on the reputation scheme by giving more weight to votes from users who have a good reputation amongst other users. This can increase scoring quality and relevance and prevent trolling and Sybil attacks.
- **Anonymous feedback:** Comments and complaints from device users should be anonymised and sensitive information about device users should be removed from public feedback to encourage honest feedback and to avoid targeted attacks that rely on which apps a device user installed.
- **Exchanging reputation information:** When possible, reputation information about the same app in other (trusted) appstores should be taken into account. To allow for such an exchange apps should have unique identifiers and signatures to allow referencing across different appstores.
- **Permission feedback:** reputation systems should allow users to give separate feedback on specific security-relevant features such as excessive permission requests (e.g. snake game asks for access to GPS and telephone calls).

App revocation or kill-switch (K)

Many smartphone platforms implement an app revocation mechanism, or kill-switch, which uninstalls apps installed on user devices. App revocation can be triggered either by the app store or by the smartphone (platform) vendor. Important aspects of app revocation mechanisms are the following.

- **User communication and consent:** Removing apps from user devices is a controversial subject. Users should be informed about the reasons for the removal and, if this does not jeopardize other users, given a chance to opt-out. There are also settings where app revocation goes against security policy. For example in a military setting, apps may be mission-critical and the app revocation mechanism may need to be turned off.
- **Spawning:** It is important to prevent a malicious app from spawning – installing difficult to remove code – across the user device during installation or runtime. I.e. it should be possible, by uninstalling the app to reverse all changes caused by the app. On PC's for example attackers often spawn malicious code in places where it is difficult to remove (rootkits). In the DroidDream attack, the Android kill-switch could not be used because the malicious code was outside the sandbox. In such a case, a custom removal tool may be required.
- **Update frequency:** Smartphone (platform) vendors should encourage users to update frequently. Security updates should be kept small in size whenever possible, to prevent that users with limited-bandwidth or metered internet access skip large updates.
- **Detection:** The reputation mechanism (see above) plays an important role because it often provides the basis for a revocation. Comments and complaints should be monitored

continuously. Apps may exhibit malicious behaviour at a specific moment (say Easter eggs), at random intervals, or randomly across users as seen in malvertisement attacks[9].

- **False positives:** App revocation could potentially be used to uninstall good apps from user devices. The app revocation mechanism should only be accessible to security teams who can base their decisions on app reviews, user reviews, complaints.

Device security (D)

Appstore defences rely for a large part on a secure implementation on the device. App sandboxes are vital for the security of the app ecosystem because they limit what apps can do on the device[10]. Important device-side security features are the following:

- **Code signing:** The user device should only accept apps that are signed by the right app store.
- **Sandboxes:** The user device should have sandboxes (or containers) for apps, so that apps are installed and run in isolation, to reduce the impact of malware.
- **Minimal set of privileges:** In the sandbox, apps should have only a minimal set of privileges by default (applying the principle of least privilege) and if an app needs additional privileges (for example to access GPS data) this should be handled with care; either the user should be explicitly asked for consent, or the app store should grant permission during app review. An important issue here is that users are often not in a position to make good security decisions, especially when asked frequently.
- **Monitoring by the smartphone user:** An important feature of sandboxing is monitoring of the app inside. The device should monitor apps after they have been installed. Device users should have the possibility of seeing reports of the activity of an app (network usage, resource usage, and so on) to detect resource-intensive code.
- **Clean slating:** The device should – when triggered by the appstore or the user – uninstall the app and return the device to a pre-install state.

Jails or walled gardens (J)

The device vendor (or the platform distributor) can restrict installation of apps to one or more trusted appstores or warning when the user is installing apps from untrusted appstores. This is commonly referred to as a jail or a walled garden. The approach to app installation policies (and jails in particular) is crucial, because most of the above-mentioned defences are useless if users are quick to skip warnings or jailbreak their devices. If it is easy to skip warnings then users are vulnerable to drive-by download attacks – a common source of infection on PC's. If jails are too restrictive users will try to jailbreak their devices which could expose them to even higher risks, for example when jailbreaking

[9] *In some malvertisement attacks, attackers have uploaded genuine-looking - but malicious – banners (to exploited advertisement servers) that infect only a few users at random moments, to prevent detection.*

[10] *Sandboxing and capabilities was rated as the top information security opportunity in our smartphone risk assessment.*

removes other defences in the process. There are several models in use, each with different security considerations:

- **Closed app ecosystems:** In a closed app ecosystem, there are one or more designated appstores that can sell or distribute apps. The devices are configured in such a way that users can only install apps from one or more designated appstores. The appstores in the closed system form a federation and can agree on common security practices.
- **Enterprise app stores:** For enterprise users, smartphones could be configured to allow only apps from a dedicated enterprise app store, allowing the enterprise tight control on app installation.
- **Open app ecosystems:** In an open app ecosystem anyone can open an appstore and start selling or distributing apps to device users. While this gives the device user maximum freedom of choice, just like on a traditional PC, it also increases the risk of drive-by download attacks.
- **Federated appstores:** Appstores could form federations by agreeing, inter alia, to a minimum level of security. Such a federation of app ecosystems could give end-users more choice, while at the same time preserving the security benefits of a closed app ecosystem.
- **App reputation across appstores:** Another possibility is to keep a single central list of 'well-reputed' apps as the app whitelist of the app ecosystem. Such a reputation system would provide a central repository of security information about apps, independently of where the apps are sold. In this way, the central list is only responsible for assuring the authenticity of the list entries and their reputation in terms of security and privacy.

This concludes the body of our paper. We refer the reader to the executive summary for our conclusions.

(This page is intentionally left blank)

P.O. Box 1309, 71001 Heraklion, Greece
www.enisa.europa.eu