Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-1030 Wien, Seidlgasse 22 / 9
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

A-8010 Graz, Inffeldgasse 16a
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

STATIC ANALYSIS OF WINDOWS PHONE APPLICATIONS

Version 1.0, 27/11/2015
Alexander Marsalek – alexander.marsalek@a-sit.at

Executive Summary: The objective of the project was to analyse a number of Windows Phone Apps on common security issues. It started with the manual analysis of selected applications. Soon it became evident that many of the analysis steps can be easily automated to save time. Another observation was that several applications suffer from similar security issues. The found issues correspond to four groups:

- Broken Cryptography,
- Insufficient Transport Layer Security,
- Exposed API keys and
- Other exposed client secrets.

As most of these issues can be found by an automated analysis, an automated framework for downloading and analysing Windows Phone applications was developed. Using this framework, roughly 40.000 distinct Windows Phone 8.1 application files were downloaded and analysed. Older applications for e.g. Windows Phone 7 are published encrypted, which makes them resistant to static analysis. The extracting and decompiling process of the ~40.000 applications resulted in over 2 million unique source-code files that were statically examined. 8.5% of all analysed applications contain at least one security relevant issue. The following list shows the number of affected applications per issue group:

- 13 applications use broken cryptography,
- 83 applications ignore HTTPS related certificate validation errors,
- 3.362 applications expose their Google API key,
- 23 applications expose their Amazon Web Service key, and
- 230 applications expose other API keys or client secrets.

The impact of these issues cannot be determined automatically, but it ranges from endangered user's privacy in case of broken cryptography or insufficient transport layer security, to monetary losses for the developer, when an attacker abuses the developer's billing enabled API key.

Contents

Version	Date	Changes made	Modified by
1.0	27.11.2015	Initial document	Alexander Marsalek

1. Introduction

A previous project analysed the possibilities of the Windows Phone platform regarding application analysis [1]. This project applies its results to a wider range of applications. The previous project identified several applications sources, from which a Windows Phone application can be retrieved. This project will use the official Windows Store as source, as it is the most feasible source in the scope of this project.
In the beginning, selected applications were manually downloaded and analysed to identify common problems. The obtained findings were then used to build an automated analysis framework, which allows to search for simple problems in an automated way. This report will not give details about distinct applications and their security problems, it will only give numbers regarding concerned applications. The following chapter describes the manual analysis and the identified security problems. Chapter 3 describes the automated analysis framework and Chapter 4 presents the results.

2. Manual Analysis

Windows Store applications are either published in “XAP”, “APPX” or “APPXBUNDLE” format. The “XAP” format was used for Windows Phone 7 and Windows Phone 8 applications. Microsoft encrypts “XAP” applications, which makes them resistant to static analysis. In contrast “APPX” and “APPXBUNDLE” applications are not encrypted. They can easily be extracted with a standard ZIP extractor. For this reason, this project will focus on “APPX” and “APPXBUNDLE” applications.
The first steps were to manually download selected applications, extract them and finally decompile the executable files. During the analysis of the source-code files, it soon became evident that several applications suffer from the same security issues.

2.1. Identified Security Issues

The found security issues can be categorized into four groups. Not all groups have the same impact on the security of the application.

2.1.1. API Keys

Several applications use third-party online services. Usually, an API key is needed to access these APIs. This API key allows the service provider to identify and to charge the user. Many service providers offer a limited number of requests per time for free. If the free usage limits are exceeded, the customer can enable billing. When classical web applications use such a third-party online service, the API key is stored only on the server, where it is inaccessible to a user. Nowadays, more and more classical web applications are offered as smartphone applications to end-users. This shift to end-user devices brings new security challenges, as the API key is typically needed by the mobile application and, therefore, stored locally, where it can be extracted by an attacker.

2.1.2. Poor Authorization and Authentication

During the manual analysis, it was striking that many applications use hard-coded credentials. Most of the found credentials were used to authenticate at (remote) APIs. Furthermore, several applications contain hard-coded client secrets for their OpenID Connect or OAuth clients. Some applications also use hard-coded credentials to authenticate at particular websites using the HTTP basic access authentication mechanism.

2.1.3. Insufficient Transport Layer Security

The Windows phone platform allows developers to ignore certain SSL server certificate errors. This feature was added for scenarios where e.g. a self-signed certificate is used that cannot be deployed on all clients as trusted root certificate. Table 1 lists the possible certificate chain validation results.

Chain Validation Result	Description
Success	The certificate chain was successfully verified.
Untrusted	At least one certificate in the chain is not trusted.
Revoked	At least one certificate in the chain was been revoked.

Expired	At least one certificate in the chain is expired.
Incomplete Chain	One or more certificates are missing in the certificate chain.
Invalid Signature	The signature of at least one certificate in the chain could not be verified.
Wrong Usage	At least one of the certificates in the chain is used for a purpose that is not specified by its CA.
Invalid Name	At least one of the certificates in the chain has a name that is not valid.
Invalid Certificate Authority Policy	At least one of the certificates in the chain has a policy that is not valid.
Basic Constraints Error	At least for one of the certificates in the chain the basic constraint extension has not been observed.
Unknown Critical Extension	At least one of the certificates in the chain contains an as critical marked extension that is unknown.
Revocation Information Missing	No DLL was found to verify revocation
Revocation Failure	The revocation server for at least one certificate could not be reached.
Other Errors	During the validation process occurred an unexpected error.

Table 1: Chain Validation Results (Source: [2])

If a developer wants to ignore one or more of these chain validation results, she could create an *HttpBaseProtocolFilter* object and add the chain validation results to the object's *IgnorableServerCertificateErrors* list. This feature was used by several of the manually analysed applications. Ignoring certificate errors can render the whole transport security mechanisms useless.

2.1.4. Broken Cryptography

The manual analysis revealed that some applications have serious security issues in their cryptography-related code. These issues range from hard-coded values (SALT, IV, passwords used for key derivation and keys), to too few iterations used for key derivations functions, to self-developed encryption algorithms. A hard-coded key or a key derivation from hard-coded data renders the complete encryption process pointless. Furthermore using only a few iterations for a key derivation weakens the security significantly.

2.2. Limitations

During the manual analysis, several observations were made. All the steps, from downloading an application, to extracting it and finally decompiling it are easy, yet they cost a lot of time. Another observation was that many apps share some classes, e.g. by using the same library. In the manual analysis, these classes were partly analysed twice. These limitations can be solved using an automated process. Another point is that an automated analysis can analyse a huge amount of applications. It can e.g. be used as filter step for suspicious applications that should be analysed manually.

3. Automated Analysis

The automated analysis consists of two parts that can be viewed separately. The first part comprises the app acquisition, the extraction and the decompilation process. The second part analyses the obtained files and reports potential security issues. The impacts of these issues are beyond the automated analysis; they need to be assessed manually.

3.1. Application Download & Preparation

The application download tool simulates a Windows Phone 8.1 device. Like the official store application, it allows to search the market by category, for new or improved applications, for

popular applications, by application name or ID. The sequence is shown in Figure 1. The download process starts with a search request to the Microsoft Store, e.g. search for all new and improved applications from the category “Business” that are free. The Microsoft Store returns a list of IDs that match the search query. It seems the maximal number of results per search query is limited to 1.000 IDs/applications. Using these IDs, it is possible to request the corresponding application metadata files. The metadata files include the application package format, the application download URL and other information as the application description. The download tool analyses this information and downloads the application if it has not been downloaded before and if it is not encrypted. After downloading, the application is extracted.

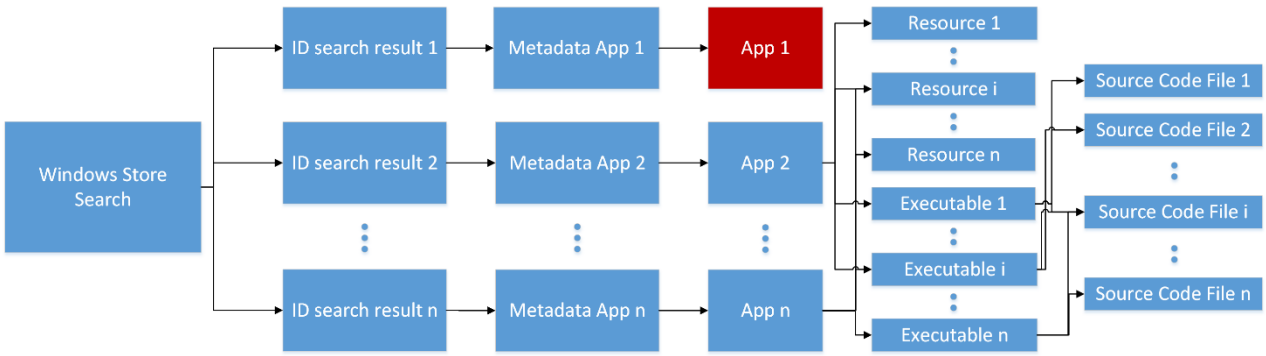

Figure 1: High-level overview of the application download, extraction and decompilation process.

The contents of the application package are stored on the filesystem, with their SHA-1 fingerprint as filename. This ensures that there are no duplicate files, which saves space and ensures that the same files are not analysed multiple times. The original names and the relationship between the files and applications are mapped in a database. After the extraction step, executable files (*.exe and *.dll) are decompiled. The gained source code files are also renamed to their SHA-1 fingerprint and their original names are stored in the database. All information is linked in both directions. This means, for example, it is possible to get all applications that use a certain source code file, which allows to identify quickly all applications that are affected by an identified security issue. On the other hand, it allows to get all resources and source code files that belong to a particular application, which comes handy for a subsequent manual analysis. The database model is shown in Figure 2.

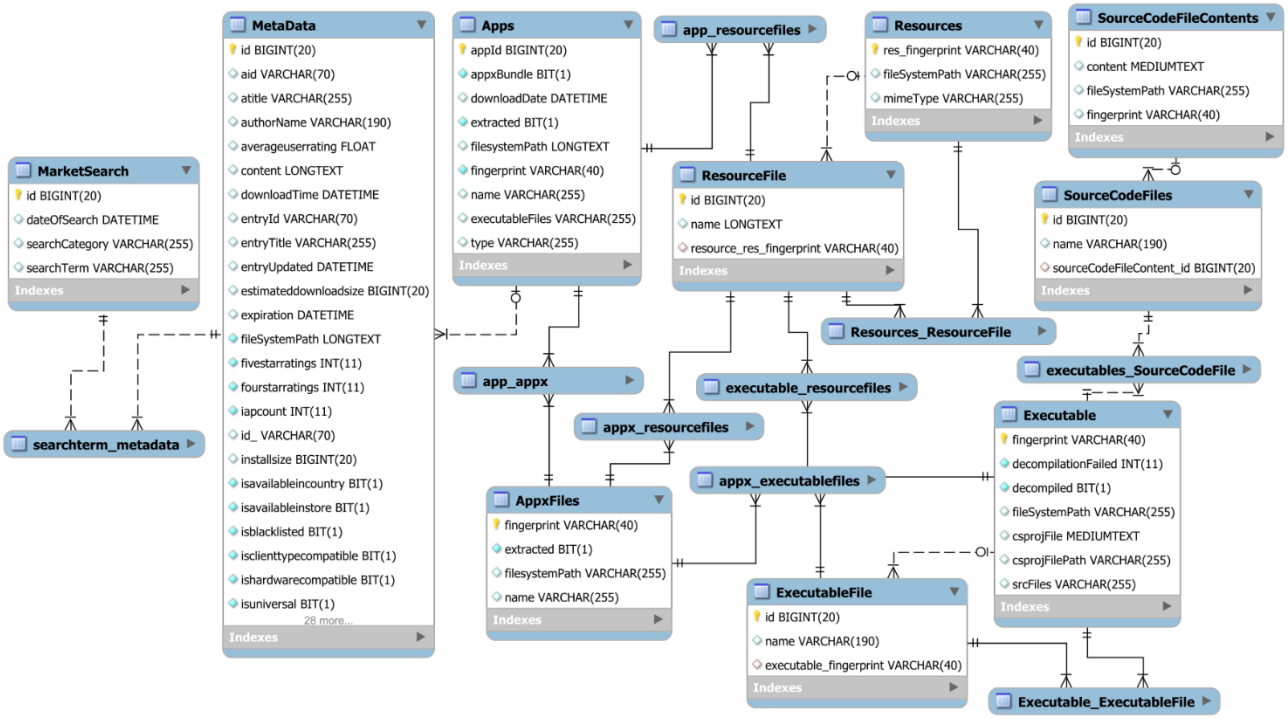

Figure 2: Database Model

3.2. *Application Analysis*

The database contains all information needed for automated analyses, but it has some drawbacks when it comes to full-text search or regular expression[1] based searches. In some cases, these queries took several hours to complete. Therefore, the source code and its relations were exported to an Apache Solr instance. Apache Solr is an open-source search server. Using this server, it was possible to get instant search results for the above-mentioned queries. The Solr interface supports manual and automated queries. Simple queries are sufficient to find certain security issues. As an example, Google API keys can easily be found, by searching for strings, which start with “AIZa” followed by 35 characters. Similar, Amazon AWS keys can be found by searching for strings of 20 characters length, which start with “AK” and contain only uppercase alphanumerical characters. Using this simple approach, false positives are possible as an application could contain a string that is not used as API key, but matches these requirements. But this circumstance is very unlikely, and a spot check of the results did not reveal any false positives.

4. Results

The database contains meta-information of approximated 300.000 applications. About 85% of these applications are encrypted (XAP applications) and, therefore, not analysable using static analysis techniques. The remaining 15% of applications were downloaded, decompiled and analysed.

These applications consist of about 100.000 distinct executables of which 86% could be decompiled successfully. The decompilation process resulted in over two million unique source-code files.

During the analysis 3.362 applications were found that contain a Google API key, that’s 7.7% of all analysed applications. 23 applications contain an Amazon Web Service key. 230 applications contain API keys or other client secrets for other online services. 83 applications contain code that can be used to ignore certificate errors. The breakdown to specific certificate errors is shown in Figure 3. 13 applications were found that use broken cryptography. Figure 4 shows the ratio of inconspicuous applications (91.5%) were no security relevant issue was found to application with at least one security issue (8.5%).

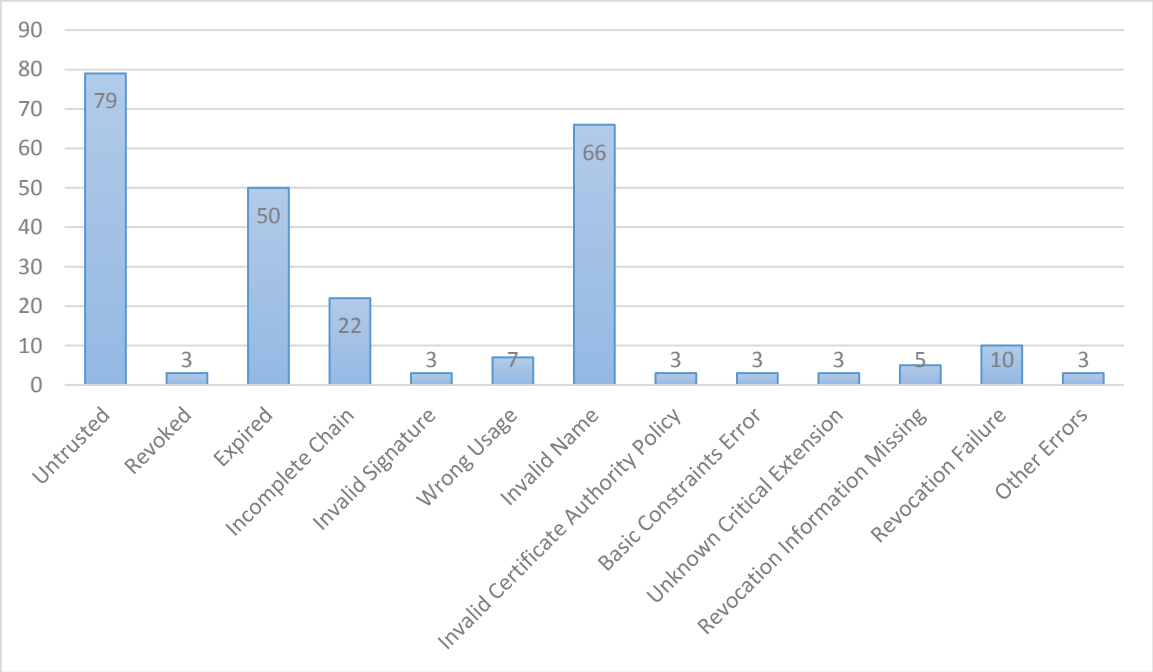

Figure 3: Number of applications ignoring a certain certificate error

[1] https://en.wikipedia.org/wiki/Regular_expression

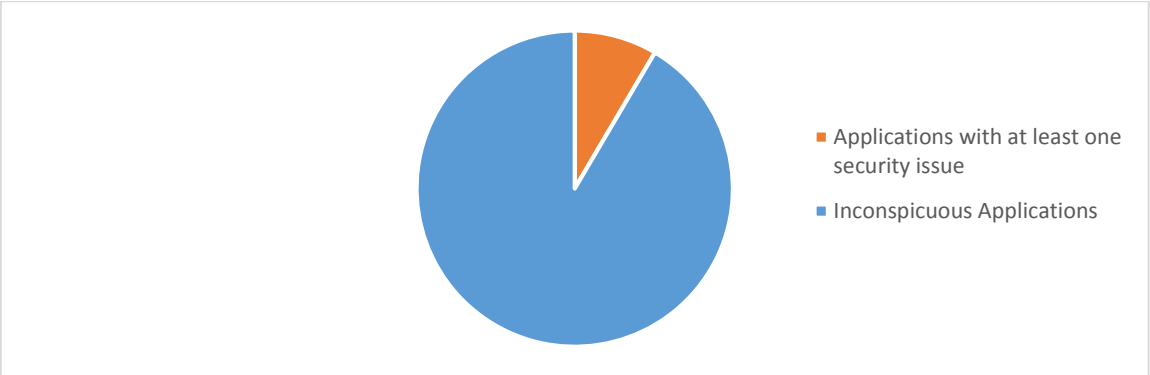

Figure 4: Ratio between inconspicuous applications to applications with at least one security issue.

5. Conclusion and Future Work

Most of the applications found in the Microsoft Store were developed for Windows Phone 7 & 8, only 15% of the applications are targeting Windows Phone 8.1 and later versions. Interestingly, only the new applications targeting Windows 8.1 or later, can be downloaded unencrypted. The older applications, targeting Windows Phone 7 & 8, are only available in encrypted form. The easy procurement process and the absence of protections mechanisms makes it easy for attackers to retrieve the source code of the applications and to steal sensitive information. The analysis revealed that many applications contain security relevant issues. 8.5% of all analysed applications contain at least one security relevant issue. The impact of these issues can range from endangered user's privacy in case of broken cryptography, to denial-of-service attacks in case of API keys where the free usage limited is exhausted by an attacker, to monetary loss of the developer in case of paid APIs.
The current analysis framework can only find simple issues, where no insights of the surrounding code is needed. This limitation will be countered in an upcoming project.

References

[1] A. Marsalek, „Analysis of Windows Phone 8 Applications," Graz, 2015.

[2] Microsoft, „ChainValidationResult enumeration," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/apps/windows.security.cryptography.certificates.chainvalidationresult. [Zugriff am 05 11 2015].