10. Jänner 2017 Studie · Mobile Sicherheit · Web & Cloud · 8 Min. Lesezeit

Automatisierte Analyse von Windows Phone Applikationen

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Quelle https://technology2.a-sit.at/veroeffentlichungen/automatisierte-analyse-von-windows-phone-applikationen/

In diesem Projekt wurden etwa 65.000 Windows Phone Apps analysiert, um sicherheitsrelevante Fehler automatisiert zu identifizieren. Bei etwa 8,5 Prozent der Apps wurden solche Fehler erkannt.

Zur Analyse wurden die .NET Compiler Platform verwendet, mit dem Ziel, sicherheitsrelevante Implementierungsfehler zu finden. Die gefundenen Fehler können in vier Kategorien eingeteilt werden:

- Fehlerhafte Verwendung von Kryptographie
- Unzureichende TLS Sicherheit
- Ungeschützte API Schlüssel
- Sonstige ungeschützte Clientgeheimnisse

Es wurden die Metadaten von 360.000 Applikationen aus dem Windows Phone Store geladen. Von diesen 360.000 Apps konnten 65.000 Apps analysiert werden, die restlichen (älteren) Apps sind verschlüsselt. Nach dem Extraktions- und Dekompilierungs-Schritt standen ca. 6 Millionen einzigartige Quelltextdateien zur Verfügung. 8,5% Prozent aller analysierten Apps enthalten mindestens eine potentielle sicherheitsrelevante Schwachstelle. 4.5% der Apps verwenden Zufallszahlengeneratoren mit fixem Seed. Die folgende Aufzählung zeigt die Anzahl der Apps pro Problemkategorie:

- 181 Apps verwenden feste Schlüssel für kryptografische Operationen,
- 293 Apps verwenden feste Initialisierungsvektoren für kryptografische Operationen,
- 138 Apps verwenden 1000 Iterationen oder weniger für Schlüsselableitungen,
- 500 Apps enthalten unzureichend geschützte Client-Geheimnisse,
- 516 Apps ignorieren bestimmte HTTPS Fehler,
- 4919 Apps enthalten unzureichend geschützte Google API Schlüssel,
- 73 Apps enthalten einen ungeschützten Amazon Web Service Schlüssel und
- 2980 Apps verwenden Zufallszahlengeneratoren mit fixem Seed.

Die Auswirkungen der gefunden Probleme können nicht statisch ermittelt werden, reichen aber von gefährdeter Privatsphäre im Fall von fehlerhafter Verwendung von Kryptographie oder unzureichender TLS Sicherheit bis hin zu finanziellen Belastungen des Entwicklers im Falle von kostenpflichtigen API Schlüsseln. Für mehr Details wird auf den Projektbericht verwiesen.

1. Einleitung

In einem vorhergegangenen Projekt wurde eine große Menge von Windows Phone Applikationen gesammelt, dekompiliert und mit simplen Methoden analysiert. Dabei stellte sich heraus, dass einige sicherheitsrelevante Probleme nur sehr schwer oder gar nicht erkannt werden können. Beispielsweise eignen sich simple Analysemethoden gut zur Erkennung von fest hinterlegten Schlüsseln, welche ein bestimmtes Format erfüllen. Wird aber beispielsweise eine Variable an eine Verschlüsselungs-API übergeben, ist es schwer bzw. teilweise unmöglich mittels simpler Methoden herauszufinden, welcher Wert übergeben wurde bzw. ob dieser fest hinterlegt ist. Ziel dieses Projektes ist die Beseitigung dieser Limitierungen um auch komplexere Sicherheitsprobleme zu erkennen. Die .NET Compiler Platform [1], auch bekannt unter dem früheren Codenamen „Roslyn“ eignet sich gut hierfür, da sie Programmierschnittstellen zur Codeanalyse bietet. Der nächste Abschnitt beschreibt kurz die .NET Compiler Platform und die gängigsten Sicherheitsprobleme. Abschnitt 3 beschreibt die automatische Analyse und Abschnitt 4 die Ergebnisse. Abschließend wird ein Fazit gezogen.

2. Hintergrund

Roslyn stellt eine API zur Verfügung, welche Zugriff auf Informationen aus allen Kompilierungsphasen (Parser, Symbole/Metadaten, Binder und IL Erzeuger) gewährt. Roslyn arbeitet auf Quelltextebene, dies bedeutet, dass die einzelnen Projekte, aus denen sich eine App zusammensetzt als Visual Studio Projekte zur Verfügung stehen müssen. Mittels dieser APIs kann man dann beispielweise definieren, welche Methodenaufrufe relevant sind und dessen Parameter bekommen bzw. zurückverfolgen. In einem vorangegangen Projekt wurden bereits einige Apps aus dem Windows Store geladen, extrahiert, dekompiliert und in eine Datenbank eingetragen. Diese Datenbank wird als Basis für die im Abschnitt 3 beschriebenen Analysen benutzt.

2.1. Sicherheitsprobleme

Die nächsten Abschnitte beschreiben die vier Kategorien von Sicherheitsproblemen, nach denen gesucht wird.

2.1.1. API Schlüssel

Einige Apps binden externe Services ein. Zur Authentifizierung am externen Service wird in den meisten Fällen ein API Schlüssel benötigt. Dieser Schlüssel erlaubt dem Betreiber die Zuordnung der Anfrage zu einem Kunden und schlussendlich die Abrechnung. Einige Betreiber bieten ein bestimmtes Kontingent gratis an, erst bei Überschreitung dieses Kontingents wird die Benutzung des Services kostenpflichtig bzw. deaktiviert, falls keine Zahlungsinformationen hinterlegt wurden. Früher wurden solche Services hauptsächlich in Webanwendungen eingebunden. Dadurch lag der Schlüssel am Server außerhalb der Reichweite des Benutzers. Im Falle von mobilen Apps ergibt sich das Problem, dass der Schlüssel sich nun auf Endkundengeräten befindet und von diesen leicht extrahiert werden kann.

2.1.2. Hinterlegte Authentifizierungs-Daten

Ein weiteres gängiges Problem sind fest hinterlegte Authentifizierungsdaten. Diese werden benutzt um sich gegenüber externen Services zu authentifizieren. Beispiele hierfür sind HTTP Basic Authentication Daten oder OpenID Connect bzw. OAuth Client Geheimnisse.

2.1.3. Mangelhafte Transport Layer Sicherheit

Microsoft bietet Entwicklern die Möglichkeit, bestimmte SSL Serverzertifikatsfehler zu ignorieren. Dadurch kann ein Entwickler beispielsweise definieren, dass auch abgelaufenen Zertifikaten weiterhin vertraut werden soll.

Kettenvalidierungsergebnis Erklärung	Col2
Success	Die Zertifikatskette wurde erfolgreich verifiziert.
Untrusted	Mindestens einem Zertifikat in der Kette wird nicht vertraut.
Revoked	Mindestens ein Zertifikat in der Kette wurde zurückgezogen.
Expired	**Mindestens ein Zertifikat in der Kette ist abgelaufen.**
Incomplete Chain	In der Zertifikatkette fehlt mindestens ein Zertifikat.
Invalid Signature	Die Signatur von mindestens einem Zertifikat in der Kette konnte nicht verifiziert werden.
Wrong Usage	Ein Zertifikat in der Kette wird für einen anderen Zweck verwendet, als durch die Zertifizierungsstelle angegeben.
Invalid Name	Ein Zertifikat in der Kette hat einen ungültigen Namen. Der Name ist entweder nicht in der Liste zulässiger Namen enthalten oder wurde explizit ausgeschlossen.
Invalid Certificate Authority Policy	Ein Zertifikat in der Kette hat eine ungültige Richtlinie.
Basic Constraints Error	Die Basiseinschränkungserweiterung eines Zertifikats in der Kette wurde nicht beachtet.
Unknown Critical Extension	Ein Zertifikat in der Kette enthält eine unbekannte Erweiterung, die als "kritisch" eingestuft wird.
Revocation Information Missing	Es wurde keine installierte oder registrierte DLL zur Überprüfung der Sperrung gefunden.
Revocation Failure	Es konnte keine Verbindung mit dem Sperrserver hergestellt werden.
Other Errors	Beim Überprüfen der Zertifikatskette ist ein unerwarteter Fehler aufgetreten.

Tabelle 1: Kettenvalidierungsergebnisse (Quelle: [2]) Um einen oder mehrere dieser Fehler zu ignorieren, muss ein Entwickler bzw. eine Entwicklerin ein *HttpBaseProtocolFilter* Objekt erstellen und die Fehler die ignoriert werden sollen zur *IgnorableServerCertificateErrors* Liste hinzufügen. Durch Benutzung dieser Funktion kann die gesamte TLS Sicherheit gefährdet werden.

2.1.4. Falsche Verwendung von Kryptographie

Die richtige Verwendung von kryptografischen Funktionen ist essentiell, um ein hohes Sicherheitsniveau zu erreichen. Typische Probleme sind die die Verwendung von fest hinterlegten Schlüsseln, Initialisierungsvektoren, Salt-Werten oder Passwörtern. Dadurch werden verschlüsselte Daten unzureichend geschützt. Beispielsweise kann ein Angreifer selbst per AES verschlüsselte Daten leicht entschlüsseln, wenn der Schlüssel nur aus der App extrahiert werden muss. Ein weiteres Problem ist die Verwendung von zu wenigen Iterationen bei Schlüsselableitungsfunktionen. Auch die Verwendung von Zufallsgeneratoren mit einem festhinterlegten Startwert (Seed) kann gravierende Auswirkungen haben. Dadurch kommt immer dieselbe Sequenz an „Zufallszahlen" zustande. All diese Probleme können die Sicherheit signifikant schwächen bzw. unwirksam machen.

3. Automatische Analyse

Die automatische Analyse kann in zwei Schritte unterteilt werden: Im ersten Schritt werden die Informationen aus der Datenbank ausgewertet und zur Wiederherstellung der Struktur der dekompilierten Apps verwendet. Nach der Wiederherstellung der Struktur können die Apps im zweiten Schritt mithilfe der auf Roslyn basierenden Analysen untersucht werden.

3.1. Wiederherstellung der App Struktur

Im ersten Schritt werden die Daten aus der Datenbank genutzt, um die Struktur der Apps unter Windows wiederherzustellen. Dieser Schritt wird benötigt, da Roslyn auf Projektebene arbeitet und nicht wie die vorhergegangen simpleren Analysen direkt auf den einzelnen Quelltextdateien. Der Download, Extraktions- und Dekompilierungsprozess ist in Abbildung 1 dargestellt.

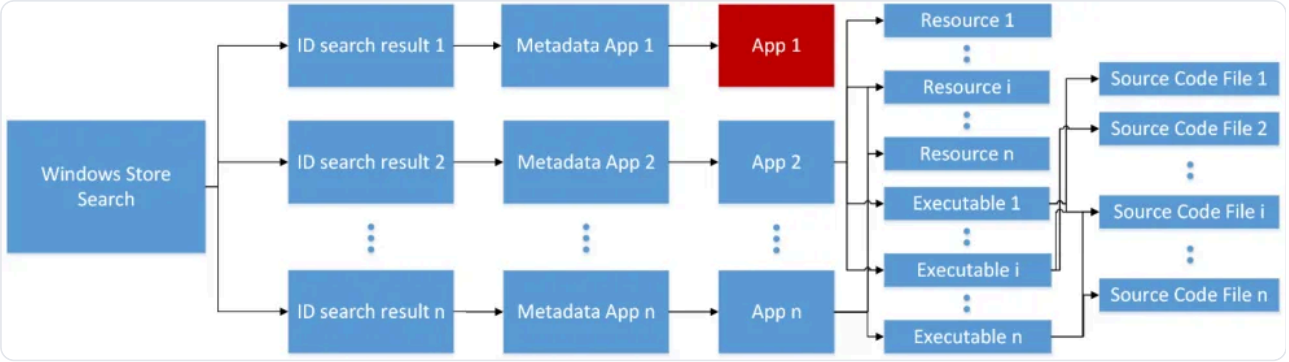


Abbildung 1: Überblick über den App Download-, Extraktions- und Dekompilierungsprozess

3.2. App Analyse Analysis

Nach fest hinterlegten API Schlüssen wird wieder per Regulären Ausdrücken [1] gesucht. TLS Probleme und nach fehlerhaften Verwendungen von Kryptographie wird mittels Roslyn gesucht. Dazu wurde ein im Rahmen einer Bachelorarbeit [3] erstelltes Framework angepasst und erweitert. Alle oben beschriebenen Problemkategorien werden von den Analysen abgedeckt.

4. Ergebnisse

Die Datenbank enthält die Metadaten zu ca. 360.000 Applikationen. Davon handelt es sich bei ca. 61.500 Applikationen um unverschlüsselte Appx bzw. AppxBundle Dateien. Die Applikationen wiederum bestehen aus insgesamt ca. 525.000 ausführbaren Dateien (.exe, .dll), von denen ca. 152.000 einzigartig sind. Von diesen konnten ca. 133.000 erfolgreich dekompiliert werden. Daraus entstanden ca. 6 Millionen einzigartige Quelltext-Dateien. Es wurden 4.919 Applikationen gefunden, welche einen Google API Key eingebunden haben, dies entspricht 8%. 73 Applikationen enthalten einen Amazon Web Service Schlüssel. 500 Applikationen enthalten Client Geheimnisse. 516 Applikationen ignorieren bestimmte Zertifikatsfehler. Abbildung 2 zeigt wie viele Apps welchen Zertifikatsfehler ignorieren. 293 Applikationen verwenden fest hinterlegte Initialisierungsvektoren (IV) und 181 Applikationen verwendeten fest hinterlegte Schlüssel zur Verschlüsselungsaufgaben. 138 Applikationen 1000 100 10 1

491	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Col13
491	491	403	403	403	403	403	403	403	403	403	403	403
491	491	403	403	211	211	211	211	211	211	211	211	211
						68	68	68	68	68	68	68
						68	68	~~19~~	~~19~~			
										~~17~~		~~15~~



Abbildung 2: Anzahl der Apps die bestimmte Zertifikatsfehler ignorieren

1 https://en.wikipedia.org/wiki/Regular_expression verwenden unter 1000 Iterationen für Schlüsselableitungsfunktionen. 2980 Apps verwenden einen fixen Startwert (Seed) für Zufallszahlengeneratoren. Abbildung 3 visualisiert die Ergebnisse.

10000 1000 100 10 1

4919	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Col13	Col14	Co
4919	4919	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~2980~~	~~29~~
				516	516	500	500	500	500	500	500	500	500	50
				516	516	500	500	~~293~~	~~293~~	~~293~~	~~293~~	~~293~~	~~293~~	~~29~~
				516	516	500	500	~~293~~	~~293~~					
				516	516	500	500	~~293~~	~~293~~					
										181		138	138	13
										181		138	138	73



Abbildung 3: Anzahl der Apps mit bestimmten Fehlern

5. Schlussfolgerungen

Die Analyse zeigte, dass viele Applikationen sicherheitsrelevante Probleme haben könnten, die je nach App einer genaueren manuellen Analyse unterzogen werden sollten. Von den 61.500 analysierten Apps enthalten 7.173 (ca. 15%) Probleme, davon sind 8,5% sicherheitsrelevant. Die restlichen Probleme können sicherheitsrelevant sein, je nachdem wofür die Zufallszahlengeneratoren benutzt werden. Die im Vergleich zum vorherigen Projekt verbesserten Analysemethoden finden mehr Probleme und haben trotzdem weniger falsch positive Ergebnisse. Die bestehenden Analysen könnten in einem zukünftigen Projekt weiter ausgebaut werden, um noch mehr Problemkategorien abzudecken.

References

[1] Patric Boscolo, „Die Microsoft .net Compiler Platform (Roslyn) - Teil 1: Grundlagen," 18 02 2015. [Online]. Available: https://www.microsoft.com/germany/techwiese/know-how/diemicrosoft-net-compiler-platform-roslyn-teil-1-grundlagen.aspx.

[2] Microsoft, „ChainValidationResult enumeration," [Online]. Available:

https://msdn.microsoft.com/enus/library/windows/apps/windows.security.cryptography.certificates.chainvalidationresult.

[Zugriff am 05 11 2015].

[3] M. Krawanja, „Windows Phone App Analysis: Automated Security Issue Detection," 2017.

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