

Sichere mobile Apps mit plattformübergreifenden Frameworks entwickeln

Herausforderungen und Lösungsansätze für EntwicklerInnen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Lisa Maria Leonhartsberger, BSc.

Matrikelnummer 01125289

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Mitwirkung: Univ.Lektorin Dr.techn. Katharina Krombholz-Reindl

Wien, 25. Jänner 2018

Lisa Maria Leonhartsberger

Edgar Weippl

Secure Mobile Apps based on Cross-Platform Frameworks

Challenges and Approaches for Developers

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

Lisa Maria Leonhartsberger, BSc.

Registration Number 01125289

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Assistance: Univ.Lektorin Dr.techn. Katharina Krombholz-Reindl

Vienna, 25th January, 2018

Lisa Maria Leonhartsberger

Edgar Weippl

Erklärung zur Verfassung der Arbeit

Lisa Maria Leonhartsberger, BSc.
Ahornweg 16, 3683 Yspertal

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. Jänner 2018

Lisa Maria Leonhartsberger

Acknowledgements

First of all I'd like to thank my advisors Edgar Weippl and Katharina Krombholz-Reindl for their help. They aided me with their constructive feedback and valuable suggestions all the time.

I would also like to thank SBA Research and my colleagues for the great time in the six-months internship. It allowed me to obtain insights and experiences in IT security research, which I wouldn't have made otherwise.

Moreover, I thank my husband, Reinhard Leonhartsberger for his love and moral support through all the years of my bachelor and master studies. I also like to extend my gratitude to my parents, Franz and Brigitte Fichtinger who always enhanced my skills and enabled me a great course of education.

Last but not least, I'm very grateful for all the friends I made on university. We had a lot of useful learning sessions, discussions, interesting projects and fun together.

Kurzfassung

Mobile Endgeräte, wie Smartphones oder Tablets sind heute wesentliche Bestandteile unseres privaten und beruflichen Lebens und werden für verschiedenste Tätigkeiten verwendet. Die verschiedenen Geräteplattformen und Systeme sind eine große Herausforderung für mobile EntwicklerInnen heute. Plattformübergreifende Frameworks mit dem ideologischen Ansatz “Write Once, Run Anywhere” wurden entwickelt, um diese Herausforderungen zu überwinden. Beispiele solcher Frameworks sind Apache Cordova von Apache Software Foundation, Xamarin von Xamarin Inc. einer Tochtergesellschaft von Microsoft und React Native von Facebook Inc.

Die Verwendung digitaler Technologien erzeugt eine enorme Menge an Daten mit sensiblen Informationen über die eigene Persönlichkeit, Vorlieben und Verhalten. Der Schutz dieser Daten vor unerlaubtem Zugriff ist notwendig um die Privatsphäre jedes Einzelnen zu gewährleisten und zu respektieren. SoftwareentwicklerInnen, die diese Technologien und Anwendungen entwickeln, spielen dabei eine wichtige Rolle.

Deshalb untersuchen wir die Probleme und Herausforderungen der EntwicklerInnen um sichere mobile Apps mit plattformübergreifenden Frameworks zu entwickeln. Wir evaluieren offizielle Dokumentationen der Frameworks und sammeln und analysieren Stack Overflow Beiträge, da diese wichtige Informationsquellen für EntwicklerInnen sind. Dabei hat sich herausgestellt, dass in allen drei Frameworks OAuth 2.0 Autorisierung für viele EntwicklerInnen sehr herausfordernd ist. Trotz der vorhandenen Spezifikation ist die Implementierung in der Realität komplex. Deshalb untersuchen wir die verschiedenen Ansätze zur Umsetzung in den Frameworks, welche je nach OAuth Provider unterschiedlich sein können und diskutieren die zugrundeliegenden Sicherheitsaspekte.

Abstract

Mobile devices such as smartphones or tablets are an essential part in our daily lives. To date multiple platforms pose a major challenge for mobile developers. Cross-platform frameworks, based on the so-called “Write Once, Run Anywhere” approach were established to overcome these issues. Some of these frameworks are Apache Cordova from the Apache Software Foundation, Xamarin from Xamarin Inc. a subsidiary company from Microsoft and React Native from Facebook Inc.

Our daily use of digital technologies produces an enormous amount of data including sensitive information about their users’ personality, preferences or behaviors. Protecting these data to avoid unauthorized access is required to ensure and respect everyone’s privacy. Software developers that are implementing and establishing these technologies and applications thereby play an important role.

Thus, we investigate challenges for secure mobile app development from the developers’ perspective built with cross-platform frameworks. We evaluate the frameworks’ official documentations and collect and analyze posts from Stack Overflow as they are major information sources. We found that authorization with the OAuth 2.0 protocol is a big challenge for developers with respect to all three frameworks. Despite of a specification, the implementation in reality is complex. Thus, we investigate different implementation approaches across the cross-platform frameworks, which may be different depending on the OAuth provider and furthermore discuss their impact on app security.

Contents

Kurzfassung	ix
Abstract	xi
Contents	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	1
1.3 Aim of this Work	2
1.4 Methodology	2
1.5 Structure	3
2 Related Work	5
3 Fundamentals	11
3.1 Mobile Development Approaches	11
3.2 Cross-Platform Frameworks	11
4 Methodology	19
4.1 Security Handling	19
4.2 Developers' Challenges	20
4.3 User Study	29
5 Results	33
5.1 Security Handling	33
5.2 Meta Topics	43
5.3 Posts about Authentication & Authorization	46
5.4 Online Survey	63
6 Discussion	73
6.1 Security Handling	73
6.2 Developers' Challenges	74
6.3 Authentication and Authorization	74
	xiii

7 Conclusion	79
A Online Survey	81
List of Figures	101
List of Tables	103
Bibliography	105

Introduction

1.1 Motivation

The first smartphone as we know it today was an iPhone released only 10 years ago. Since then a lot of innovations and developments have been happened, which changed our daily lives enormously. Today, we use the mobile phones for multiple purposes and not only for calling and texting. Smartphones and tablets replace digital cameras, recorders, navigation systems or remote controls of TVs. Browsing, shopping, gaming, streaming music or videos, reading newspapers, books or social media posts are some more activities we use our mobile devices for. So to say, we are producing a mass of data with all these activities that are processed, recorded, stored or even analyzed by mobile applications on the device or by remote servers the apps are communicating with.

1.2 Problem Statement

The large number of daily activities with mobile devices involved yields to new challenges in the development of applications for these devices. Multiple platforms and operating systems are available which is a major challenge for mobile software developers to date [32]. Many applications should target users of different platforms. However programming languages, APIs or tools vary between platforms, such as Android, iOS, Windows or Blackberry. But how can software developers meet this requirement? Implementing an application for each platform in its native language needs much time and effort and requires knowledge in several languages and platforms. Moreover, maintenance and support of multiple implementations in parallel is very expensive.

“Write Once, Run Anywhere” seems to be a very good approach to overcome these issues. Several cross-platform frameworks like Apache Cordova from the Apache Software Foundation, Xamarin from Xamarin Inc. a subsidiary company from Microsoft and

React-Native from Facebook Inc. have been developed. They are based on different concepts, but their goal is to write common code that can be compiled or packed to run on multiple platforms.

As mobile devices are used for various personal and professional activities, it is obvious that a lot of sensitive data are created. Probably, many of them should not become public like messages and communications, family photos or videos, watchlists on streaming services or the credit balance. Data leaks and unauthorized access of such resources can cause big harm for individuals, enterprises or organizations. Thus, software developers and enterprises are responsible for the protection of their user's data by providing and implementing respective security functions in their applications. As the world is not only black or white, it's wrong to just say the developers are solely responsible for the security of apps and data, but as major stakeholders in the software development process they play an important role.

1.3 Aim of this Work

The security and privacy of mobile apps built with cross-platform frameworks is influenced by many factors, such as the end user or the platform and operating system of the device. But also software developers and their decisions about the software architecture, design patterns, frameworks, libraries and developing tools they use and include, contribute much to a secure application. Thus, our aim is to investigate the security in cross-platform mobile apps from the developers' perspectives to get a deeper understanding about the issues and problems they are faced during development and which impact the cross-platform frameworks have. This leads to several questions. For instance, do the frameworks provide sufficient information for developers and features to protect user data. Moreover, which resources are available and on which do developers rely on? Are developers aware of their responsibilities towards their users and is this reflected in their actions?

1.4 Methodology

Our methodology to investigate security in cross-platform mobile apps from the developers' perspectives contains the following three major steps.

- At first, we evaluate how the three cross-platform frameworks Apache Cordova, Xamarin and React Native deal with security and privacy and how they communicate security best-practices to app developers. We examine their official documentations and websites to analyze which information they provide and which topics they leave out.
- In a second step we investigate challenges from the developer's perspective by collecting and analyzing posts on Stack Overflow. The tools we implemented in

the course of these analysis are publicly available on <https://github.com/lislehabe/stack-analysis-tools>.

- Based on the results of step two, we finally conduct an online survey to gain insights into the developers' experiences, knowledge and implementations of OAuth and OpenID Connect in mobile development.

1.5 Structure

The remainder of this thesis is structured as follows. State of the art research in context of this work is described in Section 2. Fundamentals about the cross-platform frameworks Cordova, Xamarin and React Native are provided in Section 3. Section 4 describes the three steps of our methodology to evaluate the frameworks' and developers' perspectives on security and privacy in mobile apps. The results of analyses and evaluations are described in Section 5 and in Section 6 the results and findings are discussed. Finally, Section 7 concludes the thesis.

Related Work

In this chapter we discuss existing publications in the research fields of this thesis. Challenges of mobile app development, Stack Overflow posts and cross-platform frameworks have been studied several times from different aspects. Moreover, various attacks on mobile platforms have been detected and published. OAuth protocol and OpenID Connect specification have mainly be studied for Web platforms, but there are also some publications based on the mobile platform.

Mobile App Development. Main challenges of mobile app development were studied by Joorabchi et al. [32] 2013. They conducted a qualitative study interviewing 12 senior mobile developers and an online survey with 188 participants. Their results showed that the biggest challenge for developers was dealing with multiple platforms because of fragmentation across and within platforms. It means, that mobile platforms tend to differ regarding user interface, user experience, programming languages, Application Programming Interfaces (APIs), Software Development Kits (SDKs) and supported tools. Moreover, various devices or operating system levels also differ within one platform. Testing and monitoring apps across multiple platforms is also more complicated and challenging as useful tools are currently missing.

In 2017 Francese et al. [24] also conducted a qualitative study to identify main aspects of mobile app development interviewing software managers and executing a survey with software professionals. They found that mobile apps were mostly developed by junior developers and although, there is no cross-platform framework that is considered as the best, they are widely adopted. Fragmentation and less support for testing are still big challenges, as already investigated in [32].

Ali et al. [3] conducted a different approach to study developer challenges of cross-platform apps and analyzed apps from the Google Play and Apple store, to detect commonalities and differences across platforms. They compared store attributes, such as versions,

stars and prices and analyzed the content of textual user reviews to find the major user concerns or complaints.

Stack Overflow Data Analysis. Various works [34, 5, 92, 57, 91, 44, 9] analyzed textual contents of Stack Overflow posts by the use of natural language processing. They aimed to get an overview about discussed topics and insights into developers' concerns and therefore used topic modeling with latent Dirichlet allocation (LDA).

Yang et al. [91] focused their large scale study on security questions in general and concluded the top five security categories discussed on Stack Overflow were web security, mobile security, cryptography, software security and system security.

Rosen and Shihab [44] also conducted a large scale study on Stack Overflow posts using topic model LDA, but focused on mobile developer questions. They investigated issues and topics in context of the mobile platforms Android, iOS and Windows Phone Mobile as well as mobile development in a whole. App distribution was identified as the most popular topic across all platforms, while popularity was measured using view counts of questions. Questions related to APIs were identified as the most difficult ones, which means the median time between creating a question and getting an accepted answer is higher than in other topics. Moreover, Rosen and Shihab labelled a random sample set of questions by their types into *how*, *why* and *what* classes to investigate its distribution. Most of the questions were *how* questions, followed by *why* and *what*. They also argued that mobile developers primary looks for working examples.

Some research about the impact of Stack Overflow on software development, software quality and security exist.

Acar et al. [1] conducted a lab study to investigate differences on the security level of code depending on the used resources to look up. Therefore, participants had to implement Android issues in limited time using only Stack Overflow, official Android documentation or books and one group could choose the resource themselves. As a result, the code of developers using only Stack Overflow was significantly less secure than the code of developers using official Android documentation or books. On the other hand, participants that only looked up at the official Android documentation implemented less functional code than those using Stack Overflow.

Fischer et al. [23] designed, implemented and evaluated a fully automated large-scale processing pipeline to detect and measure code snippets published on a platform being copied and pasted into real-world apps available on a software repository. In detail, they found security-related Android code snippets from Stack Overflow in 15.4% of 1.3 million Android apps from Google Play store, while 97.9% of them contained at least one insecure code snippet.

An et al. [4] studied reused code snippets from Stack Overflow in real-world Android apps focused on correct licensing as all posts on Stack Overflow are published under *Creative Commons Attribute-ShareAlike 3.0 Unported* license. They aimed to raise awareness and

argued developers should pay more attention on right licensing when copying & pasting code, as they detected more than 1200 potential license violations.

Cross-Platform Frameworks. Many surveys and evaluations [11, 30, 89, 47, 2, 61, 13] about cross-platform development approaches and frameworks have been published in the last years. To the best of our knowledge, no such security evaluation as we conducted has been published before.

Xanthopoulos and Xinogalos [89] presented the results from a survey and analysis of cross-platform approaches in 2013. They highlighted advantages and disadvantages of each approach.

A comprehensive survey and introduction about cross-platform approaches is provided by El-Kassas et al. [13]. They described each approach in detail with its pros and cons and categorized them. Moreover existing frameworks and tools based on these approaches are listed, described and compared.

Heitkoetter et al. [30] evaluated apps implemented with Web technologies, PhoneGap, Titanium, Android and iOS on infrastructure and development criteria, such as license, supported platforms, look and feel, GUI design, ease, speed and cost of development, maintainability and scalability and resulted PhoneGap has to be preferred.

Ahti et al. [2] provide an evaluation framework of cross-platform development tools, that use qualitative measures (i.e. app starting time, RAM memory usage, app size) and quantitative measures (i.e. user experience, appearance, ease of development) to analyze it against native development. Moreover, they validated the framework with a case study using PhoneGap.

Another comparison of PhoneGap and Titanium is provided in Solanky et al. [47].

Dalmasso et al. [11] classified so-called Write Once Run Anywhere (WORA) tools and provided a survey of some cross-platform frameworks, such as PhoneGap, Titanium and Sencha Touch. They built Android test apps with PhoneGap and Titanium to evaluate the frameworks in terms of CPU performance, memory usage and power consumption. Based on their results they argued that PhoneGap uses minimum memory, CPU and power, but provides only a very simple user experience.

Willox et al. [61] performed an in-depth performance analysis of multiple cross-platform apps in comparison to native apps. They implemented one testing app in native Android, iOS and Windows Phone as well as with 10 cross-platform frameworks, two of them were Apache Cordova and Xamarin.

Martinez and Lecomte [37] introduced their future work as they want to implement an automated bug repair tool for mobile apps implemented with cross-platform framework like Xamarin or React Native.

Mobile Security. A survey about Android security is presented in [22]. Faruki et al. [22] depicts security issues and existing security enforcement mechanisms. They also

discuss penetration and stealth techniques of existing Android malware applications. An overview about analysis and detection approaches as well as state of the art tools are provided.

Buhov et al. [8, 7] wrote about network security challenges in Android applications. They took a focus on wrong SSL implementations in Android applications, which is still an existing problem. Developers often fail to implement a proper certificate validation and thus the whole network communication of the app gets insecure. Buhov et al. [7] provide a device-based solution that affects the user and not the developer. They combine certificate pinning with dynamic instrumentation techniques to automatically correct developer's mistakes and re-establish a secure network communication.

Mobile apps implemented with cross-platform technologies, such as HTML5 originate new or modified security issues. For example Jin et al. [31] found a new form of code injection attack. It has the same fundamental cause as Cross-Site Scripting, but uses more channels to inject code, e.g. SMS, barcodes, RFID tags, metadata fields in media files such as JPEG or MP3 and so on [31]. When a mobile app gets data from outside by one of these channels and displays them inside the HTML5 page, unvalidated code can be executed with the permissions of the app. In worst case, this means injected code can be executed with access to local resources.

One problem in hybrid apps is the mixture of access control policies as indicated in [25]. Web code is governed by the same origin policy, while local code is governed by the access-control policy of the operating system. When a framework bridge is not properly protected by the same origin policy, a "fracking" attack is possible. This means foreign-origin web content is included into the hybrid app, drills through the layers and has directly access to the device resources. Georgiev et al. [25] studied fracking vulnerabilities in free Android apps based on PhoneGap and presented an platform-independent defense method that is compatible with any framework and embedded browser.

In [40] mobile web app vulnerabilities were studied. Mutchler et al. [40] analyzed the real occurrences of untrusted web content loads, stateful web navigation to untrusted apps and URL load leaks to untrusted apps. They found out that 28% of the studied apps have at least one vulnerability and finally, they offered changes to the Android APIs to mitigate them.

Habchi et al. [28] studied code smells in iOS apps and compared them against Android apps. They find that apps on iOS have less code smells than on Android due to the platform and not the programming language.

OAuth and OpenID Connect in the Mobile Environment. The OAuth protocol as well as its challenges and pitfalls have been studied by many others. Most of them investigated only the Web platform, while a few investigated OAuth in mobile applications [10, 59, 58, 90].

Chen et al. [10] studied three OAuth protocol flows (OAuth 1.0, OAuth 2.0 implicit grant, OAuth 2.0 authorization code grant) and their usage in mobile applications, since

there are many differences to the web. They detected common misunderstandings about the flows among real-world developers, such as storing access tokens locally on devices, distinguishing authentication and authorization or redirecting secret token. 59.7% of 149 mobile apps contained faulty and vulnerable implementations.

Wang et al. [59] introduced a framework called `AuthDroid` to detect vulnerable OAuth implementations in Android applications using static analysis and dynamic network traffic analysis. They investigated 15 OAuth service providers and more than 4000 mobile apps from the Chinese market and detected that 86.2% of the apps employing OAuth services were vulnerable.

Wang et al. [58] investigated OAuth-based authentication techniques, such as Single-Sign-On (SSO) on the Web and mobile platforms, as OAuth was initially not intended for it. They analyzed real-world SSO mechanisms by considering identity providers and relying parties as black boxes and detected their used authenticators and discussed several vulnerabilities.

Fundamentals

In this chapter we provide some essential fundamentals in context of this thesis. We describe several approaches to build cross-platform mobile apps and give an introduction about the architectures of Apache Cordova, Xamarin and React Native.

3.1 Mobile Development Approaches

A mobile app can be implemented in various ways based on different approaches. Each approach has its individual advantages and disadvantages as discussed in [11, 89, 47].

Native applications are developed specifically for one operating system. Each platform has its major programming language, as it is Java for Android, Objective-C or Swift for iOS and C# for Windows [11]. A web app built with web technologies, such as JavaScript (JS), Hyper Text Markup Language (HTML) and Cascading Style Sheets (CSS) is running within a browser on desktop computers or on mobile devices. In case of a mobile device it is also called mobile web app. A combination of native and web apps is established with so called hybrid apps. It is actually a web app running within a native container, such as the UIWebView in iOS or WebView in Android, which enables hardware and data access of the device. In interpreted apps the user interface consists of platform-specific native components, but the application logic is written in a cross-platform language like Java, Ruby or XML [89].

3.2 Cross-Platform Frameworks

Based on the different approaches in Section 3.1, there are various developing tools for mobile apps. In this thesis, we focus on the following three frameworks. Apache Cordova and Xamarin belong to the most popular installed mobile SDKs of 2016 [48] and React Native of Facebook is a rather new framework, but gets more and more popular. In the following we describe their basics and principles.

3.2.1 Cordova

Apache Cordova, called Cordova in the remaining thesis, is an open-source mobile development framework, that allows using standard web technologies, such as HTML, CSS and JS to write hybrid apps.

The project is supported by the Apache Software Foundation and the source code is available at github.com¹. Cordova was first released in November 2012. All our research and description is based at least on release number 6.4.0. Cordova supports several mobile platforms, such as Android, iOS, Windows Phone 8, Windows 8.1, Universal Windows Platform (UWP), Blackberry 10, Ubuntu and OS X.

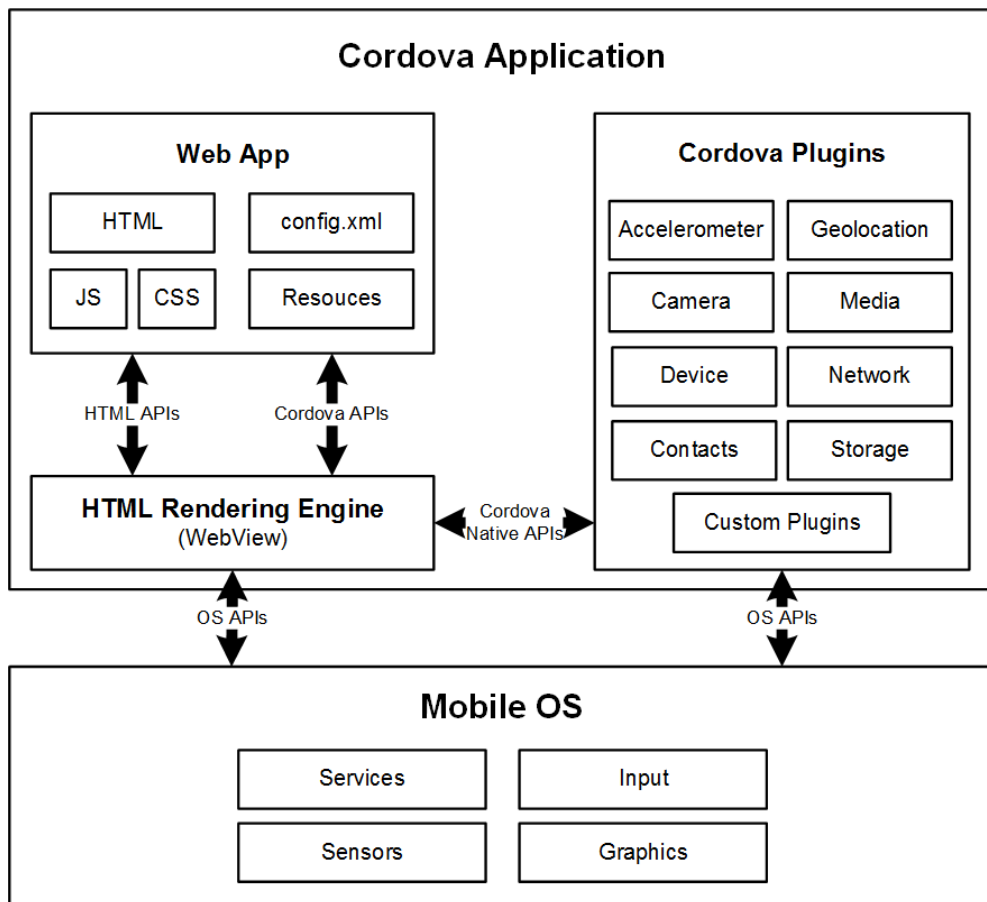


Figure 3.1: Architecture of a Cordova Application (cf. [50])

A Cordova application consists of three main components, the Web App, an HTML Rendering Engine and Plugins. Figure 3.1 shows this architecture and how Cordova interacts with the mobile operating system.

¹<https://github.com/apache?q=cordova>

Web App. The web app contains the application code. Basically, the app is a web page that references CSS and JS or any other resources, e.g. images, media files and so on. Additionally, a configuration file to define general app information and certain app behavior is included [50].

HTML Rendering Engine. An embedded browser that renders web pages and interprets JS is the second component of a Cordova app.

The Cordova WebView depends on the WebView of the underlying operating system. This means, that supported functionalities, behavior and performance vary from device to device. Therefore, providing same user experience and correctness of an app on each device is a great challenge for developers.

The Crosswalk Project² provides a WebView based on Google Chromium that is deployed together with an app. This enables developers focus on only one web runtime, which behaves the same way on every device regardless of manufacturer and operating system. Crosswalk is compatible with Cordova apps, but also with native apps for Android, iOS or Windows.

Plugins. Interfaces to the operating system provide access to hardware components and device capabilities, such as camera, fingerprint sensor, speakers, file system, battery, camera, contacts and many others. Cordova plugins are interfaces that invoke native code from JavaScript.

A collection of basic plugins, such as camera, file, media, contacts, battery status and many more are maintained by the project. They are bundled as so-called “Core Plugins” and are available for Android, iOS and Windows 10.

If the existing plugins are insufficient, developers can implement their own plugins. Many of such third-party plugins are published at the package manager npm. A plugin search is also available at Cordova’s website.

At the beginning of a Cordova project, there is no plugin included, not even the core plugins. All required plugins must be added manually to a project [50].

3.2.2 Xamarin

The basic idea of Xamarin is writing native apps for Android, iOS and Windows in C# language. Xamarin is developed by Xamarin Inc., a company with more than 350 employees [62]. In February 2016 Xamarin Inc. was acquired by Microsoft [27].

Figure 3.2 describes Xamarin’s principle of code sharing for cross-platform development. Xamarin apps are so-called generated apps that include additional SDKs, e.g. Xamarin.Android or Xamarin.iOS to make them running on the native platform. Code for application logic and user interface is written in C#, while common code parts can be shared across the different platform projects by shared projects or portable class libraries.

²<https://crosswalk-project.org/>

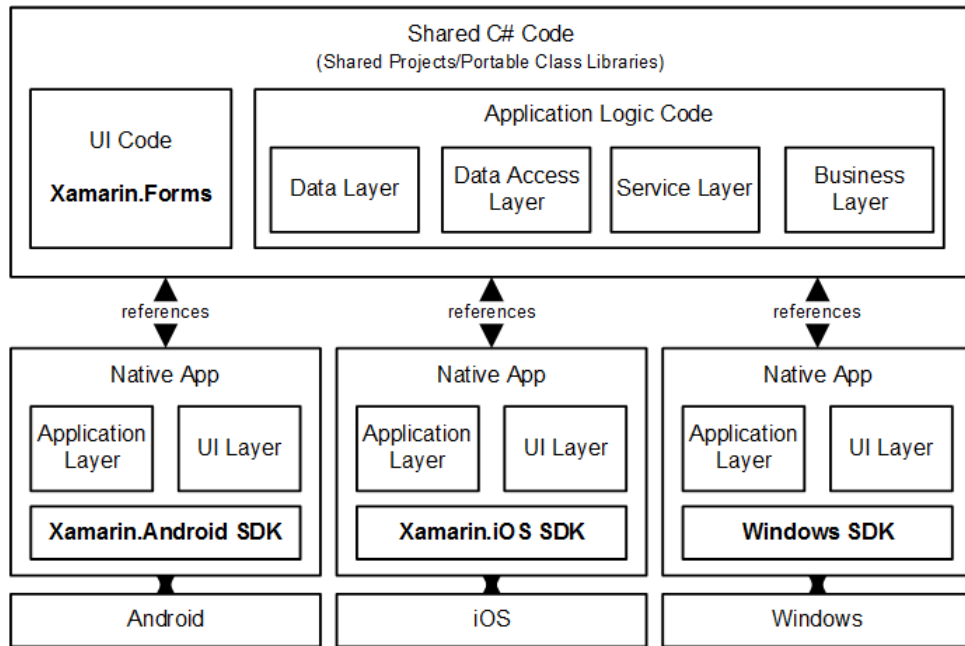


Figure 3.2: Xamarin - Shared Code (cf. [67])

Shared Asset Projects. Each platform requires its own application project and common code is organized within a Shared Project. Moreover, compiler directives allow different and platform-specific code paths as well [67].

Portable Class Libraries. A portable class library is a .dll compiled for various platforms on which the library should run. A “Profile” identifier describes the supported platforms [65].

Xamarin.Forms. Xamarin.Forms is a cross-platform User Interface (UI) toolkit that enables developers writing UIs with XAML and C#. Xamarin.Forms components, such as pages, layouts, views and cells are mapped to the native equivalent at runtime. This means that the native control is actually rendered while one common source code is written by the developer [63].

Xamarin.Mobile. Xamarin.Mobile offers cross-platform APIs for common device functionalities across Android, iOS and Windows. Actions, such as taking a photo or reading contacts can be implemented at once for all three platforms [70].

Mono. Xamarin applications are built and executed with Mono. Mono is an open source version of the .NET Framework based on ECMA standards for C#. It supports multiple platforms, e.g. Linux, Microsoft Windows, Mac OS X, BSD, Nintendo Wii, Sony PlayStation 3, Apple iPhone, Android and more.

Two important components of Mono are the `C# Compiler` and `Mono Runtime`. The compiler compiles `C#` code into Microsoft Intermediate Language (MSIL) and the runtime implements the Common Language Infrastructure (CLI), provides a Just in Time (JIT) compiler, an Ahead of Time (AOT) compiler, a library loader, the garbage collector, a threading system and interoperability functionality [39].

For `Xamarin.Android` applications, the runtime compiles the MSIL to native code with the JIT compiler. For `Xamarin.iOS` applications however, the AOT compiler is used to produce a native iOS binary, because Apple disallows the execution of dynamically generated code on a device [66].

Xamarin.Android. Basically, a `Xamarin.Android` application package looks like a normal Android application package. It is a ZIP container with the `.apk` file extension, that additionally includes assemblies and native libraries for Mono environment [69].

As depicted in Figure 3.3 Mono runs side-by-side with the Android Runtime (ART). Managed Callable Wrappers (MCWs) and Android Callable Wrappers (ACWs) act as Java Native Interface (JNI) bridge between native and managed code, e.g. all `Android.*` namespaces are MCWs. They handle the conversion from managed to Java types and invoke corresponding Android methods via JNI [69].

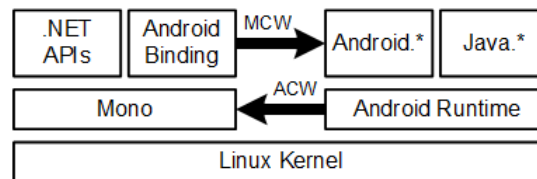


Figure 3.3: Architecture of Xamarin.Android (cf. [69])

Xamarin.iOS. Figure 3.4 shows that Mono runtime and Objective-C runtime work side-by-side and bindings allow the interaction between these two environments. Selectors are used to expose Objective-C to `C#` and registrars are used for the other direction, which means they expose managed code to Objective-C [66].

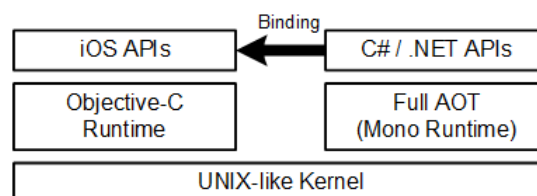


Figure 3.4: Architecture of Xamarin.iOS (cf. [66])

3.2.3 React Native

React Native³ is an open-source framework to implement native apps with JavaScript and React⁴, which is a JavaScript library to build user interfaces. Both frameworks, React and React Native are maintained by Facebook. As React Native follows the principle of “Learn once, write anywhere” it supports multiple mobile platforms such as Android, iOS and UWP.

The basic concepts of React Native are the same as of React as we describe in the following. Although the UI components are developed in JavaScript, they are running as native components in the application and not as web components.

Basics. A *component* is anything that can be seen on the screen. The JavaScript object requires a `render()` function that returns some JSX to render [16]. React Native provides predefined components, such as a button, text or view component, that are rendered into native components. JSX is a special templating language that enables writing markup language inside code [16, 20]. Attributes and parameters that are used to customize components are called *props*. They can be accessed via `this.props.xxx` in the JSX templates [14]. Data that change during a component’s lifetime are handled via states. A *state* should be initialized in the constructor of the component while changes should only be propagated by calling the `setState()` function. States can be accessed in templates with `this.state.xxx` [15].

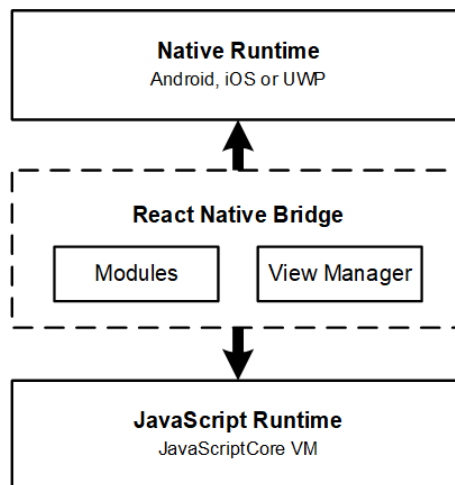


Figure 3.5: Architecture of React Native (cf. [33])

Architecture. Figure 3.5 shows the architecture of React Native. The gap between JavaScript and native environment is closed with a React Native bridge developed in

³<http://facebook.github.io/react-native/>

⁴<https://facebook.github.io/react/>

C++ for iOS, Java for Android and C# for Windows. This means, the bridge handles communication and data exchanges across these two worlds. The React Native app is not running in a WebView, but it is running in a virtual machine and controls native UI [33].

Two important components belonging to the bridge are modules and view managers. Modules provide states and methods that can be called from the JavaScript side, while the mapping from JavaScript views to native views is defined in a view manager. Moreover, app developers can extend the bridge with their own modules and view managers [33].

Methodology

Our methodology to investigate and evaluate security in mobile apps built with cross-platform frameworks consists of three major stages.

In a first stage we evaluate how the three cross-platform frameworks (CPFs) Apache Cordova, Xamarin and React Native deal with security and privacy and how they communicate it to the app developers. Therefore we examine their official documentations and websites to analyze which information they provide and which topics they leave out.

In a second stage we investigate developers' challenges and questions related to the CPFs. Stack Overflow (SO) is one of the most used online discussion forums for developers to date. That's why we collect and analyze posts about the frameworks from SO to detect developers' usual issues.

Based on analysis results of second stage, we conduct an online survey to gain insights into developers' experiences, knowledge and implementation of OAuth and OpenID Connect in mobile development.

4.1 Security Handling

To a certain degree, a developer depends on the tools she is using concerning functionality and also security and privacy it offers. But also its documentation is of high priority. On the one hand it is required to describe the tools' features and on the other hand it can create awareness for non-functional requirements, such as tips for maintainability, availability, performance improvements, privacy or security. So to say the documentation may give insights into the frameworks' relation and open-mindedness to these topics.

Therefore we investigate official documentation and websites of Cordova, Xamarin and React Native to discover and compare which security and privacy topics are covered and which are missing.

4.2 Developers' Challenges

As a first step we investigated the documentation of the three CPFs while in this phase we focus on the active developing process using these frameworks. We explore errors, problems, challenges and issues developers are faced with during coding and what they need help for from the community.

Stack Overflow (SO) is the most frequently used online discussion forum for developers [23] to date. It is free in use and each registered user can ask questions or provide answers to an existing question. Even unregistered and anonymous users can search posts to find solutions for their own issues. Due to its popularity we choose it as primary source to collect and analyze posts and code snippets to gain deeper insight in developers' challenges in cross-platform mobile development with Cordova, Xamarin or React Native.

4.2.1 Data Crawling

We conducted the data crawling process in two stages. Due to Stack Overflow's tagging mechanism, the first step was to collect all tags related to the three CPFs and then to fetch all questions per relevant tag. To automate these steps, we implemented a tool called *Stack Crawler*, which is a Node.js application that executes HTTP requests against Stack Exchange API v2.2 and inserts the collected data into a PostgreSQL 9.6 database. The tool uses PostgreSQL client `node-postgres`¹ and native Node.js modules² `http`, `request` and `zlib`. Source code of this and the following described tools are publicly available on Github³.

Stack Crawler provides a function `getTags(keyword)` to fetch all tags from Stack Overflow that include a certain keyword in their names and stores them in database table `tags`. A tag includes information, such as its name, synonyms, number of posts that are tagged with it and a timestamp when tag was fetched from SO and the keyword it was found for.

For the second stage of data crawling, Stack Crawler has a function `getPosts(keyword)` that fetches all SO posts for all stored tags based on the keyword. Therefore the crawler sends a request for each saved tag to an API endpoint `/question` to receive all questions being tagged with it. A self-created filter is included in the requests to ensure that all answers and comments of a question, all comments of an answer, the body, number of up and down votes and many more fields of a question are returned by the API. Question data from the API are stored as JSON document in the `json_questions` table in the same way as they are received. Additionally, each table row also includes a timestamp of the request and the original tag the question was searched for.

Stack Exchange API supports paging and to decrease the runtime of crawling, page size is set to maximum, such that 100 questions are returned at once. An access token for

¹<https://www.npmjs.com/package/pg>

²<https://nodejs.org/dist/latest-v4.x/docs/api/>

³<https://github.com/lislehabe/stack-analysis-tools>

the API is also included to the requests to increase the daily rate limit of the API from 300 to 10,000 requests per day.

4.2.2 Dataset Description

We crawled a dataset of Stack Overflow posts about the three CPFs Apache Cordova, Xamarin and React Native on June 13th, 2017 using Stack Crawler of Section 4.2.1.

First we fetched tags for each framework and removed irrelevant tags manually from the database, such as “*reactive-programming*” or “*reactive-cocoa*”. Table 4.1 shows the number of received tags and after cleaning. There are 27 tags for keyword “cordova”, such as *apache-cordova*, *cordova*, *cordova-plugins*, *ngcordova*, *cordova-x.x.x* while x.x.x stands for a version number of Cordova and many others. Some of the 28 tags for keyword “xamarin” are *xamarin*, *xamarin.android*, *xamarin.ios*, *xamarin.forms*, *xamarin-studio*. The 20 tags for keyword “react” include tags such as *react-native*, *react-native-android*, *react-native-ios*, *react-native-listview*, but also tags such as *reactjs*, *react-router*, *react-redux*, *react-jsx*.

Keyword	Tags	
	fetched	cleaned
cordova	27	27
xamarin	28	28
react	30	20

Table 4.1: Number of SO Tags

In the second step we fetched all posts of each of the 85 framework tags, which took about 55 minutes and 34 seconds. Table 4.2 shows that there are 748,131 posts in total. Stack Overflow posts can be categorized in four different types. About 20% of it are questions, while 23% are answers. Comments of questions and answers are each about 29%.

Post Type	Frequency	
	absolute	relative
questions	147,528	19.7 %
answers	168,582	22.5 %
question comments	213,764	28.6 %
answer comments	218,257	29.2 %
Σ Posts	748,131	100.0 %

Table 4.2: Post Type Frequency

Distribution across framework tags. Table 4.3 shows the number of posts per type of each framework in our dataset. There are 275,414 posts about Cordova, 176,832 about Xamarin and 66,405 about React Native. Most of them are only about one framework. However, there are a few posts dealing with two frameworks, such as Cordova and Xamarin or Cordova and React Native. Less posts are about React Native and Xamarin, but there are no posts about all three frameworks.

Concerning React Native posts, we refer to posts whose tags include **react-native**. As the scope of our study is about frameworks for mobile app development, we do not further consider posts with **react** tags. Therefore most of the posts are about Cordova, followed by Xamarin and React Native last. However, if considering ReactJS and React Native posts together, they would be the most ones.

Tags	Questions	Question Comments	Answers	Answer Comments	Σ Posts
cordova	54,301	79,654	62,515	78,547	
cordova, react-native	28	24	32	37	
cordova, xamarin	40	56	69	111	
cordova, xamarin, react-native	0	0	0	0	
Σ Cordova	54,369	79,734	62,616	78,695	275,414
xamarin	34,892	51,151	39,377	51,116	
xamarin, cordova	40	56	69	111	
xamarin, react-native	6	1	5	8	
xamarin, cordova, react-native	0	0	0	0	
Σ Xamarin	34,938	51,208	39,451	51,235	176,832
react-native	15,151	16,746	15,792	18,575	
react-native, cordova	28	24	32	37	
react-native, xamarin	6	1	5	8	
react-native, cordova, xamarin	0	0	0	0	
Σ React-Native	15,185	16,771	15,829	18,620	66,405

Table 4.3: Posts across Framework Tags

Distribution of answered questions. A question can have several answers and if one of them is marked as the accepted answer or the score of any answer is greater than zero, then the property *is_answered* is true. Table 4.4 shows that for each framework more than half of the questions are answered. The answering rate of Xamarin questions

is the highest with 60%, while Cordova with 52.4% and React Native with 51% are close to each other.

It is interesting that four questions are marked as answered, although no answer post is available for these questions. A closer look shows that they are duplicated questions of another ones. Since the original question is marked as answered, the same is true for the duplicate.

14/15% of the answered questions do not have a certain answer selected as the correct one. This means, they are marked as answered due to scoring feature of Stack Overflow.

answers available	is an- swered	marked answer	Cordova		Xamarin		React-Native	
			absolute	relative	absolute	relative	absolute	relative
✗	✗	✗	12,996	23.9%	7456	21.3%	4236	27.9%
✓	✗	✗	12,867	23.7%	6516	18.7%	3200	21.1%
Σ Unanswered			25,863	47.6%	13,972	40.0%	7436	49%
✗	✓	✗	3	0.0%	0	0.0%	1	0.0%
✓	✓	✗	8384	15.4%	4867	13.9%	2172	14.3%
✓	✓	✓	20,119	37.0%	16,099	46.1%	5576	36.7%
Σ Answered			28,506	52.4%	20,966	60.0%	7749	51.0%

Table 4.4: Rate of (Un-)Answered Questions across Tags

4.2.3 Data Analysis

We used several tools to analyze the data of SO. For simple querying and database management of PostgreSQL database we used `pgAdmin4`. Query scripts for quantitative analysis, such as described in Section 4.2.2 were executed with the console tool `psql`.

The analysis of post contents was conducted in two phases. In phase one we categorized the posts to get a thematic overview about the posts for each framework. Therefore we used topic modeling with LDA and manually filtered and grouped 200 fine-grained topics into meta-topics. In second phase we focused on one meta-topic and investigated for each framework 100 posts of this meta-topic. The manual content analysis was conducted with a self-implemented web tool called *Stack Web Viewer*.

Topic Modeling

In order to investigate issues, challenges and problems developers are talking about on Stack Overflow, we used natural language processing. Topic modeling based on latent Dirichlet allocation (LDA), introduced by Blei et al. [6], is a popular technique to find

discussion topics in natural text documents, as discussed in various publications described in Section 2.

LDA is a statistical modeling technique that infers latent topics to describe content of text documents [44]. The idea beyond topic models is that documents contain a mixture of topics, while each topic is a probability distribution over words [49]. LDA is an extended topic model with prior parameters called hyperparameters. First parameter α describes topic distribution among documents. A high value leads to a smooth distribution while $\alpha < 1$ enables that a document gets assigned less topics, but more specific ones. Second hyperparameter β describes the word distribution among topics. With an high β value each topic may contain most of the words and with a low β value a topic may contain just a few of the words [6].

We used MALLET version 2.0.8 [38] to apply LDA with Gibbs sampling algorithm on our SO posts. Therefore we implemented a simple Java tool that loads data from PostgreSQL database, performs modeling and sampling using MALLET and exports results into **.txt* files. We used all posts about Cordova, Xamarin and React Native as input. Security questions are only rarely declared as such and therefore limiting the posts by criteria, such as tags or a keyword list would falsely remove posts too early. Moreover considering relations of topics on the whole set instead of a subset is more meaningful.

One model instance is composed of a question body and the bodies of its comments, answers and answer comments. As the bodies may contain HTML code, such as tags, which are irrelevant in content and only required for presentation in the browser, tags are removed and only inner text of the tags are included in a model instance. Further conversions, such as transforming data strings of model instances into lowercase, tokenizing them into sequences, removing stopwords and transforming them into feature sequences are executed through MALLET's pipeline.

MALLET requires several parameters that impact modeling results. Based on our experiments and the official tool documentation, we set the number of topics to $K = 200$. This produces a fine-grained topic result. The number of sampling iterations was set to 2000. Hyperparameters α and β were set based on default heuristic $\alpha = 50/K$ and $\beta = 0.01$ [49]. Optimization of them was performed after all 10 iterations. Random-seed was not explicitly configured and thus its default value, the timestamp was used. Thus, our topic model estimation is not reproducible.

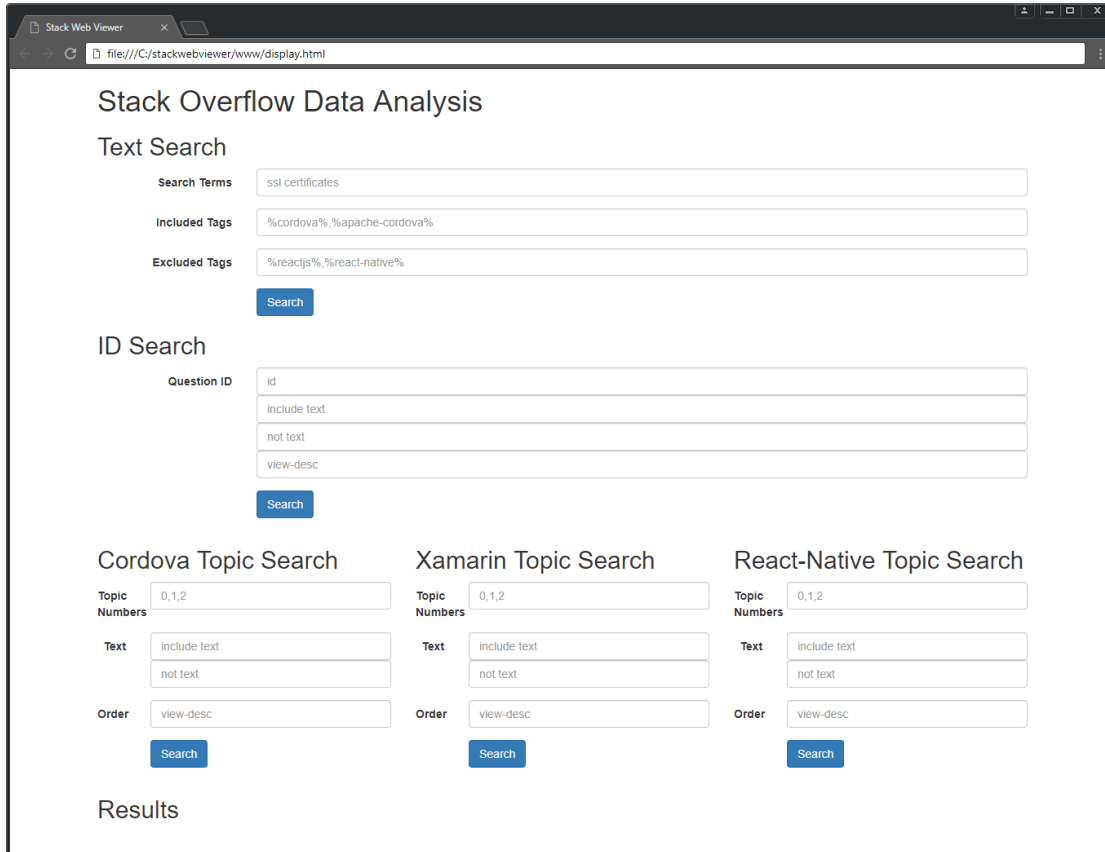
Meta Topics. As a result of topic modeling we got a list of 200 topics with its 10 top words for each framework. Since it resulted in fine-grained topics, we investigated them manually and labeled them into 12 categories, so-called meta topics. A detailed description about the meta topics is provided in Section 5.2.

Stack Web Viewer

Searching, filtering, reading and reviewing the textual contents of the posts is an important step in our analysis. The data from Stack Exchange API include HTML tags and are

designed for presentation on the web. Therefore we implemented a web tool called *Stack Web Viewer* to facilitate manual analysis.

The backend service of Stack Web Viewer is a Node.js application. It uses PostgreSQL Client `node-postgres` as well as `express` and `bodyparser` for a webservice. The Web UI of the viewer uses `handlebars` as template engine, `bootstrap` and `jquery`.



The screenshot shows the Stack Web Viewer UI with the following sections:

- Stack Overflow Data Analysis**
 - Text Search**
 - Search Terms:
 - Included Tags:
 - Excluded Tags:
 -
 - ID Search**
 - Question ID:
 -
 -
 -
 -
 - Cordova Topic Search**
 - Topic Numbers:
 - Text:
 -
 - Order:
 -
 - Xamarin Topic Search**
 - Topic Numbers:
 - Text:
 -
 - Order:
 -
 - React-Native Topic Search**
 - Topic Numbers:
 - Text:
 -
 - Order:
 -
- Results**

Figure 4.1: Stack Web Viewer UI - Search Input Screen

As shown in Figure 4.1 Stack Web Viewer provides three different ways to display posts. The first one provides full text search with optional definition of tags that must be included or excluded in the results. In the second case questions can be searched by id(s) with a comma separated id list. Moreover phrases can be entered that must be included or excluded in the results. Ordering of search results can also be changed. Available options are *view-desc*, which means the number of views of the question, *score-desc* sorts by the question's score, *create-desc*, *create-asc* means the creation date of the question or *activity-desc* which sorts by the date of the last activity. Default order is descending per question id. The third way to display posts is by topic number of Cordova topics, Xamarin topics or React Native topics based on the results of topic modeling. Display order can be changed in the same way as in case of search by question id.

4. METHODOLOGY

After clicking on the search button the UI sends an HTTP request to the webservice to get all questions with their comments, answers and answer comments that satisfy search input. Figure 4.2 shows the result section of our Stack Web Viewer. A summary about the number of found questions, the number of questions with answers available as well as how many questions are unanswered or answered with or without a selected answer is displayed. Each found *question block* consists of meta data, such as the question id, tags, creation date, last activity date, score, views, owner, answered and link to the original post on Stack Overflow. The question itself with title, body, comments, answers and answer comments is included in a bootstrap panel such that details can be opened or closed. Moreover paging is implemented to increase performance of data requests and rendering. Default page size includes 10 question blocks at once.

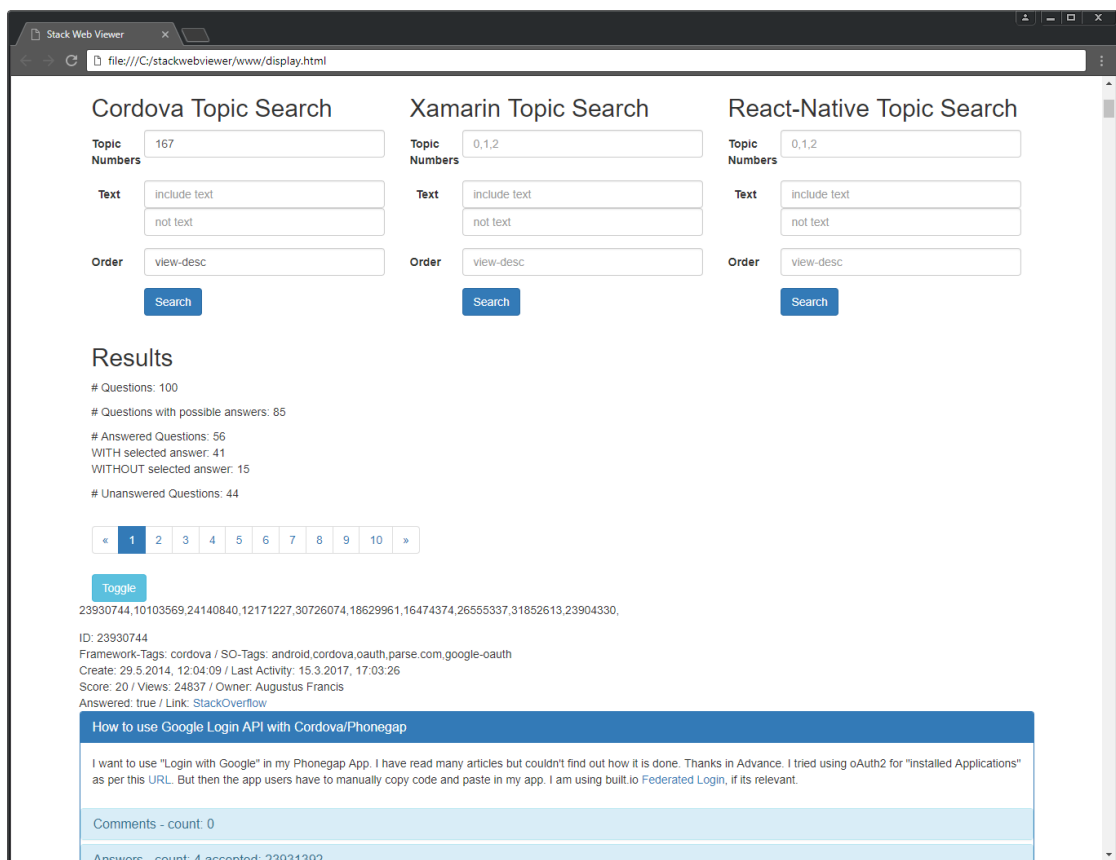


Figure 4.2: Stack Web Viewer UI - Result Screen

The webservice is an Express 4.x application that provides HTTP GET methods `/posts` and `/posts:ids` that can include several query parameters based on the above described search features, e.g. *keys* for full text search, *includedtags* or *excludedtags* to define tags of the searched questions and so on. Full text search is enabled by the built-in

mechanisms for text search of PostgreSQL⁴. Therefore the searchable original document must be reduced to the data type `tsvector`. A `tsvector` value is a sorted list of distinct and normalized words to meet different variants of the same word. Sorting and eliminating duplicates is done automatically during input. Normalization is only executed within functions as `to_tsvector` [56]. Various configuration options for normalization are available. Defining stop words, that are not included in the `tsvector` value, e.g. “a”, “and”, “the”, ignoring space symbols, removing plural forms, e.g. parse “hats” to “hat” and so on. Search terms are formatted as `tsquery` value, which consists of lexemes to search for, that can also be combined with Boolean operators, e.g. `&` (AND), `|` (OR) and `!` (NOT) and the phrase search operator, e.g. `< - >` (FOLLOWED BY). The function `to_tsquery` converts normal strings into `tsquery` format and does normalization [56]. As search is performed on the same table fields each time, `tsvector` values are created in advance and stored in own columns of the `json_questions` table. Moreover, we indexed the vector columns to speed up search.

Manual Content Analysis

The meta topic *Auth* consists of several keywords about authentication and authorization listed in Table 4.5. The id for each topic in the table is composed of the first letter of the framework and a consecutive number. C1, X1, R1 and R3 has more general words about authentication, such as “user”, “login”, “username”, “password”. More specific keywords are “oauth”, “token”, “code”.

ID	Keywords (Top 5)
C1	user, login, token, password, authentication
C2	facebook, login, app, plugin, sdk
C3	<i>oauth, token, url, code, google</i>
X1	user, password, login, username, email
X2	<i>facebook, token, login, oauth, user</i>
R1	user, login, profile, home, screen
R2	<i>token, user, auth, app, parse</i>
R3	password, login, username, email, this.state.password
R4	facebook, error, login, react-native-fbsdk, const

Table 4.5: Authentication Topics

Authorization. Access control is required to prevent unauthorized use of a resource and to prevent use of a resource in an unauthorized manner [60]. If access is approved or disapproved is verified based on access control policies.

⁴<https://www.postgresql.org/docs/9.6/static/textsearch-intro.html>

Authentication. The process to verify the identity of a person or process to be valid means authentication. It prevents impersonation for example and ensures authenticity. A user can authenticate herself in the following ways. A *knowledge factor* for example is something a user knows, such as a password, a PIN or a an answer to a certain question. Something the user has is an *ownership factor*, such as an ID card, cellphone, hardware token, etc. The third way is called *inherence factor* and means something the user is or does, such as his fingerprint, voice, face or another biometric identifiers [60].

OAuth 2.0. OAuth 2.0 is an authorization framework that defines access control for a third-party application to an HTTP service [29].

Figure 4.3 shows the principles of OAuth 2.0 in mobile apps. The mobile application is the *client* that wants to access a resource from the *resource server*. Therefore the app requires an access token from the *authorization server*. Depending on service architecture, authorization and resource server may be one or separated endpoints.

Before receiving an access token, the app has to be authorized by the *resource owner*, e.g. the user. Therefore the app initiates authorization by a request to the authorization server including its *client ID* and *redirection URI* via a so-called *user-agent* on the mobile device. Then the user authenticates himself against the authorization server with his credentials via the agent and grants permissions to the app. At the end, the server redirects the user-agent to the redirection URI including the server response [see step (1) in Figure 4.3].

Based on the used grant flow, the server response at the end of authorization differs. In case of an *implicit grant*, the server immediately returns an access token and step (2) in Figure 4.3 is omitted as the app can continue with step (3). In case of *authorization code grant*, the authorization server returns a code that has to be used in an additional request to the authorization server to obtain an access token [see step (2) in Figure 4.3].

In the last step, the app requests a resource from the resource server including the access token. Based on the token's validity, resource server returns the resource or denies access [see step (3) in Figure 4.3].

OpenID Connect 1.0. OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It allows apps to verify the identity of the End-User as well as to obtain basic profile information [46].

Martin et al. [36] and OWASP [41] indicate that secure authentication and authorization to protect app and user data are two important security requirements that should be prioritized in each data processing application. Official documentations of the frameworks don't provide much information about these important security topics. Thus, we investigated the usage of OAuth 2.0 for API authorization and authentication more in detail. Therefore we selected the topics C3, X2 and R2 and analyzed content of the top 100 question blocks of each topic using Stack Web Viewer. One question block means a

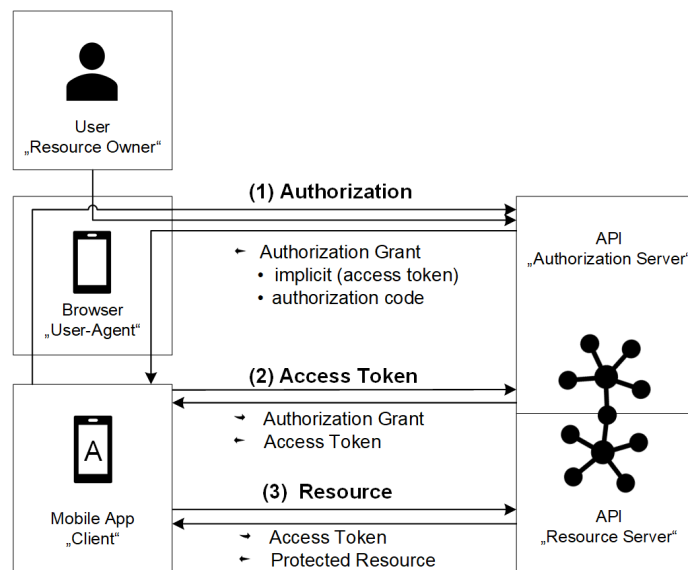


Figure 4.3: OAuth 2.0 Protocol in Mobile Apps (cf. [29])

question and all its comments, answers and answer comments. Results of the content analysis are presented in Section 5.3.

4.3 User Study

Our analysis of Stack Overflow posts allow various conclusions, such as the most popular tools, implementation trends and obvious problems as they are posted to ask for help. However it also reaches its limits as the development process itself can not be investigated and reasons for the developers' decisions remain unexplored. A user study could counteract these limitations.

Our first approach was to conduct a lab study with mobile developers close to a scenario as realistic as possible. One possible scenario would be the participant should imagine he is working in a software developing enterprise that provides web and mobile developing services and his team is developing a meta social media app that publishes one post automatically on multiple social media platforms at once. Therefore an API integration with the Facebook API, the Twitter API, the Google+ API etc. has to be implemented.

In preparation to this study we implemented a Cordova app for Android with Facebook and Google login ourselves using the providers' SDKs. Apart from the steps and time it took to setup the developing environment such as installing Android SDK and all its dependencies and getting a sample app running, implementing the login with the provider SDKs is quite straight forward to get it working. Providers indicate in their documentation using the SDKs is a secure way for authentication. However it is hard for a developer to verify this statement as detailed information is missing and therefore the

provider has to be trusted.

The test implementation showed that for a lab study we would need participants with CPF experience, but not only in coding, but also knowing how to compile, debug or run an app on an emulator or a device with the framework. There are many side tasks in real developing, which requires much many effort and knowledge from the participants, but brings no added value for the study results itself. As the main part to be examined is how and why developers' make certain decisions during implementation. Thus, we arrived at the conclusion that an online survey with questions about developers' experience of OAuth or OpenID Connect implementations would lead to nearly the same results.

4.3.1 Online Survey

Based on our findings, we conducted an online survey in the third step to get personal experience and judgment of developers about secure implementation of OAuth and OpenID Connect. Thus, we composed a questionnaire using the online tool `www.soscisurvey.de`, which is popular and free for academia.

Target Groups. Mobile developers with experience in Cordova, Xamarin or React Native that have already implemented OAuth or OpenID integration in their cross-platform app are our main target group. These are quite a lot of specific factors which would be hard to meet for a wide crowd. Thus, we also put the implementation questions to developers with OAuth experience in other applications. On the other side we are also interested in first feelings and opinions of developers that are not familiar with OAuth and therefore the survey also includes questions of type "what would you do".

Composition. The questionnaire includes several pages as can be seen in Appendix A. First question ensures that only software engineers are asked about the subsequent developing questions. Then software developer should rate their skills in some selected engineering fields, such as mobile or web development, IT security, the three frameworks and so on. Moreover they were asked if they know OAuth and OpenID Connect and if they have ever developed a mobile app with one of the three frameworks. In a next stage, basics about OAuth and OpenID Connect as well as their differences are explained to bring the interviewees on the same level of knowledge at least about the basics. Additionally the participants are asked if they have ever developed an app with OAuth or OpenID Connect integration. Based on their developing experience, the following questions are about their implementations or in case of no previous experience they are asked so called "what would you do" questions.

In case of experience with OAuth or OpenID Connect we ask some implementation details, such as which providers they included. Based on the entered providers we ask about the concept and tools they used, which grant flow they implemented, which user-agent or redirect-uri they used and if they implemented Proof Key for Code Exchange (PKCE). Additionally for each of these questions we asked why they did it in this or another way.

Concerning possible answers we always provide evasive options such as “I don’t know, the tool did that for me.” or “I can’t remember”.

The “what would you do” questions cover a ranking of preferred information resources, which concept, user-agent and redirect-uri they would use. As not experienced developers may not know these topic specific terms they are shortly explained.

At least demographics such as gender, age and country of the interviewee are asked.

Execution. The survey was open for 22 days from November 21st, 2017 to December 13th, 2017 and was advertised and shared via Twitter, Facebook, LinkedIn, email, chat rooms of computer science groups of university and within developer groups on Facebook.

Results

In this section we present our results on cross-platform mobile apps from the developer's perspective.

First, we analyze the official documentations and resources provided by the framework developers, which discloses similarities and differences how the frameworks handle security and privacy.

Then, we describe the major topics of developer's challenges based on the analyzed Stack Overflow posts in Section 5.2. Furthermore, we present results of our extensive manual analysis on authentication posts in concern of OAuth and OpenID Connect.

Last but not least, we evaluate our small user study about implementation experiences on OAuth and OpenID Connect across cross-platform developers in Section 5.4.

5.1 Security Handling

In this chapter we evaluate how the three frameworks manage security. The websites and official documentations of the frameworks are our primary source to analyze how openly they address security. Additionally, we consider third-party extensions and plugins to improve security.

5.1.1 Cordova

The official documentation of Cordova¹ provides several sections on security and privacy issues.

¹<https://cordova.apache.org/docs>

Communication

Concerning secure the network communication, certificate pinning and self-signed certificates are discussed in official documentation. Android does not provide a native API to intercept SSL connections, which is required for certificate validation. Basically, certificate pinning in Android is possible with Java and JSSE, but in Cordova server connections are handled by the WebView, which is written in C++. Thus, Cordova cannot provide certificate pinning for Android. As no support implies no consistent support across multiple platforms, Cordova doesn't include certificate pinning in Cordova at all [54].

An alternative verification method is to check the public key of the server, which is called *fingerprint*. However, such a check must be executed manually, while true certificate pinning automatically verifies the expected values on each connection to the server. Third-party plugins for fingerprint checking are available. Furthermore other plugins execute true certificate pinning for some platforms. However, they assume that all network requests are done by the plugin instead of traditional XHR or AJAX requests [53].

Self-signed certificates without certificate pinning are vulnerable to man-in-the-middle attacks. Therefore, self-signed certificates are not recommended in the official documentation. However SSL errors due to certificate chain validation errors can be permitted by certain configurations, which is recommended only for testing purposes and during development. The documentation discourages turning validation off in production mode [53].

Data Storage

Cordova's documentation has a dedicated part about storage. It provides links to a more detailed explanation about Web Storage APIs and a short overview about the following APIs directly in the documentation [54].

LocalStorage is a synchronous key/value storage of strings with a simple API. WebSQL stores data in a structured database, that can be queried with standard Structured Query Language (SQL) syntax. IndexedDB combines the strengths of LocalStorage and WebSQL that enables to store JavaScript objects indexed with a key. The SQLite plugin is almost equal to WebSQL, but overcomes the storage amount limitations and provides support for Android, iOS and Windows. The File Plugin of Cordova enables storing data on the local file system for each available Cordova platform.

Table 5.1 shows a summary of these storage APIs. LocalStorage, WebSQL and IndexedDB limit the amount of storage to approximately 5 Megabyte (MB), while the plugin based solutions does not have such a limitation. Search performance of LocalStorage is worse than the performance of WebSQL and IndexedDB because of missing search indices. On the other hand LocalStorage provides support for all Cordova platforms, while WebSQL only supports Android and iOS and IndexedDB only supports Android and Windows, but with limitations.

	LocalStorage	WebSQL	IndexedDB	SQLite Plugin	File Plugin
Storage Limit	approx. 5MB	approx. 5MB	approx. 5MB	none	none
Performance	bad	good	good	—	
Android	✓	✓	✓	✓	✓
iOS	✓	✓	✗	✓	✓
Windows	✓	✗	✓ (limit.)	✓	✓
All Cordova Platforms	✓	✗	✗	✗	✓

Table 5.1: Cordova - Storage APIs

Details on data and database encryption are not provided at Cordova’s website. However, there are some plugins available at the package manager npm that implement encryption. The SQLite plugin can be extended with encryption by the plugin `cordova-sqlcipher-adapter`² that provides a native interface to SQLCipher. It supports Android, iOS and Windows apps. The `cordova-plugin-secure-storage`³ provides an encrypted storage for key/value pairs and is also available for Android, iOS and Windows. Another encryption plugin uses the Keychain in iOS or the Keystore in Android to store encrypted local data. However, it is only available for Android and iOS⁴. An alternative to store encrypted data or files is to encrypt them directly before putting them into the storage. JavaScript provides cryptographic libraries for that.

Privacy

Best practices on how to treat user privacy are explained in a dedicated privacy guide at Cordova’s website. Each app should provide a privacy policy that describes what kind of information is stored, how those data are used and with whom they are shared and how users can make privacy-related choices. Users should actively be asked for permission before an app collects sensitive data, e.g. by just-in-time dialog boxes. If data are shared with third parties, such as social networks or advertising companies, users should be informed and provided an option to opt-out [52]. Further information and best practices for developers are provided by external links at the privacy guide.

HTML5 Security

A link to the HTML5 Security Cheat Sheet⁵ of OWASP is provided under recommended articles at Cordova’s security guide. Web Security is an important part for Cordova’s

²<https://github.com/litehelpers/Cordova-sqlcipher-adapter>

³<https://github.com/Crypho/cordova-plugin-secure-storage>

⁴<https://www.npmjs.com/package/n1-afas-cordova-plugin-securelocalstorage>

⁵https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

security, because native and web techniques are combined.

WebViews

As described in Section 3.2.1 WebViews are a basic component of Cordova apps. However, using WebViews can be risky concerning the security of the underlying app. Cordova's security guide provides a link to a paper about attacks on Android's WebView from 2011.

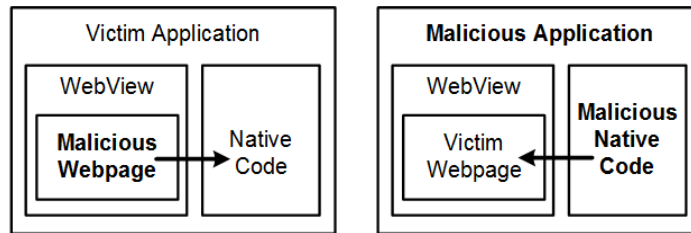


Figure 5.1: Threat Models of WebViews (cf. [35])

Luo et al. [35] define two different groups of threats as shown in Figure 5.1. In one group the app used by a victim is basically benign. However, the attacker tricks the victim to load malicious pages via the WebView into the app, e.g. by email, social networks or advertisement. Then, the loaded pages launch attacks on the victims device.

Sandbox Holes. One reason for such attacks are holes in the browser's sandbox. On Android's WebView for example, the API `addJavascriptInterface` enables JS code to invoke native code. Native interfaces, e.g. that access system resources such as camera, files or databases can be registered to the WebView and become global to the application. Therefore, each web page - no matter from which origin - can access them and defeats the browser's Same-Origin Policy (SOP).

Frame Confusion. Asynchronous function calls can also cause those attacks. When JS code, for example calls an asynchronous function of the native code through the interface, it does not wait for the results. Rather, native code invokes a JS function passing the results. Mainly, these callbacks are nothing unusual, however if a web page has iframes, the frame making the invocation may not be the same that receives the callback. Such attacks can happen from child frames, but also on main frames.

In the second group of threats the attacker owns a malicious app and wants the victims to use the app directly. The malicious app A can be a third-party app that provides special features, which extend functionality of another application B. The attacker can, for instance use Javascript Injection or Event Sniffing and Hijacking in app A to get data from app B.

Javascript Injection. Native applications can inject arbitrary JS code into web pages, that are loaded by the WebView, via the WebView API `loadURL`. It executes each string

that starts with "javascript:" within the context of the current page, provided the WebView's option `javascriptenable` is set to true, which can be done by the app easily. Thus, manipulating the DOM tree, loading malicious scripts or stealing sensitive information, such as cookies is possible.

Event Sniffing and Hijacking. Another exploitation approach are hooks, that can intercept events by registering event handlers using a customized `WebViewClient` instance. `onLoadResource`, `doUpdateVisitedHistory` or `onFormResubmission` are examples for those events, but also keystrokes, touches and clicks can be observed. Moreover, actions and content of event handlers can be modified. For example the event `shouldOverrideUrlLoading` can be misused to malicious URL redirections [35].

Whitelists

If an app needs access to external domains, they should be whitelisted to only allow access to these specific domains and subdomains. By default, access to any domain is granted, which poses risks. Inexperienced developers will not change some of those default settings, due to lack of knowledge or time. Hence, this security concept can be considered as vulnerable by default.

Cordova provides different whitelist types. One whitelist holds all URLs to which a WebView can navigate to. By default only URLs navigating to `file://` are allowed, but with the tag `<allow-navigation>` further URLs can be defined in the `config.xml` file. A whitelist for intents controls the URLs for which the app can ask the system to open, while external URLs are not allowed by default. The third whitelist type for network requests was replaced by the Content Security Policy (CSP), which can be defined within a HTML tag `<meta>` on all pages [51].

Since Cordova-Android version 4.0, a separate plugin, the so-called *cordova-plugin-whitelist* is required to configure whitelists. Cordova-iOS version 4.0 and greater and Windows Phone 8 don't require the plugin, but configuration details are the same. They are stated with the `<access>` tags in the `config.xml` according to the W3C Widget Access specification⁶.

Moreover, it is indicated that in iOS 9 a new feature acting as a whitelist, called App Transport Security (ATS) was established. The already known tags `<access>` and `<allow-navigation>` are converted to the ATS directives. A link to Apple's website for further details about ATS is provided at the whitelist guide of Cordova [55].

Iframes

An iframe has full access to the native Cordova bridge. As a result, content and especially malicious content within an iframe from a whitelisted domain, e.g. a third-party advertising network can break out of the iframe and perform malicious actions.

⁶<https://www.w3.org/TR/widgets-access/>

Therefore, iframes should not be used within the app unless the server, that hosts the content, is under the developers' control [53].

Coding Tips

The security guide provides tips for developers to create apps with high code quality. For example, it is very important to validate all user input to detect manipulated HTML and JS assets on client and server side. Websites from outside should only be opened with the `InAppBrowser`, because it uses native security features of browsers and does not provide access to the whole Cordova environment. Sensitive data, such as usernames, passwords or geolocations should not be cached to avoid unauthorized access. Furthermore the JavaScript function `eval()` should be avoided, because it can become a vulnerability for injection attacks when used incorrectly. Last but not least, developers must not assume that their source code is secure, because it can be reverse engineered [53].

5.1.2 Xamarin

The official documentation of Xamarin is very detailed. It includes guides, workbooks, recipes, samples and the complete Xamarin API reference⁷. Although there are no dedicated sections on “security” or similar and searching for “security” doesn't lead to many results, the documentation contains much information about it.

Platform Features

Information about new platform features, changes and enhancements is given for different Android and iOS versions in the corresponding sections [81, 82].

Communication

Detailed information about web services is given in the developer's guide. Different web service technologies, such as REST services, ASP.NET web services (ASMX) and Windows Communication Foundation (WCF) services are discussed [77].

The importance to use the latest version of Transport Layer Security (TLS) to secure network communications between devices is explicitly stated in documentation. Since February 2017, TLS 1.2 is used by default in Xamarin apps. Requirements as well as a step by step instructions to update to the latest version manually are described in detail. The `HttpClient` implementation can use two different options for communication. There are a *Mono* based networking stack or an API stack provided by the underlying native platform, while only the second one provides TLS 1.2 [88].

ATS. In iOS 9 (and OS X 10.11) a new security feature, called App Transport Security (ATS) was introduced. ATS enforces all connections between resources, such

⁷<https://developer.xamarin.com/guides/>

as app and backend server to be secure. Therefore any connection established with `NSURLConnection`, `CFUrl` or `NSURLSession` uses ATS by default. Apps that use `HttpWebRequest` or `WebServices` are not affected.

This means each connection must fulfill the following requirements, otherwise it fails with an exception:

- Connection ciphers: must use forward secrecy
- Cryptographic protocol: TLS 1.2 or greater
- Certificates: at least a SHA256 fingerprint with a 2048 bit (or greater) RSA key or a 256 bit (or greater) Elliptic-Curve key

Sometimes HTTPS and secure communication is not possible at all and in such a case the above described requirements can be disabled in the file `Info.plist`, which is the app's configuration file. However, the Xamarin documentation emphasizes, that Apple highly recommends using ATS by default [72].

The ATS requirements meet state-of-the-art best practices for secure online communication as described in [43]. Thus, forcing the settings for each communication already on platform-level is a good approach to increase security and establish them as default practices in mobile apps.

Authentication

The developers guide of *Xamarin.Forms* describes authentication mechanisms against web services to ensure that users only have access to their own data [74].

General descriptions about methods to implement *Basic Authentication* in Xamarin as well as security instructions are given. Basic Authentication should only be used over HTTPS connections and credentials should not be stored in an insecure format at service side. `Xamarin.Auth` provides features to store them securely. The default approach for logging a user out is to end the session [73].

Xamarin.Auth is a component available for Android and iOS that provides OAuth authentication for Google, Microsoft, Facebook and Twitter. Furthermore it includes an account store to store account information securely with `Keychain Services` in iOS and `Keystore` class in Android [75].

Authentication techniques for other web services, such as Azure Active Directory, Amazon Web Services etc. are also described.

Data Storage

The guides treat very detailed how to access data in Xamarin apps. The information for Android and iOS is quite the same at [79, 86], which means both platforms support the same storage technologies.

Moreover, the following types of storage systems on devices are explained shortly in the documentation [71, 80]:

- **Key-Value Pairs:** Android and iOS provide a built-in mechanism to store simple key-value pairs, e.g. user settings or personalized data.
- **Text Files** can be stored directly on the file-system. This can be used for example to store user input or caching downloaded content.
- **Serialized Data Files:** Objects can be saved as XML or JSON on the file-system. Serialization and de-serialization is directly supported by .NET libraries.
- **Database:** Android, iOS and Windows have SQLite database built-in, which can be used to store structured data that should be able to be queried or sorted.
- **Image Files:** Binary data are recommended to be stored directly on the file-system and not in the database. For large images, a caching strategy should also be planned to delete files from time to time to free user's storage space.

If data should be stored in a database, the cross-platform database system SQLite is recommended by Xamarin. Moreover, frameworks and libraries to access the database are described.

Loading and saving files on devices locally are also described in Xamarin's official documentation. Each platform has its own native file system, that needs to be addressed. Both, Xamarin.Android and Xamarin.iOS support the `System.IO` classes from .NET Base Class Library (BCL). Therefore common code can be used for them. Windows, however only supports `IsolatedStorage` and `Windows.Storage` APIs. As a result it requires platform-specific implementation [68].

Tips or warnings that sensitive data should be encrypted or similar information is not given in the documentation. Instructions, how to encrypt and decrypt data or files is not provided either, with the exception of some entries in API references.

Privacy

General information about privacy and how developers should deal with user data is not provided at Xamarin's websites. However, Apple made some security and privacy enhancements in iOS 10 and Xamarin explains how to deal with them for Xamarin.iOS apps. It requires to define so called `privacy Keys` in the file `Info.plist` file to get access to certain features or user data. A privacy key should explain the user why the app wants or needs this access [85].

WebViews

In Xamarin apps WebViews can be used in the same way as other UI elements. The developer guides only contain how to use WebViews, but there is no reference to security vulnerabilities about them. Xamarin.Android apps use the Android WebView and its documentation is based on Android's tutorials [64]. As Apple provides three different WebViews, their similarities and differences are explained in Xamarin's documentation and a short reference to ATS is provided too [78].

Code Obfuscation

The documentation of Xamarin.Android covers techniques to protect application code. First of all, debugging mode should be disabled in release versions. Moreover, danger of reverse engineering or code tampering is shown [84].

`Dotfuscator` is mentioned as a possible solution against these problems. It provides code obfuscation and injects runtime security state detection code at build time. The community edition can be used with Visual Studio, but it does not support Xamarin Studio [84].

Another tool is `ProGuard`, which is a Java class file shrinker, optimizer, obfuscator and pre-verifier. However, in Xamarin.Android `ProGuard` does not obfuscate the APK, it only performs the shrinking and optimization steps. Moreover, only a selection of the `ProGuard` options can be configured [83].

Moreover, Xamarin Enterprise license provides the possibility to bundle assemblies into a native binary, which keeps code safe [84].

5.1.3 React Native

React Native's official documentation⁸ is clearly arranged into basic knowledge, guides and API documentation. Thus, a beginner gets first introductions quickly, but also advanced topics and issues are described in detail. However, the descriptions and topics more focus on the framework's features to build a user interface, platform issues for Android and iOS or general development tasks like debugging, testing or releasing while sections about security or privacy are hardly found.

Communication

Different approaches how to load resources from remote URLs are described in an own networking section. React Native supports networking libraries such as *Fetch API*, *XMLHttpRequest API* or *WebSockets*. In context of *XMLHttpRequest* it's indicated that the security model is different in React Native than on web, because the concept of Cross-Origin Resource Sharing (CORS) is not available in native apps [21]. More detailed information is unfortunately not described at this point. Like the other frameworks

⁸<http://facebook.github.io/react-native/docs/getting-started.html>

Cordova and Xamarin, also React Native provides a short description about ATS in it's docs and provides a link to Apple's documentation. Moreover in context of network images it is recommended to use *https* to satisfy ATS [19].

WebViews

In React Native WebViews can be used like an UI element to load external web content, but also for communication between the native UI components and React Native. A tutorial how to integrate and combine these two worlds is given with a comprehensive example. However, concerns or a discussion about possible vulnerabilities like in Cordova's documentation about WebViews is not provided.

Platform Specific

Various platform specific issues are described in the documentation. There are even two sections *Guides (iOS)* and *Guides (Android)* which for example describe how to implement custom native modules to access the platform API. Moreover, React Native also provides a lot of cross-platform API modules, which in some cases require platform specific configuration like Android permissions in the file *AndroidManifest.xml* and iOS properties in file *Info.plist*.

Data Storage

In React Native a simple, unencrypted, asynchronous, persistent key-value storage system called *AsyncStorage* is available. According to documentation it should be used instead of *LocalStorage*. On iOS data are stored in a serialized dictionary, while on Android *RocksDB* or *SQLite* is used [17]. Further database systems or data encryption are not covered in documentation.

Development Process

Developing issues such as debugging, running the app in a simulator or how to release and sign the app are handled in documentation. The tool *ProGuard* is described to reduce the size of the Android Package Kit (APK). In context of APK signing a note is given not to store credentials in cleartext and that a certain *Keychain Access* app can be used on OSX [18].

Privacy

Although privacy is not discussed on its own in React Native's documentation, it can be discovered on some sites. For example a privacy key concerning photo libraries in iOS is described in context of React Native's camera module.

5.1.4 Comparison

Each framework provides a large documentation with guides, tutorials and specifications of APIs and plugins or modules.. We categorized the content of the official sources into security relevant topics to compare them to each other. Table 5.2 shows for each of the seven categories if they were covered in the frameworks' documentation or not.

Security Topic	Cordova	Xamarin	React Native
Platform Usage	✓	✓	✓
Data Storage	✓	✓	✓
Communication	✓	✓	✓
Code Protection	✓	✓	✓
Privacy	✓	✓	✓
Authentication	✗	✓	✗
Authorization	✗	✗	✗

Table 5.2: Security Topics in Official Documentations

All three framework documentations handle topics such as platform specific issues, data storage, communication, code protection and privacy, but they very differ in its comprehensiveness.

In the documentation of Cordova and Xamarin the available storage systems are described in much more detail than in React Native's one. However, no one provides information about data encryption. In context of secure network communication, Cordova's documentation is focused on SSL certificate validation with certificate pinning and self-signed certificates. As Xamarin provides TLS 1.2. since February 2017, they describe in detail how to integrate it in Xamarin apps. A more general description about different networking technologies and their usage in React Native is given in their documentation. One common denominator regarding secure communication is a description of the ATS approach in iOS. Cordova's official website an own guide describing best practices how to treat users' privacy is available. Although there isn't such a guide on Xamarin's or React Native's website, similar information can be found combined with explanations about device permissions.

On the other hand authentication mechanisms are only described in Xamarin's documentation, while the terms "authentication" and "authorization" are mixed up. Authorization is not explicitly covered by all three documentations.

5.2 Meta Topics

As described in Section 4.2 we investigated challenges and issues of developers from about 520,000 Stack Overflow posts about Cordova, Xamarin and React Native. In a first step we conducted topic modeling on the posts to categorize them into 200 topics

per framework. Each topic is described by 10 keywords which are characteristic for it. We manually labeled the fine-grained topics for each framework and grouped them into 12 cross-platform meta topics. In Table 5.3 the meta topics are listed with their keywords and Table 5.4 shows for each framework how many topics its meta topic consists of.

Meta Topic	Keywords
Auth	account, auth, login, oauth, session, token
Browser	browser, webview, childbrowser, iframe, inappbrowser, uiwebview
Build, Release	version, update, platform, apk, sign, gradle, bundle, compile
Communication	ajax, cors, fetch, http, network, request, response, service
Device	bluetooth, camera, nfc, barcode
Development	studio, editor, tools, plugin, package, install, npm, nuget, git
File, Media	image, audio, video, canvas, pdf, file, load, download, upload
Messaging	email, contacts, notification, sms, message
Platform	android, ios, windows, permission, intent, fragment, key, info.plist
Storage	database, storage, localstorage, asyncstorage, sql, sqlite
User Interface	css, style, layout, view, item, list, tab, menu, touch, orientation
Others	i'am, i've, it's, can't, don't, app, problem, solution, issue, found, work, error, code, failed, public, void, override, class

Table 5.3: Meta Topics

Meta Topic	Cordova	Xamarin	React Native
Auth	3	2	4
Browser	5	2	2
Build, Release	12	12	9
Communication	12	4	8
Device	4	3	2
Development	10	13	9
File, Media	8	7	5
Messaging	4	3	4
Platform	6	10	4
Storage	2	2	2
User Interface	30	40	37
Others	104	102	114

Table 5.4: Number of Topics across Frameworks

In most of the topics the top 10 words are very meaningful and allow several conclusions. In the following we describe subjects of the meta topics and describe our findings about content similarities and differences across the frameworks.

Auth. Major keywords across all three frameworks are *user*, *facebook* and *login* in meta topic *Auth*. So to say, it is primarily about authentication. Moreover, Cordova and Xamarin topics also include *oauth* and *token*. Thus, it is also about authorization. Results of our detailed content analysis of this meta topic is described in Section 5.3.

Browser. Browser topics are quite similar in all three frameworks. Their keywords are about webviews, urls, links, HTML and JS. However, Cordova topics also include more specific words, such as *inappbrowser* and *childbrowser*, which mean Cordova plugins for in-app browsers and posts how to open links and URLs with the plugins.

Communication. Network communication topics are different across the frameworks. We identified 12 communication topics for Cordova, 8 for React Native and only 4 for Xamarin. Cordova's topics are about AJAX and XMLHttpRequest as well as whitelists and CORS while React Native posts are about fetch API, OkHttp client and self-signed SSL certificates. Both frameworks have topics about WebSockets and WebRTC. Xamarin's topics include more general keywords such as web, server, connection, but also Azure services and WCF.

Storage. The topics on storage represent different storage systems across the frameworks. While window.localStorage is dominant in Cordova posts, SQLite is discussed in Xamarin posts and AsyncStorage in React Native.

Device, File, Media, Messaging. Hardware devices and features such as camera, Bluetooth and NFC are own topics in all three frameworks. Interestingly, scanning barcodes with camera is discussed across the frameworks too. Also issues about files (e.g. images, PDFs), such as loading, down- or uploading them, playing videos or sounds or sending messages such as emails or sms are cross-platform topics.

Platform. Android and iOS are the most dominant mobile platforms across the frameworks according to platform specific topics. Intents and permissions are top keywords belonging to Android applications. Keys and plist, which stands for property list files are own topics about iOS. Topics with keywords about Windows and UWP are also identified in Cordova and Xamarin posts, but not in React Native. They have supported building apps for UWP only since April 2016. Maybe there are too few posts to influence the topic models because the dataset includes only posts until June 2017.

User Interface. Some examples of layout and style keywords are *width*, *height*, *size*, *orientation*, *portrait*, *landscape*. Typical HTML tags such as *div*, *class*, *img* are found in Cordova posts. CSS keywords exist in Cordova and React Native posts. Moreover React Native UI components such as ScrollView, ListView, Slider, Toolbar have own topics. Also topics about Xamarin specific UI components and widgets such as ListView, Spinner, Picker, Calendar or EditText exist.

Development. Trends about development tools can be detected from this meta topic. Based on the programming languages of the frameworks, Cordova and React Native use the JavaScript package manager *npm* while Xamarin uses the package manager of Microsoft development platform *NuGet*. For debugging it seems that using Chrome as emulator is popular in Cordova and React Native. In contrast, Xamarin developers use emulators of the IDEs Xcode or Xamarin Studio. Moreover debugging GTK is also on an own topic in Xamarin. Also different testing tools are discussed across the frameworks. In Cordova posts the automated testing tool *Appium* occurs, while in React Native posts Facebook's testing tool *Jest* and *Mocha* are discussed. In Xamarin the .NET unit test tool *NUnit* is popular.

Build, Release. In each framework between 9 and 12 topics deal with compiling, build and release issues. Signing Android APKs with the Android KeyStore is included in each framework. Moreover, framework typical build tools can be detected from the topic keywords such as Gradle in Cordova, AOT and JIT compiler in Xamarin and Babel in React Native.

Others. Topics with general words such as *it's*, *don't*, *work*, *problem*, *errors*, *question*, *answer* are not meaningful enough and therefore put into a generic meta topic called *Others*. It also includes topics that are not comparable across the frameworks.

5.3 Posts about Authentication & Authorization

In the second phase of challenge analysis we focused on meta topic *Auth* and analyzed 100 question blocks per framework as described in Section 4.2.3. Therefore, we selected the Cordova topic *C3* including the keywords *oauth*, *token*, *url*, *code*, *google*, Xamarin topic *X2* described by *facebook*, *token*, *login*, *oauth*, *user* and React Native topic *R2* dealing about *token*, *user*, *auth*, *app*, *parse*.

A question block consists of a question with all its comments, answers and answer comments. Major subjects of the selected topics are about authentication and authorization issues, API access with OAuth 2.0 and login or session management using OpenID Connect or other authentication libraries. Various aspects, such as trends concerning tools, libraries and plugins used for OAuth 2.0 and OpenID Connect as well as main problem areas can be identified from recurrent questions and error messages that are posted. Basics about OAuth 2.0 and OpenID Connect are provided in Table 4.2.3.

5.3.1 Statistics

A first overview about the 100 question blocks of each framework topic is provided by the following statistical observations.

The distribution of framework question blocks between January 2012 and June 2017 is shown in Figure 5.2. The number of Cordova questions is equally distributed with about

20 questions in 2013, 2014 and 2016. Most of the questions are from 2015. Xamarin questions have increased continuously from 2013 to 2016, while questions about React Native have increased four times from 2015 to 2016. The numbers from 2017 cannot be compared to the previous years, because they include only questions for the first 6 months. However, it can be assumed, that the number of questions about Cordova decreases this year, while questions about React Native and Xamarin seem to get more. The novelty of React Native could be one reason for that. It was first released at the beginning of 2015, while Cordova has existed since 2012. The chart shows first releases, since first authentication questions appeared nearly at the same time. This fact applies to all three frameworks.

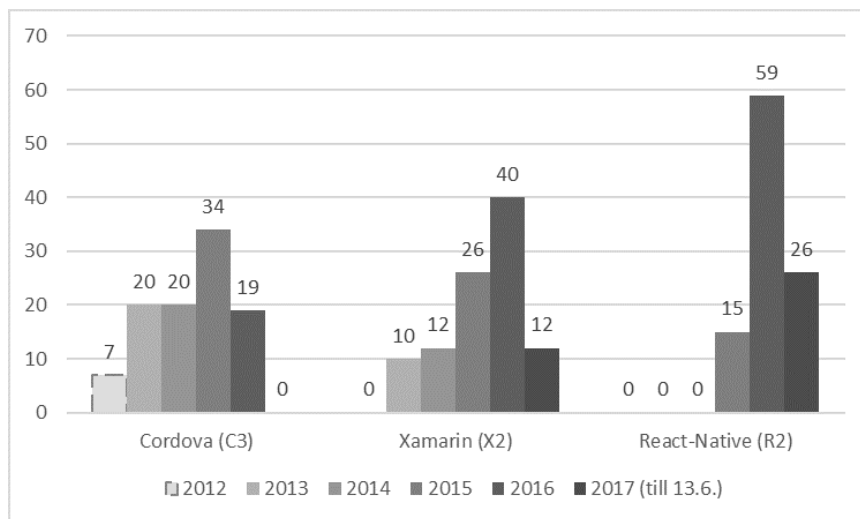


Figure 5.2: Number of Auth-Questions per Year and Framework

At first glance it seems obvious that popularity of a question block can be defined by the number of its views. However, this hides a pitfall as the following example shows. Our dataset has more questions with high view counts about Cordova than about React Native. Thus, we conclude that Cordova questions are more popular than React Native questions. However, we did not regard the factor of lifetime. Therefore questions for a problem in React Native may have to be created first with zero view counts, while similar questions about Cordova may exist and users can search and view the existing ones, which increases their view counter. Therefore we additionally have to consider the creation date of a question when talking about its popularity.

Figure 5.3 shows the distribution of view counts across creation date of question blocks for each framework topic. Four question blocks in C3 and R2 and 3 question blocks in X2 have more than 5000 views, but all of them have been created before March 2016. In each topic, the most often viewed block represents an outlier with more than 20,000 views created between 2014 and 2015. Each question block has been viewed more than 100 times. Two-thirds of React Native's topic, half of Xamarin's topic and only 37% of

Cordova's topic have been viewed less or equal than 400 times. Hence, we argue that the number of views correlates with the creation date.

5.3.2 Post Types

Several reasons may impact whether a developer posts a question. Rosen and Shihab [44] categorized questions on Stack Overflow into *what*, *how* and *why*, which is quite similar to our categories, but in comparison, we consider question blocks containing questions, answers and comments instead of single questions.

Best Practices. Developers ask for approaches, concepts, tools, technologies to use for a certain issue. They may not know the available possibilities and therefore they ask the community about their experiences which concepts are the best. Sometimes these questions cause discussions about pros and cons of approaches and tools among more experienced developers. Often such question block results in a collection of different approaches and suggestions the developer can use or try for his implementation. Rosen and Shihab [44] call them *what* questions, which are more abstract and conceptual and to ask for help in making a decision.

How To. Another major type are how to question blocks. In principal they are more specific to a certain issue about a tool, framework, library or technology in contrast to best practice questions, which are more conceptual. But also best practices about the tools can be asked here. Rosen and Shihab [44] describe *how* questions are used to ask for instructions. Many questioner in topics C3, R2 and X2 mentioned being a newbie in the subject and requiring help from more advanced developers. Answers and comments of these questions sometimes extend official documentation as required information is missing there. In other cases official documentation gets replaced by the posts as questioners do not read documentation before, they ask the community first.

What is Wrong - How to Solve. In contrast to the above types, in this category developers have coded something and got errors where they need help to solve. In some cases they don't know what the problems are and so they post error messages, stack traces and screenshots. Rosen and Shihab [44] call them *why* questions as developers ask for possible reasons of their errors. We also identified question blocks where developers know why the error occurs, but they have no idea how to solve it. So they ask the crowd for their experience.

Self-Answered. Several questions in topics C3, R2, X2 are self-answered by the questioners. Nobody else suggested a possible or usable answer and in the meantime they found a solution by themselves. The questioners posted their solution to share gained knowledge with the community, such that others can profit.

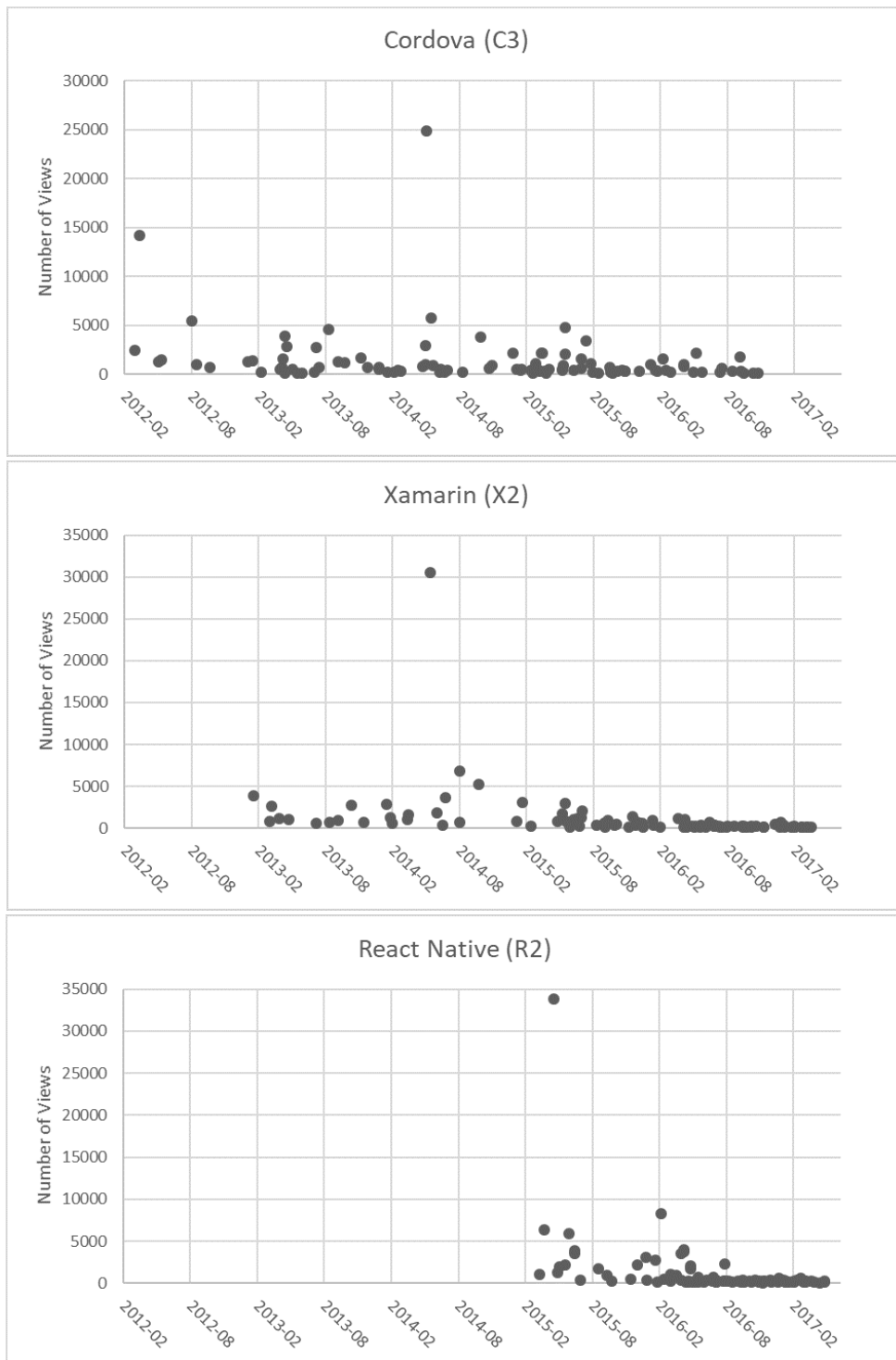


Figure 5.3: Auth-Question View Counts by Creation Date

Unsustainable. In contrast to self-answering questions, we also identified question blocks that are marked as answered, but it is unclear which solution really worked. So to say, such a question block is useless for other developers as they can't retrace the working solution. At first it is nice to see somewhere else had same problems and solved it, but it is frustrating not to find out how.

Security Relevant. There are rather few questions where developers directly ask if this approach or solution is secure. For example some questions are about storing credentials and tokens securely, which result in long discussions. Other security relevant posts are comments or answers where writers warn about security issues and try to create awareness. For example they warn that secret data in binaries of a mobile app is insecure as they can be extracted easily.

5.3.3 Libraries, Plugins, Services and APIs

Various libraries, plugins or services to implement authentication and authorization exist for each framework and the most common ones from our datasets are described in this section.

Cordova (C3)

Many provided code snippets, custom implementations and suggested plugins in the C3 question blocks require Cordova's plugin `Childbrowser` or it's successor plugin `InAppBrowser`. The basic approach of the snippets is quite the same, as they open the consent page of the identity provider in a new embedded browser tab alias web view and retrieve the access token from the redirect URI in the `loadstart` event of the embedded browser. This means OAuth flow is directly executed from client application. Therefore data such as `client_id` and `client_secret` has to be put into the client application, which can be easily extracted and misused by attackers. As a result, web view approach is insecure by design as stated in [10].

The plugin `ng-cordova-oauth` is an AngularJS Apache Cordova OAuth library. It requires the Cordova `InAppBrowser` and Cordova `Whitelist` plugin, because it implements the above described web view approach. We found 16 question blocks where developers either were faced with some errors based on the plugin or other developers recommended the plugin on best practice questions. The plugin supports a huge number of different identity providers, however it does not support Google anymore, because Google has been blocking authentication requests from web views since October 2016.

The plugin `cordova-plugin-googleplus` handles OAuth2.0 authentication against Google provider by using Google Sign-In API⁹ instead of an embedded browser, as the other described plugins do. Since Google no longer allows OAuth requests from embedded browser, this plugin is recommended for Google authentication in newer comments and answers.

⁹<https://developers.google.com/identity/>

Xamarin (X2)

In Xamarin's official documentation libraries such as *Xamarin.Auth*, *Xamarin.Social*, *Active Directory Authentication Library (ADAL)*, *Microsoft Authentication Library (MSAL)* and *Azure Mobile Client SDK* are described and recommended. As a result, we frequently identified these libraries in topic X2 posts too.

Xamarin.Auth is an official component of Xamarin for Android and iOS that handles authentication, as already mentioned in Section 5.1.2. 50% of X2 question blocks deal with it. In 78% of these question blocks users had a specific question about the component, were faced with an implementation problem and/or needed technical explanations additionally to the documentation. The other 22% question blocks are best practice questions of beginners where other developers recommended to use *Xamarin.Auth* or developers described how they used *Xamarin.Auth* for a similar problem.

11% of X2 question blocks deal with authentication and authorization against identity provider Azure AD using *Active Directory Authentication Library (ADAL)*. Questions about ADAL are quite new, as they are from 2015 till 2017. Moreover, they are viewed only 286 times in average, excluding one outlier with 870 views.

Microsoft Authentication Library (MSAL) is a new authentication library of Microsoft for Azure AD. There are 3 question blocks using MSAL in topic X2. According to the official github repository¹⁰, it is a preview version currently, but can be used in production.

Another described approach for authentication of Xamarin is using Azure Mobile Apps as backend services, that authenticate against third party identity providers, such as Google, Facebook, Twitter or Azure AD [76]. The *Azure Mobile Client SDK* is a library to access the registered Azure Mobile App instance and execute its provided authentication methods. In topic X2 we found 6 question blocks that had problems with their custom implementation or configuration on Azure portal.

Moreover, two others concepts are discussed in topic X2 commonly. One way is to use a backend as a service provider, such as *Auth0* or *Okta* that acts as client and handles OAuth requests. Another way is to use official mobile SDKs from OAuth providers, such as *Facebook SDK* or *OneDrive SDK*.

React Native (R2)

Numerous authentication plugins and libraries are referenced in posts of topic R2 only once. In contrast to this, there are services such as *Firebase Authentication* and *Auth0* that occur in multiple posts. Some questions are based on platform *Parse*. As this platform was shut-down in January 2017, we don't consider it further.

¹⁰<https://github.com/AzureAD/microsoft-authentication-library-for-dotnet>

Firestore¹¹ is a platform backed by Google that provides backend APIs and services for mobile apps. Firebase Authentication is one of these services to easily implement secure authentication systems according to its official website. As we found 28 question blocks in topic R2 dealing with Firebase, it can be assumed that the platform is commonly integrated in React Native apps. 82% of 28 question blocks are tagged with “firebase” and/or has it in the question title. Moreover, we found two answers where Firebase was recommended to use for authentication.

Auth0¹² is a third-party service that provides APIs for developers to easily implement and integrate authentication and identity management in their own applications. Various authentication protocols, such as OAuth 2.0, OpenID Connect, SAML, WS-Federation and LDAP are supported by Auth0. In our analysis we found four questions where developers claim using Auth0 and getting some errors or developers have a how-to question about Auth0. Two other times Auth0 was recommended in best practice questions.

We identified many custom implementations in topic R2 sending requests with Fetch API. In 21 question blocks Fetch API is mentioned, explained or used in code snippets. Fetch API is a Web API to fetch resources across the network, similar to XMLHttpRequest API, while Fetch API is newer and uses promises instead of callbacks.

5.3.4 OAuth Providers and Protocols

An OAuth provider offers endpoints for requesting access to certain resources on the provider’s site. According to our dataset, mostly the same providers are integrated across all three frameworks. We identified Facebook, Google, Twitter, LinkedIn as the most common ones. Other providers such as Instagram, Azure AD or Spotify are used too.

Many question blocks in topics C3, X2 and R2 include code snippets due to several reasons. Developers that need help to find mistakes in their implementation add code to questions, such that more advanced developers can take a closer look to it. On the other hand, users that provide answers add code snippets to their explanations to better describe their solution.

As described Xamarin.Auth is a very popular plugin for authentication in Xamarin apps and in 90% of Xamarin.Auth question blocks code snippets are included. The distribution of OAuth protocols in these snippets is given in Figure 5.4. The snippets are considered per post type, while in one group all snippets of questions are aggregated and the other group contains snippets in answers and comments. Generally, there are more code snippets in questions than in answers and comments. More than half of the snippets include protocol OAuth 2.0. There are five question snippets with OAuth 1.0, but only one snippet in an answer or comment. Snippets with implementations for both protocols are rare. Only one snippet is in a question and two in an answer or comment. In the 10 posts of group *Others*, users talked about Xamarin.Auth but provided solutions

¹¹<https://firebase.google.com/>

¹²<https://auth0.com/>

with different approaches, as for example self-implementations using `httpClient()` or extending `Xamarin.Auth` authenticator class `OAuthXAuthenticator`.

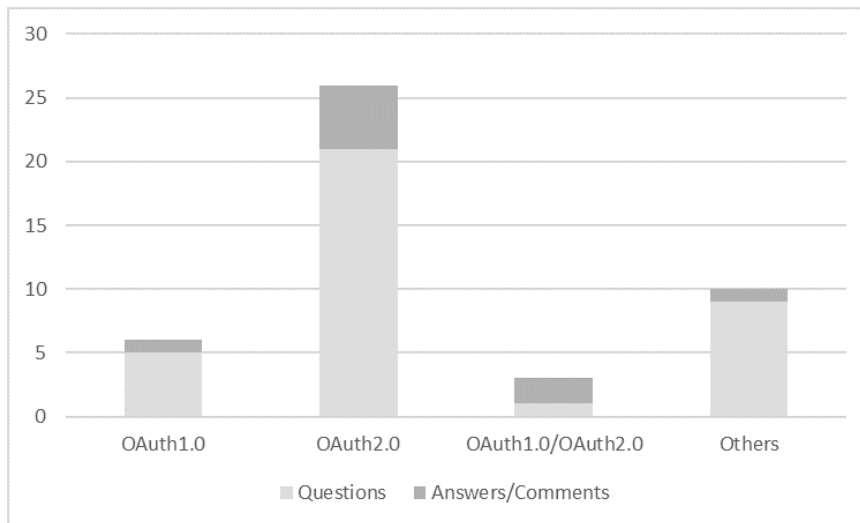


Figure 5.4: Distribution of OAuth Protocols in Xamarin.Auth Code Snippets

In addition, the OAuth protocol indicates the underlying OAuth provider in some cases. OAuth 1.0 posts mostly deal with Twitter API, while major providers in OAuth 2.0 posts are Facebook and Google.

5.3.5 Popular Question Blocks

In each of the selected framework topics popular question blocks with several thousand views exist. Considering their post types the following best practice and how to questions are heading the list.

Cordova (C3)

The most popular question block of topic C3 with more than 24,800 views is a general question how to implement Google API login with PhoneGap created in May 2014. One answer provides a code snippet based on the web view approach with authorization code grant flow described in Section 5.3.3. This answer has a score of 28 and we found the same snippet in answers of other questions too. As [23] showed there is a correlation between view counts and scoring of posts to the actual copy and paste rate, we assume the snippet is included in many released Cordova apps.

One commenter of the answer indicates that he researched a lot for the topic and rates this solution as simple and the best. However, some comments earlier another user warns about putting the secret in the app. Moreover, we detected a comment in another *how to* question block, that the web view approach is the best. Therefore we can follow the

observations of [1, 23], that developers prefer functional solutions over secure ones and often ignore existing security warnings in comments.

Although the question block was created in May 2014, it was regularly commented and updated. As one comment from 2017 describes that the web view approach does no longer work for Google API, because Google completely blocks OAuth requests from embedded browsers from April 2017 on. Therefore using plugin `cordova-plugin-googleplus` in this case is recommended, as described in Section 5.3.3.

Another popular question block with about 5700 times is also a *how to* question about Google OAuth 2.0, but in combination with AngularJS. Since nobody provided an answer to the question, the asking developer published his custom solution as help for others.

Xamarin (X2)

The most often viewed X2 question block is a *best practice* question how to login to Facebook with Xamarin.Forms. Some of the answers advise to use Xamarin.Auth as in many other posts of topic X2 too. A detailed description of Xamarin.Auth with extra code samples that are also available on Github is provided. Another provided solution that was posted is also available on Github but uses Facebook SDK instead of the web view approach. However, it seems that this solution is outdated for Android since April 2017, because Facebook changed its Android SDK.

The issue that Google does not allow web view implementations anymore also occurs in Xamarin posts. A developer asks how to implement Google OAuth with Azure Mobile App backend. There are several answers providing valuable information such as Azure Mobile Apps use Xamarin.Auth beyond and therefore they will support Google OAuth as soon as Xamarin.Auth supports it. Additionally a workaround is described in the same answer from April 2017 to use native provider SDK to get a token, send it to the mobile backend and swap it into a ZUMO token. For further information a link¹³ to a self-written book about Xamarin and Microsoft Azure Apps is given. We found links to this book in other posts too.

In another answer from May 2017 a user corrects that Xamarin.Auth supports native UI, such as Android CustomTabs and iOS SafariViewController since version 1.4, but not Xamarin.Forms which is wanted by some users, but version 1.5 will support it. Release of Xamarin.Auth 1.5 was in June 2017 and it really added support for Xamarin.Forms. So to say, Stack Overflow is used to disclose insider knowledge and inform users about new features in advance of the release.

React Native (R2)

In the most popular question block with more than 33,000 views a developer had a problem using Fetch API. However he found the answer on its own and posted the solution, that he used the method `fetch()` incorrectly.

¹³<http://aka.ms/zumobook>

In a very popular question block with about 8300 views a developer asks about the best approach to implement basic authentication for a React Native app. Various answers and approaches as for example JSON Web Token (JWT), links to tutorials using Auth0 and Firebase are provided. Moreover the importance of storing tokens securely is mentioned, for example by using Keychain on Android and KeyStore on iOS. The selected answer provides a custom implementation of HTTP Basic Authentication, hashing the token and getting access and refresh token. Requests in this approach are sent over the Fetch API.

The fourth most often viewed question block is about authenticating against a Ruby On Rails backend application. Different approaches are described, such as using authentication based on tokens or cookies with web view. Links to tutorials are also provided.

Another developer asked for the best approach to implement a login for Facebook, Google and Twitter accounts with a Node.js application in the backend. One responder provides a link to a tutorial that uses web view, node modules `express`, `passport.js` and `react-native-cookies` library and another answer includes a comprehensive description about OAuth 2.0, tokens and explains implementing implicit grant flow and resource owner password grant flow.

In another question block a developer converts her Cordova app into a React Native app, she gets confused that there are no cookies used in `react-native-google-signin` plugin as in her Cordova app. A user explains the difference that Cordova is like a web app using web view and therefore cookies are available while a React Native app is more like a native app and therefore a token based authentication is a better choice.

5.3.6 OAuth 2.0 Grant Flows

Four different types of authorization grant are defined in OAuth 2.0 specification. These are the *implicit grant*, the *authorization code grant*, the *resource owner password credentials grant* and the *client credentials grant*, while the last two are rarely used.

In the authorization code flow the client, e.g. the mobile app is authenticated against the authorization server using the public identifier called *client_id* and the secret parameter like a password called *client_secret*. If the resource owner, e.g. the user permits access to the resource, the authorization server returns an authorization code to the client. In a second step the client requests an access token from the authorization server using the above received authorization code. In contrast to this flow, the authorization server immediately returns an access token to the client after the first request including only *client_id* in the implicit grant flow [29].

According to the OAuth 2.0 specification there are two different types of clients. A confidential client is on a server where access to client credentials can be restricted and controlled while a public client is incapable of ensuring the confidentiality of the credentials. This means a mobile application that executes the grant flow from client side is a public client [29].

Cordova (C3)

The most often implemented or discussed grant flow in Cordova’s topic C3 is authorization code grant followed by the implicit grant. We investigated code snippets and questions by the parameters `response_type` of the authorization request and `grant_type` of the token request to detect the flow types as listed in Figure 5.5.

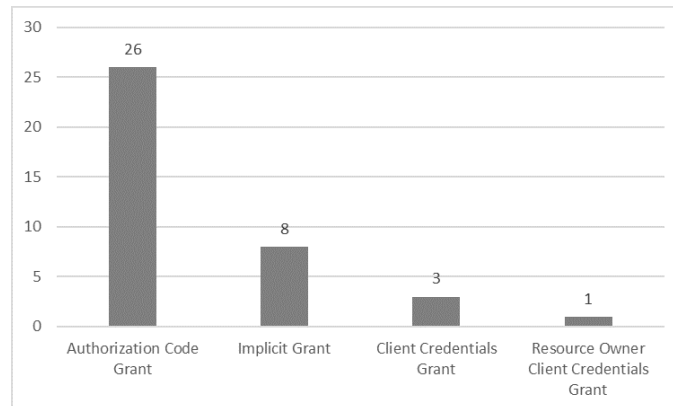


Figure 5.5: Number of OAuth Grant Types in C3

As authorization code grant requires to include the client secret in requests, we detected only 4 posts about concerns including the secret in app binaries and that it might be a security risk. One user asked if including the secret in source code of client application is a problem. A commentator only indicates that code parts using client secret should be executed on server side. In fact, it is true, but not quite helpful without more explanation.

In the most popular question block a user wrote a comment, that the secret should not be put inside the app. This has resulted in a discussion about workarounds, e.g. adding a config file or creating a system variable. However, this does not solve the issue that it can be accessed on the client side, as another user warned correctly.

The third concern we found was only a side note in an answer, that it is insecure, but without providing another solution. Paradoxically, the writer provides a custom implementation that also requires the secret in the client app.

In the fourth question block a user asked why he doesn’t get a refresh token with `ng-cordova-oauth` plugin. Therefore one responder described the implicit grant and authorization code grant and mentioned that client secret should not be exposed in the end user code.

On the other hand there are also some posts where using implicit flow is recommended in Cordova, because it doesn’t require a client secret.

Investigating the code snippets of topic C3 more in detail, we were astonished how often developers published code snippets on SO with real credentials such as `client_id` or `client_secret` as they were not replaced by placeholders. In fact, we found 12 of such

posts. As it is possible, that some of them are only development or testing credentials, the risk of high damage is lower. However, we assume that some of the published secrets are also used in production, which can cause big harm.

Xamarin (X2)

Like in Cordova, also a few Xamarin developers concerned about putting the client secret into the mobile app.

One developer posted his idea to use a nodeJS server as a proxy for sending OAuth requests to the endpoints, but he was not sure how to put the token back from the proxy to the app. A comprehensive discussion arose, where some users wanted to play down the security risks, while others persisted in their viewpoint that “OAuth sucks for mobile clients”.

In another case, a developer was not sure if Xamarin.Auth can be used securely with Facebook OAuth, because Facebook’s documentation stated that app secret and access token should never be included in client side code, such as HTML or JS, or native apps that can be decompiled. As a result, other developers explained the difference of implicit and authorization code flow. Implicit flow is designed to be used for mobile apps as they can not hold a secret. A secret must only be stored at a location where it is unreachable for third parties, such as a server. Authorization code flow uses a client secret as additional protection to identify a specific party. If a developer configures his app as a native/desktop app in Facebook Developer Console, Facebook ignores a client secret in the OAuth flow requests, because it’s confidentiality can’t be ensured. So it is described that a developer can either configure the app in the console as another kind than a native app or she can use the implicit flow. Using implicit flow in Xamarin.Auth was recommended from multiple developers and marked as the accepted answer.

React Native (R2)

In topic R2 the OAuth flow itself is not discussed so often than in topics C3 and X2. However, in R2 it is talked about all four flow types at least once.

In one case a developer first tried to use the implicit flow to implement an own OAuth server and asks if it is the right way. In the answers a difference between implicit and resource owner password grant type is explained, such that if it is a third-party service the *implicit grant* and in case of an own service the *password grant* can be used.

In a question block how to implement OAuth for Instagram, one user recommended using Auth0 and *authorization code grant* with PKCE and to define the redirection per custom URI scheme or a local web server, which corresponds to a security best practice example for OAuth in native apps.

Topic R2 posts also include concerns about putting client secrets in the mobile app. Ideas to solve the problem are similar to those posted in the other topics. A server-side component that holds the secrets and actually performs authorization requests

are suggested. In other ways, it requires a proxy server in the middle of client and authorization server, that can be secured to hold the secrets.

Security Discussion

As described a mobile app executing OAuth on client side is a public client, that can't ensure confidentiality of a secret. Authorization code grant flow requires a *client_id* and *client_secret* as credentials to authenticate the client, while implicit grant flow only requires a *client_id*. That's why implicit grant flow was the recommended flow for public clients in OAuth 2.0 specification. The same and similar description was posted in grant flow discussions in Cordova and Xamarin.

However, this flow has many drawbacks like no refresh token support, no client authentication and therefore it is vulnerable to client impersonation and other attacks as described in [10, 90].

According to the current best practices about OAuth 2.0 for native apps, which was finally released in October 2017, the recommended OAuth flow in native apps is authorization code grant flow with PKCE [12].

The concept of Proof Key for Code Exchange (PKCE) is to add an additional parameter holding a secret to authorization and access token request that is required to obtain an access token. In detail the client creates a *code_verifier* and transforms it into the so-called *code_challenge*. The code challenge and method of transformation *t_m* are sent along in authorization request, which are recorded by the authorization server. When the client requests an access token it sends authorization code and code verifier. Then the server transforms the verifier with the previous obtained method and compares the own generated value with the stored code challenge. Only if the two values match, an access token is returned. As a result, an attacker who intercepts the authorization code on the client's device can't get an access token, because he doesn't know the code verifier [45].

In investigated SO posts PKCE was not discussed at all, except in one or two React Native posts. On the other hand, third party providers like *Auth0*, *Firebase* or *Okta* describe in their documentation to support it.

5.3.7 User-Agent and Redirection

At the end of OAuth 2.0 authorization flow the authorization server redirects the user-agent of the resource owner (e.g. a browser, embedded browser, system browser) back to the client (e.g. a mobile app). As our analysis shows, defining, using and configuring this redirect endpoint correctly is a common challenge across all three frameworks. We found plenty of questions in the selected topics with error messages about redirect URI mismatches or invalid redirect URIs from authorization providers, such as Google, Facebook, LinkedIn, Twitter, Dropbox, Azure AD, Wordpress, Instagram, Spotify and so on.

Cordova (C3)

18% of questions in Cordova's topic C3 deal with the redirect URI. Commonly the mobile application has to be registered at the provider in advance to get a `client_id` and `client_secret` and to define URIs that are valid to redirect the user-agent back with authorization code or access token. Many problems concerning the redirect URI are caused by misconfiguration on the providers' console or using not registered URIs. Error messages such as *“redirect URI did not match a registered URI”*, *“client application failed validation: not a valid URL format”* from various OAuth providers such as Google, Facebook, Twitter, LinkedIn are posted.

Another reoccurring issue was to configure the right application type in Google's developer console, such that it works. Some had to change their configuration from web application to installed application and others had to change it the other way round.

In some cases developers didn't understand the differences between the URI variants and needed additional explanation from more advanced users. Sometimes helpful descriptions are provided in answers and comments, but sometimes only parts of official documentation are copied into answer posts.

In concern of using Google as OAuth provider many developers posted that their default redirect URIs according to Google console settings are `http://localhost` and `urn:ietf:wg:oauth:2.0:oob`. Many of these posts are from 2014 or 2015. Current documentation about Google OAuth in mobile and desktop apps describes four possible variants. For Android, iOS and UWP apps using a custom URI scheme for redirection is recommended. Loopback IP address such as `http://127.0.0.1` or `http://localhost` should be used if the platform can listen to a local web server, because the authorization server then returns authorization code within a query parameter to this address. In the last two variants authorization code is included in the title bar of a HTML page to which the user-agent is redirected. In case of `urn:ietf:wg:oauth:2.0:oob` as redirect URI the user has to copy and paste the code from HTML page into application manually, while with URI `urn:ietf:wg:oauth:2.0:oob:auto` the application itself has to read the title from HTML page and user may has to close it. According to official documentation this method was designed for embedded browsers like web views, but as it is less secure than the first two types, it is deprecated now [26].

Xamarin (X2)

Various question blocks facing issues with redirect URI parameter can also be found in Xamarin's topic X2.

One developer used `http://www.facebook.com/connect/login_success.html` as redirect URI to authenticate against Facebook using Xamarin.Auth and got the error message: *“One or more of the given URLs is not allowed by the App's settings. It must match the Website URL or Canvas URL, or the domain must be a subdomain of one of the App's domains.”* In official documentation of Xamarin.Auth and Facebook the URI is

described to use, but according to the error message the developer didn't configure it correctly on the provider's side. Another developer with the same problem asked why the URI is actually needed, because in Xamarin native examples it isn't required. One user explained that Xamarin.Auth uses a web view and the access token is included in the redirect URI as query parameter. Moreover, he described the required configuration on the provider's side as extra security layer, as the provider can thereby restrict redirection to only registered URIs. This question block was viewed more than 5200 times and we identified further posts where the redirect URI of Facebook was used, but not registered.

In other cases developers were not sure which is the correct redirect URI for Google or Twitter integration. A developer uses `https://www.googleapis.com/plus/v1/people/me` as redirect URL in June 2016, as it was suggested anywhere. According to the described error message the redirect URL does not match any URL registered in Google's developer console. Another user suggested `http://localhost:8000` and this answer was marked as accepted. In OAuth 1.0 the equivalent to *redirect URI* is called *Callback URI*. In June 2015 a developer tried `http://twitter.com` as callback URI in Xamarin.Auth, but the `Completed` event never fired. The developer found `http://mobile.twitter.com` as working solution himself and shared it with the community. Comments from other developers confirm that they benefit from the self-answered question. In June 2016 another user wrote a comment, that the callback URI has changed to `http://mobile.twitter.com/home`. The question block has more than 1100 views.

The issue to configure the application as web app instead of installed app in Google's developer console occurred in X2 posts several times, as in the Cordova posts.

React Native (R2)

In React Native's topic R2 discussions and questions about the correct redirect URI are more included in general questions and explanations how to implement OAuth and which OAuth grant flow or tools to use. In some of these question blocks developers purposed to authorize or authenticate against Wordpress, Instagram or Spotify. In comparison to the other topics, there are less error messages about URI mismatches in topic R2.

Security Discussion

By definition of OAuth protocol, redirect URIs have to be registered at the identity provider's side and the providers have to ensure that tokens are only sent to registered domains. Client applications, such as mobile apps do not have an own domain, no web service that can be used for redirect uri, so often `localhost` or `oob` are recommended and used. Attached listeners on `localhost` redirects or even on custom schemes in iOS is not only pretended to the original app. As a result, the token can get intercepted by malicious third party [10].

According to best practices [12] some URL schemes are recommended and some should be avoid. One recommendation regarding the custom URI scheme is to use a unique

scheme like a reverse domain name (e.g. “com.example.app:/”), while simple schemes like “myapp:/” aren’t unique enough. Another technique are app-claimed HTTPS URLs, which must be supported by the native platform. This means that when the browser get a claimed URL, it doesn’t load the page in the browser, but launches the native app and delivers the URL as input. Android and iOS currently support the app-claimed HTTP URLs. Since Android M so-called *Android App Links* exist and in iOS they are called *Universal Links*, which are available since iOS 9.

The user-agent handles redirection and also requires some security considerations. An embedded browser as user-agent like the Android WebView or iOS UIView is insecure because of various reasons. For example the principle of least privilege is violated as the mobile app has full access to the browser and can intercept user credentials. Thus, the embedded browser should be avoid as user-agent and in-app browser tabs should be used instead [12]. Examples of such in-app browser tabs are *Chrome CustomTabs* in Android and *SafariViewController* in iOS.

We found posts in all three topics C3, X2 and R2 where in-app browser tabs are recommended to be used, partly because Google doesn’t allow embedded browser requests anymore as already described.

5.3.8 Refresh Token

A refresh token is used by the client to get a new access token from authorization server without user interaction when the current access token expires or gets invalid. The authorization server issues such a refresh token to the client together with the access token for later use [29]. Similar questions and misunderstandings about the principles and usage of refresh tokens were found in all three frameworks.

Cordova (C3)

Cordova developers who use *ng-cordova-oauth* plugin asked how to get a refresh token. In one question block a user explained that the plugin implements the implicit grant flow and thus refresh tokens can’t be obtained. Another user posted a code snippet that uses authorization code grant flow, but the asking user concerned about the client secret in the app. Another solution using a proxy server that rewrites the URL and attaches the secret is given. In a second question block with the same problem the questioner herself answered that with the plugin only an access token can be obtained and for refreshing an authentication code is required. Therefore she posted a code snippet to get the code that has to be sent to the resource server. This means a solution is provided, but a deeper explanation of reasons is missing.

Xamarin (X2)

In Xamarin’s topic X2 developers were challenged to get and use a refresh token in combination with Xamarin.Auth plugin. Based on the SO posts, it seems that Xamarin.Auth

doesn't support token refresh by itself. Thus, developers implemented their own solutions, but as they got stuck on some point they asked for help. For example one developer looked for examples, tried requests with parameters like `grant_type=offline` and `approval_prompt=force` as they were found somewhere, but nothing really worked. Other developers provided links to tutorials or similar posts. The question block is marked as answered, but it is unclear which post really helped. In another question block a link to a Xamarin.Auth extension that should provide token refresh is provided. The link refers to the Github repository of Xamarin.Auth where discussions about refresh tokens can also be found in open issues.

In other question blocks developers needed additional information about refresh token in Auth0 or how to get access token and refresh token from Imgur API.

React Native (R2)

In React Native posts about refresh token or how to handle token expiration are different based on the libraries and custom flow implementations. One question block deals with a custom solution using JWT tokens and storing them in AsyncStorage. Another one is a general best practice question how to handle refresh of OAuth 2.0 tokens. More specific token refresh questions with less useful answers or help concerning Firebase or Auth0 integration were also identified.

Security Discussion

According to OAuth 2.0 specification refresh tokens must be kept confidential in transit and storage [29]. Therefore they should be shared and transmitted only over TLS between authorization server and client. The binding between client and token has to be verified which is only possible with client authentication. As the client is not authenticated in the implicit code grant flow, a refresh token must not be issued in this case. Some challenges and questions regarding refresh tokens posted on SO are caused by this issue, which many developers didn't know, even though it is described in OAuth 2.0 specification.

5.3.9 Token Storage

In many implementations access or refresh tokens, user data etc. has to be stored for later reuse. In some solutions the libraries or provider SDKs handle token storage, while in other cases such as custom flow solutions developers have to store these data manually.

Cordova (C3)

In Cordova tokens and user data are mostly stored in HTML5 local storage according to plenty of code snippets and posts in Cordova's topic C3. Concerns and questions if it is secure for tokens and user data was not found in the topic. But actually local storage is vulnerable to cross-site scripting and therefore it is not recommended to store sensitive data in it [42].

Xamarin (X2)

Xamarin.Auth plugin supports secure data storage with an account store that is backed by Keychain services in iOS and KeyStore class in Android [87]. In various code snippets and posts of Xamarin topic X2 the account store is used or described. Moreover, we identified one question block where a developer asked how to store credentials securely on a device for offline usage and Xamarin.Auth was recommended, but with a warning that secure saving is not possible on rooted or jailbroken devices.

React Native (R2)

Discussions and questions how to store tokens securely in the app are more often in React Native topic R2 than in the other two topics. On one hand Keychain in iOS and SharedPreferences or Keystore in Android are recommended for secure storage. On the other hand React Native's storage system AsyncStorage is also mentioned in this context. However, in some posts it is warned and described that data in AsyncStorage are not encrypted, but based on the sandbox approach and private file system space of the operating system and runtime environment it can be considered as secure assuming the devices are not rooted or jailbroken.

5.4 Online Survey

To complement the evaluations of security and privacy challenges from the developers' perspective in cross-platform mobile app development, we conducted an online survey to gain deeper insights into developers' experiences and implementations of OAuth and OpenID Connect. The small data set in our survey is not suitable for quantitative analysis. However, the results presented in this section are reliable to show tendencies among developer's behavior and understandings.

5.4.1 Demographics

27 people participated in the online survey. We removed 7 participants from our data set as they were not software engineers. Table 5.5 shows demographic characteristics of the 20 participants we used for our analysis.

More than half were between 25 and 29 years old. Three participants were in their early 20s and another three in the early 30s. Only one person was younger than 20 or older than 35. One did not provide an answer. 16 participants were male and only one was female. Three did not indicate their gender. The country distribution shows that three quarters were from Austria, one participant from Denmark and another one from the United States. Again, three participants did not declare their country.

Demographic	Number	Percent
Gender		
Male	16	80%
Female	1	5%
Decline to answer	3	15%
Age		
15 - 19 years	1	5%
20 - 24 years	3	15%
25 - 29 years	11	55%
30 - 34 years	3	15%
35 - 39 years	1	5%
Decline to answer	1	5%
Country		
Austria	15	75%
Denmark	1	5%
USA	1	5%
Decline to answer	3	15%

Table 5.5: Participant Characteristics from the Online Survey

5.4.2 Prior Knowledge and Experience

Nearly all participants (19/20) reported to know the OAuth protocol, but only 13 reported to know OpenID Connect. In the course of these questions, we provided a short explanation on OAuth and explained that it is used for authorization to access third-party APIs. We furthermore explained that OpenID Connect is an identity layer on top of the OAuth 2.0 protocol for Single-Sign On. After this description, we asked the participants if they have already known these differences. 12 participants said yes, but 8 reported to not have known about this before.

We have been careful to ensure that participants do not get influenced by short descriptions we provide before or within the questions. But nevertheless, each kind of explanation introduces bias. The described differences between OAuth and OpenID Connect are only based on the protocol definitions. We required our participants to consider their experiences with OAuth and OpenID Connect separately. Thus, we had to ensure to bring all participants on the same level of knowledge. Therefore, we provided the described explanations, which introduced an intended bias.

The software engineers reported if and in which applications they have already implemented OAuth or OpenID Connect, as shown in Figure 5.6. Most participants had no experience with OAuth (11/20) or OpenID Connect (7/20). On the other hand, most of the developing experiences are based on web and mobile native apps. Seven participants have already implemented OAuth in web apps and five in mobile native apps. OpenID

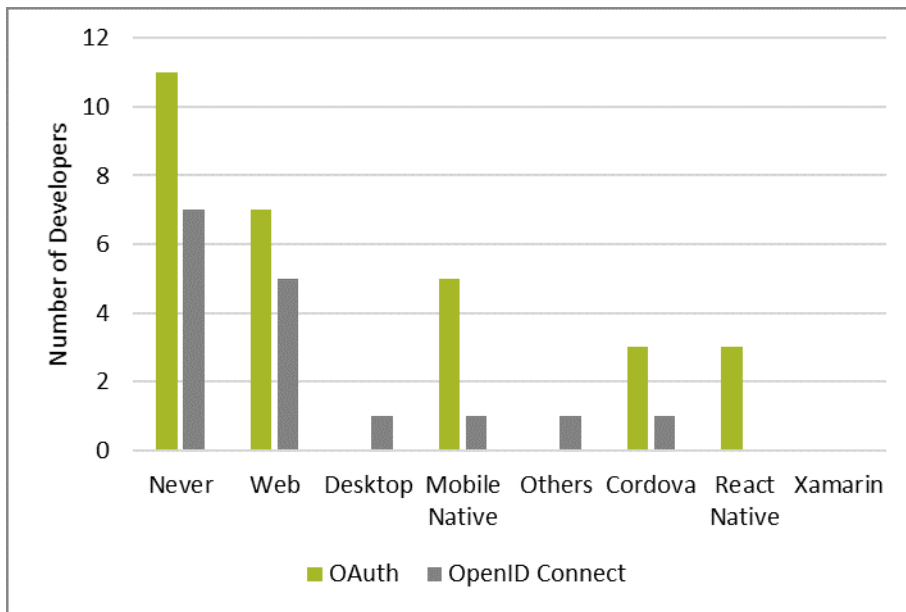


Figure 5.6: Developing Experience of OAuth/OpenID Connect

Connect has also been implemented by five participants in web apps, but only by one participant in mobile native apps. Three participants each reported having experience with OAuth in Cordova or React Native apps. Only one had experience with OpenID Connect in Cordova and no one in React Native. Moreover, none of the participants was experienced with OAuth or OpenID Connect in Xamarin.

5.4.3 User Groups

Based on their experiences, the participants were automatically assigned to one of the user groups *Cordova*, *React Native*, *Native* and *Inexperienced*. The groups were used to determine the respective set of questions the participants were asked in the remainder of the survey.

Three developers that have implemented OAuth in a Cordova app belong to the group *Cordova*. All of them also have experience with OAuth in web and mobile native apps. Only one of the Cordova developers has implemented OpenID Connect in a Cordova app, who is also experienced with it in web apps. One Cordova developer also has experience with OpenID Connect in a web app, but not in a Cordova app. The third Cordova developer has never implemented OpenID Connect.

The group *React Native* also contains three participants that have implemented OAuth in React Native apps. Two React Native developers also have experience with OAuth in web apps and one of them additionally in mobile native apps. Concerning OpenID Connect, one of the three React Native developers has implemented it in a web app

and another one in a desktop app. In contrast, no participant has implemented OpenID Connect in a React Native app.

We did not consider a dedicated group of *Xamarin*, because there was no participant with experience in OAuth or OpenID Connect and Xamarin.

Group *Native* consists of one participant who reported experience with OAuth and OpenID Connect only in mobile native apps.

The biggest group with 13 participants is referred to as *Inexperienced*. 11 of the 13 participants in this group have never implemented OAuth or OpenID Connect in a mobile app, but two of them have experience with OAuth and OpenID Connect in web or desktop development. Considering experience on cross-platform frameworks of the 13 participants shows that two developers are experienced with Cordova, one with Xamarin and another one with Cordova and React-Native. The remaining nine developers do not have cross-platform development experience. Hence, this group combines cross-platform mobile developers and other developers without OAuth or OpenID Connect experience as well as other developers with OAuth or OpenID Connect experience.

We argue that including this group of inexperienced developers in the survey is essential, because each developer regardless of previous knowledge can get into a situation where she has to implement OAuth or OpenID Connect in an application. Thus, we also evaluate first understandings of inexperienced developers based on short explanations on the basics before the questions.

5.4.4 User Group Questions

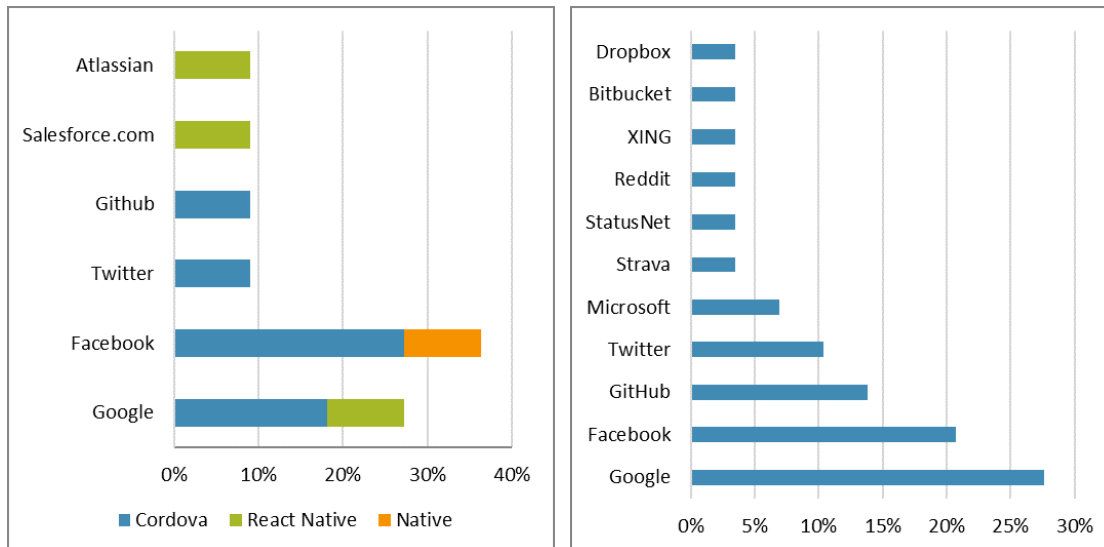
The major questions of the survey were based on the respective user group a participant was assigned to. Participants of the group *Inexperienced* were asked several *what would you do* questions about integrating OAuth in a mobile app. Short explanations about the concepts and parameters were given before each question. Developers of the other three groups *Cordova*, *React Native* and *Native* were asked the same questions about their OAuth implementations. Professional developers might have already developed multiple apps. To provide unified conditions and facilitate responses, we asked these developers to think on only one of their implemented apps when answering the questions.

Platforms. The native platforms for which the developers implemented apps were equally distributed across the groups *Cordova* and *React Native*. This means all three participants per group implemented their apps for Android and iOS. The one native developer reported to have implemented an iOS app. Other native platforms, such as Windows or Blackberry were not indicated in any group.

Providers. Regarding the OAuth/OpenID Connect providers, the experienced participants were asked to name up to 4 providers they integrated in their app. Cordova developers named the providers Facebook (3x), Google (2x), Twitter (1x) and Github (1x). Google, Salesforce.com and Atlassian were selected once by React Native developers

and Facebook was named by the native developer. All in all, the most frequently used provider in our sample was Facebook, followed by Google.

On the other hand, we asked the inexperienced developers to choose up to 6 providers they would probably include in a mobile app. Interestingly, the most frequently named providers were the same as the integrated ones named by the experienced groups. Google, Facebook, Github and Twitter were the top selected providers. Figure 5.7 shows these results in detail.



(a) Providers, that have been integrated in the past (b) Providers that users consider to integrate in the future

Figure 5.7: OAuth/OpenID Connect Providers

Concepts and Tools. Interesting observations between the concepts chosen by the experienced developers and the concepts the inexperienced developers would choose are shown in Figure 5.8.

Most inexperienced developers (46%) said they would implement the requests directly between the mobile app and OAuth provider API and only 8% would use a Backend as a Service (BaaS) provider like Firebase Auth or Auth0. The experienced participants answered it the other way round. Most of them (44%) have used a BaaS provider and only 22% implemented it directly between mobile app and OAuth provider API. Two developers reported to have used Firebase as BaaS provider.

The reasons for which the experienced developers chose the respective tools were different. Cordova and native developers mostly found them through search engines. Other Cordova developers read it on Stack Overflow or knew it from school. React Native developers also reported to know it, used Stack Overflow or got a recommendation from a friend.

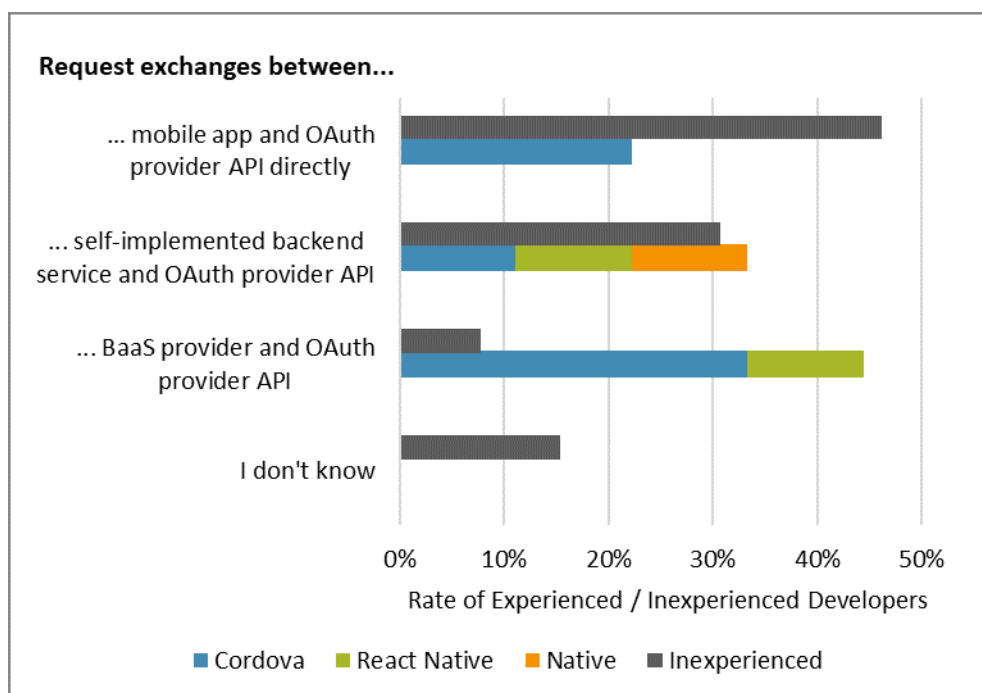


Figure 5.8: Concepts

Grant Flow. Developers across all three user groups of experienced developers did not know which grant flow was executed in their implementations. For more than half of the implementations (6/11) developers were not sure which flow was used, because the tools did it for them. In two implementations developers could not remember and in another two cases they reported that the hybrid flow was used. One React Native developer stated he had used the resource owner password flow.

Only three times developers were asked why they implemented the corresponding grant flow, because the question was omitted for previous answers like “..the tool did that for me” or “I can’t remember”. The developers reported that the flow was tried first (2x), the tools used it (2x) or it is the best flow (1x).

All in all, the decision about the grant flow was left to the tools and therefore remained largely unknown to the developers.

User-Agent. The decision about the user-agent was very different across experienced and inexperienced developers as described in Figure 5.9.

46% of inexperienced participants argued they would use in-app browser tabs like Chrome CustomTabs in Android or SafariViewController in iOS and 31% would use the insecure approach of embedded browsers like the WebView.

In case of the real world implementations it looks quite different. Developers reported

for all, but one Cordova implementations that they have used the WebView. For only one implementation a Cordova developer reported not to remember the user-agent. Also in nearly all React Native implementations the WebView was used. Only in one implementation an in-app browser tab was used. The native developer reported not to know the user-agent because the tools did the task for them.

A major reason for the selected user-agent was that the tools use it (Cordova 6/11, React Native 2/3). Moreover in three Cordova implementations the respective user-agent was tried first. In another two Cordova and one React Native implementation the developers used it because it was embedded in the mobile app.

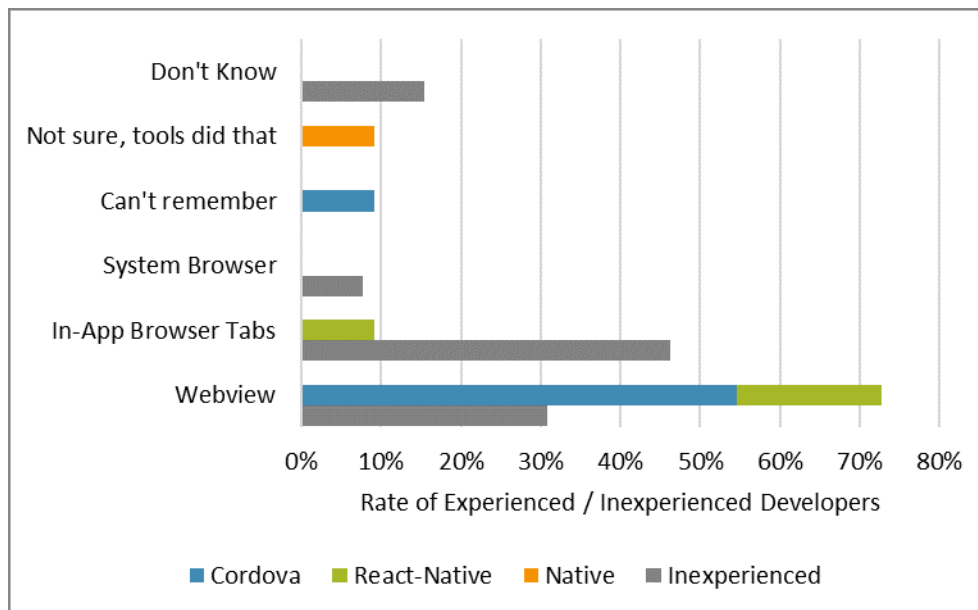


Figure 5.9: User-Agent

We furthermore asked the inexperienced developers how they would define a secure user-agent. We asked if it should be embedded in the mobile app or if it should run in a sandbox. The intention of this question was to verify the previous answer. Indeed, four answers did not fit to the previous ones. In three of these four answers the WebView was selected together with a sandbox environment, which indicates that the developers either thought a WebView would run within a sandbox or they did not mind on a secure approach when they decided about the user-agent. In another case a developer combined to use in-app browser tabs and a secure-agent should be embedded in the mobile app. We assume the developer relied on the words “in-app” and “embedded” to belong together, which is not the case. Moreover, three developers reported not to know what a secure user-agent should be.

Redirect URL. Figure 5.10 shows the answers about the used or probably used redirect URL, which were quite different between Cordova and React Native. In four of seven

Cordova implementations the *https app-URI scheme* was used, which is most recommended in OAuth best practices [12]. React Native developers reported they had used the *localhost interface* with IP address and port in two of three implementations. Moreover, the *custom URI scheme* was used in two Cordova and one native implementation.

One third of inexperienced developers selected a *https app-URI scheme* and another third did not know which scheme to choose. Nearly a fourth would use the *custom URI scheme* and only one developer would use the insecure *localhost interface* with *localhost* instead of the IP address.

Cordova, React Native and native developers decided for the respective URL because the tools described the pattern. Moreover, some Cordova developers tried this pattern first and one React Native developer read it on Stack Overflow.

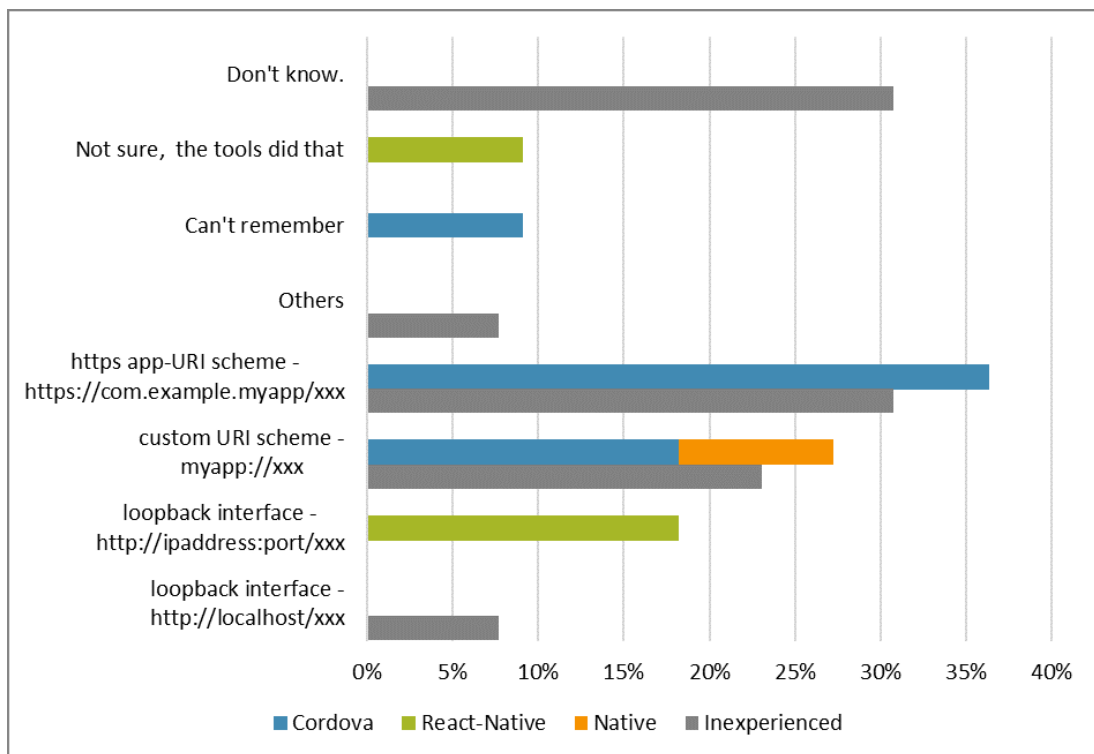


Figure 5.10: Redirect URL

PKCE. We asked the experienced developers if they had used PKCE in their implementations. Only one React Native developer wasn't sure, because maybe the tool did it and one Cordova developer indicated not to know PKCE. All others reported that they did not use PKCE.

Resources. All participants were asked to rank different resources by their frequency of use. Rank 1 means the most frequently used resource and rank 9 a hardly or never

used resource. Figure 5.11 shows the summed, relative weighted ranks per resource. This means, at first we generally weighted rank 1 with weight 9, rank 2 with weight 8, rank 7 with weight 3 and so on. Then we multiplied for each user group the amount of ranks per resource with its respective weight and summed it up. As a result we got an absolute weighted rank per resource for each user group. In the last step we calculated the relative weighted ranks per resource and user group, which are displayed in a stacked bar chart. It shows that the top four resources are search engines, Stack Overflow, provider and tool documentations.

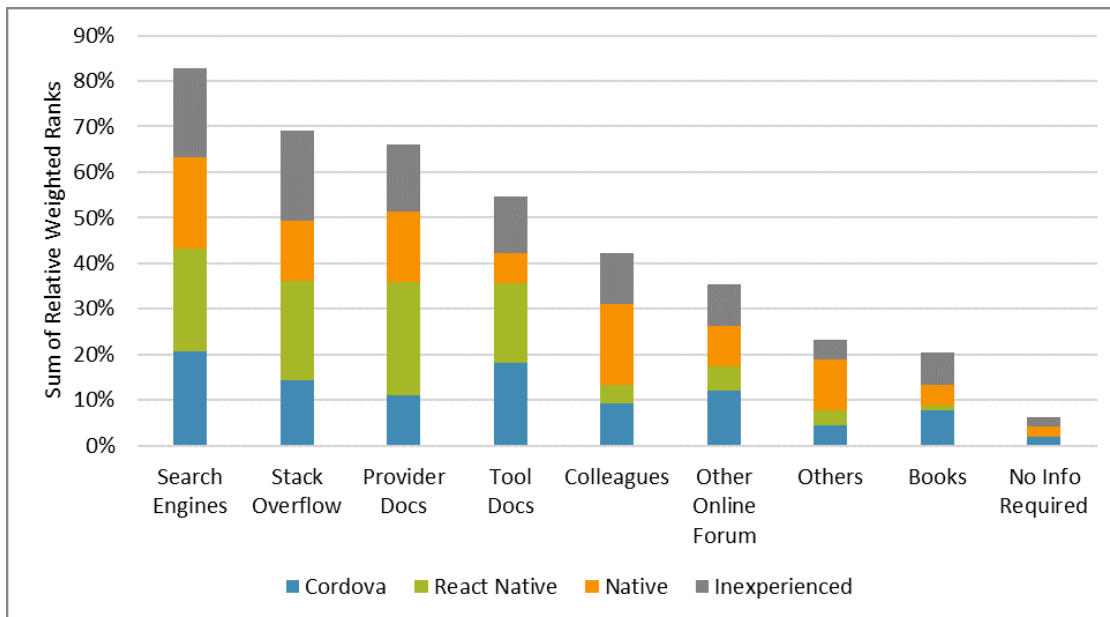


Figure 5.11: Resources Ranking

Discussion

In this chapter we discuss the results of comprehensive analyses based on the official documentation, Stack Overflow posts and survey to investigate security in cross-platform mobile apps.

6.1 Security Handling

Based on the official documentations, we have shown that the frameworks generally provide security-related information to their users, but not at the same level of detail. We found that Cordova and Xamarin generally published more security-relevant information than on React-Native.

Many parts of their documentations describe platform-specific issues and how they have to be handled and implemented within the frameworks. For example the configuration of Android permissions or property files in iOS are described for all three frameworks. Moreover, security-related information in context of the platform-driven issues are provided. This means security features of the platforms, such as ATS in iOS are described and even linked to the original resource on Apple's website across all three frameworks. But general information about secure communication like TLS is covered more by Xamarin and Cordova than by React-Native.

Each framework is built on various technologies, which are partly described in the respective documentations. Security-relevant aspects of the used technologies like JavaScript in Cordova and React-Native are only described in Cordova. They warn about using the JavaScript function *eval()* or about vulnerabilities of the WebView or HTML5 security. In React-Native's or Xamarin's documentations the technologies beyond are described only on a technical and functional level.

Most of the security warnings are stated as side notes within the pages of the respective topic. Thus, developers encounter security-relevant aspects at looking for and reading

general feature description. We found less questions about security on Stack Overflow, which indicates that only few developers actively search for security-related information. Thus, it is good practice to include it within the general information.

On the other hand, a more detailed discussion about security and privacy itself would be desirable to rise more awareness. Only Cordova provides an own section about privacy best practices. Although, the tips seem to be obvious on reading it, but they are still required to keep them in the developer's mind.

6.2 Developers' Challenges

Questions, problems and errors posted on Stack Overflow can be considered as challenges, where developers want to get further opinions or advice from more experienced developers. Our analysis of 520,000 Stack Overflow posts resulted in 200 fine-grained topics per framework, which we grouped into 12 meta topics.

Many meta topics consist of special keywords, such that the framework beyond can be detected. These framework-specific topics are also covered in the corresponding official documentation, such as storage or communication technologies like AJAX in Cordova, fetch API in React Native or XMLHttpRequest in Xamarin. Further popular topics we detected in posts across all three frameworks are about hardware devices like camera or bluetooth, files, media, messaging and authentication. One rather unexpected common topic is barcode scanning with a camera.

As the topics widely intersect with official documentation, we argue that Stack Overflow is widely used to get extended information which is missing or not understood in official documentation. Moreover, in the course of a manual analysis we found several questions with documentation snippets where developers asked the community for more details. Thus, there are many cases where the frameworks are insufficiently documented.

On the other hand, we also found questions where developers did not read documentation in advance and immediately asked in the forum for information that actually could have been found on the frameworks' websites. In these cases links to the official documentation were often sufficient to answer the Stack Overflow questions.

6.3 Authentication and Authorization

The meta topic about authentication consists of keywords about login, user, account and OAuth. These topics were not discussed in the frameworks' documentation with the exception of Xamarin's one. In the manual analysis of 100 posts per framework and the online survey about developers' experiences and opinions, we found major challenges and problems. We furthermore identified differences and similarities among the frameworks, but also among various groups of developers.

Facebook and Google, for example are the two major OAuth providers discussed and used in the cross-platform mobile apps. Chen et al. [10] also named them as popular providers.

Most-often-viewed Stack Overflow posts among all three frameworks are best practice questions and how to implement OAuth from beginners.

We also found similar major challenges or misunderstandings in the authentication posts across the frameworks. For example errors, because redirect URIs were not registered on the providers' side or using invalid ones were discussed regularly in Cordova and Xamarin posts, but only in a few React Native posts. Provider's like Facebook and Google actually describe in their documentation how to register an app in developer console, but developers still have difficulties with it. Thus, we argue that either developers don't find the correct documentation pages or they don't read it carefully. Nevertheless, relying on Stack Overflow is their way to make it work.

In one case Xamarin's documentation about Xamarin.Auth confused several developers. One example in the documentation shows the implementation with Facebook and includes a redirect URI. Many developers directly copied this code sample, but did not register their app on Facebook's developer console. Thus, the developers received *redirect URI mismatch* errors and did not understand why. This example shows the need for a full documentation that ensures an executable implementation.

We furthermore found misunderstandings about refresh tokens. In the official OAuth 2.0 specification it is described, that refresh tokens are not available in the implicit grant flow because client authentication is missed. However, developers across all three frameworks did not notice and asked on Stack Overflow about it. This indicates that developers mainly do not rely on original specifications, but rather on summarized and edited explanations, that are easier to work through.

We detected multiple discussions about the right grant flow across all three frameworks. Cordova and Xamarin posts mainly cover the implicit grant and authorization code grant, but React Native posts cover all four types. In the context of authorization grant, the risk of putting the client secret in the mobile app is often discussed. Many developers were not familiar with the associated risks. Some tried to play them down, but others described more or less secure alternatives. For example, separate config files for the secrets would not increase security, but an own proxy server that holds the secret and executes requests against the provider would work.

In the course of our survey, we asked about the used grant flow, but in most cases the developers could not answer, because they relied on the used tools. This means the tools implemented and executed the grant flow, such that developer themselves did not have to bother about it. Obviously abstraction and reuse of functionality are main ideas of third party libraries, frameworks or plugins. But on the other hand we claim that developers are trained to integrate tools without knowing what they are actually doing. They accept tools that only provide prose texts in their initial descriptions such that "it is a secure library..." and details are not questioned, which might cause severe vulnerabilities in

mobile apps. Other decisions about the user-agent or redirect-URL were also relied on the tools or based on their documentation.

The answers of experienced and inexperienced developers concerning OAuth/OpenID Connect implementations disclose differences between reality and theory. The small data set in our survey limits quantitative analysis, but nevertheless it is reliable to show at least some tendencies. For example we showed that most of the inexperienced developers would use the secure in-app browser tab as user-agent, but in reality only one of our experienced developers have really used it. More than 70 % used the insecure WebView instead. The WebView is vulnerable to intercept access tokens [10]. Moreover, the mobile apps can intercept keystrokes and cookie storage of the WebView, which violates the principle of least privilege. Thus, using an embedded browser as user-agent is risky and other types such as in-app browser tabs should be used [12].

In another example more than 40 % of our inexperienced developers would implement direct communication between mobile app and OAuth provider and less than 10 % would use a third party provider. In reality, the opposite was the case as more than 40 % used a third party provider and about 20 % the direct communication.

On the other hand, we also showed similarities among the user groups of the survey, at least in their resource preferences. They all reported to use mainly search engines, Stack Overflow and documentation of providers and tools to get information. We argue that using all these resources together is indeed the best way to get a comprehensive knowledge to establish a secure working solution.

Finally, we discuss what can be done to increase security in OAuth implementations. On the one hand the OAuth provider plays an important role. We detected incomplete or confusing documentations and therefore, we argue that OAuth providers should check and complete them accordingly.

From a cross-platform developer's point of view, we found that it is often difficult to find the right parts of a documentation about the provider tools. For example, some OAuth providers offer different SDKs and descriptions for native apps and web apps implemented with JavaScript, but do not specify which description is appropriate for a cross-platform framework. To minimize confusions, we recommend that OAuth providers should improve their documentation and add separate sections about the corresponding cross-platform frameworks. The documentation of provider platform *Auth0* is a good example how multiple frameworks on client side can be described.

On the other hand, we argue that also the documentations of Cordova and React Native have to be improved, as they do not provide any information about authentication or authorization. Only Xamarin currently covers the topic in its documentation. Thus, we suggest to provide general discussions about OAuth pitfalls as well as links to the OAuth best practices in the cross-platform frameworks' documentations.

Future work should also focus on the improvements of the tools such as plugins, libraries on both the provider's and client's side. They should prevent insecure implementations

from the ground. In the meanwhile Google refused OAuth requests from WebViews for security reasons. Also other OAuth providers should evaluate their implementations and adapt them to the current best practices.

Conclusion

The daily use of mobile devices in private and professional life results in new challenges, such as to satisfy various platforms and their fragmentation with appropriate applications or to protect the mass of sensitive data that are produced. Thus, we investigated challenges for secure mobile apps from the developers' perspectives with cross-platform frameworks.

We evaluated the official documentations of Apache Cordova, Xamarin and React Native and showed that all of these cross-platform frameworks include security topics, but in different levels of detail. Moreover, we analyzed the subjects of 520,000 Stack Overflow posts about all three frameworks with topic modeling, a natural language processing technique, which resulted in 200 fine-grained topics per framework. Based on the 10 keywords per topic, we labeled 600 topics and grouped them into 12 so-called meta topics representing common challenges in cross-platform mobile app development. We manually analyzed Stack Overflow posts about authentication. We discovered various challenges and misunderstandings, good and bad explanations as well as secure and insecure descriptions in context of authentication and authorization based on OAuth 2.0 protocol. Finally, we conducted a small online survey about developers' experiences and opinions. Based on the user groups with and without OAuth experience in mobile app development, we showed that the reality and theory concerning secure implementation often contradicts. In theory, the developers claim to use best practices, but in reality the less secure approaches are applied.

Online Survey

This chapter shows the questions of our online survey described in Section 4.3 and Section 5.4. The first few questions are the same for all user groups. Figures starting at Figure A.5 shows a sample survey for users that had already implemented a Cordova app and integrated Google OAuth. Questions for participants with no OAuth experience in mobile app development start at Figure A.10.



Figure A.1: Intro Text

1. Are you a software engineer?

Yes

No

Next

Lisa Leonhartsberger, BSc, TU Vienna - 2017

Figure A.2: First question

Mobile Cross-Platform Frameworks

Cross-platform frameworks in mobile app development follow the principle of "Write Once, Run Anywhere".

This means, you write your source code or parts of it once (e.g. with JavaScript, HTML5 and CSS) and due to the framework you can compile apps for multiple platforms, such as Android, iOS and Windows.

Moreover, the framework provides access to platform specific features and hardware, such as camera, filesystem, calendar, contacts,...

Some popular frameworks are Apache Cordova, Xamarin and React-Native.

2. Your skills in some software engineering fields

Please rate your skills of the following fields.

Not applicable: You have never had anything to do with that.

Fundamentals: You have some basic knowledge, but no experience.

Novice: You have already made your first experiences with it (e.g. < 1 year).

Advanced: You have advanced knowledge and experience (e.g. > 1 year, < 5 years).

Expert: You have a deep knowledge and long-time experience (e.g. > 5 years). Others ask for your advice.

	Not Applicable	Fundamentals	Novice	Advanced	Expert	No answer
Mobile Development (Frontend)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mobile Development (Backend)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Web Development (Frontend)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Web Development (Backend)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
IT Security	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Apache Cordova	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
React-Native	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Xamarin	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
JavaScript	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
C#	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Swift	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Java	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure A.3: Skills and Experiences (Part 1)

3. Have you ever developed a mobile app with a cross-platform framework?
If yes, which framework did you use?
Multiple selection is possible. Please enter framework names if text input field is available.

No, I have never developed a mobile cross-platform app.

Yes, I used Apache Cordova

Yes, I used a framework based on Apache Cordova (e.g. PhoneGap, Ionic,...)

Yes, I used Xamarin

Yes, I used React-Native

Yes, I used another framework:

I can't remember.

I don't know.

4. Do you know the OAuth protocol?
Have you ever heard or read about it or do you have experience with it?

Yes

No

5. Do you know OpenID Connect?
Have you ever heard or read about it or do you have experience with it?

Yes

No

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.3: Skills and Experiences (Part 2)

OAuth (for App Authorization)

OAuth is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.[1]

This means, if you want to access and use third-party APIs (e.g. Google API, Facebook API, Twitter API,...) in your application, you might get in touch with OAuth protocol.

**6. Have you ever developed apps that used APIs that were protected by OAuth?
If yes, in which apps did you integrate the APIs?**

<input type="checkbox"/> No, I have never integrated such APIs	<input type="checkbox"/> in Apache Cordova apps
<input type="checkbox"/> in web apps	<input type="checkbox"/> in apps based on Apache Cordova
<input type="checkbox"/> in desktop apps	<input type="checkbox"/> in React-Native apps
<input type="checkbox"/> in mobile native apps (e.g. Android, iOS, Windows,...)	<input type="checkbox"/> in Xamarin apps
<input type="checkbox"/> others: <input type="text"/>	<input type="checkbox"/> in other mobile cross-platform apps – <input type="checkbox"/> which framework(s)? <input type="text"/>

I can't remember. I don't know, if they were protected with OAuth protocol.

OpenID Connect (for End-User Authentication)

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It allows apps to verify the identity of the End-User as well as to obtain basic profile information. [2]

This means, if you want integrate Single Sign-On with a third-party provider (e.g. Google, Facebook, Twitter,...) in your application, you might get in touch with OpenID Connect 1.0.

**7. Have you ever developed apps with Single Sign-On to identity provider's based on OpenID Connect?
If yes, in which apps did you implement it?**

<input type="checkbox"/> No, I have never implemented SSO with OpenID Connect	<input type="checkbox"/> in Apache Cordova apps
<input type="checkbox"/> in web apps	<input type="checkbox"/> in apps based on Apache Cordova
<input type="checkbox"/> in desktop apps	<input type="checkbox"/> in React-Native apps
<input type="checkbox"/> in mobile native apps (e.g. Android, iOS, Windows,...)	<input type="checkbox"/> in Xamarin apps
<input type="checkbox"/> others: <input type="text"/>	<input type="checkbox"/> in other mobile cross-platform apps – <input type="checkbox"/> which framework(s)? <input type="text"/>

I can't remember. I don't know, if it was OpenID Connect.

Figure A.4: Experiences OAuth/OpenID Connect (Part 1)

8. Did you know the difference between OAuth and OpenID Connect before this survey?

Yes

No

[Next](#)

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.4: Experiences OAuth/OpenID Connect (Part 2)

Cordova App

In the following we will ask you some questions about your Cordova app you have implemented once.

In case of multiple Cordova apps, please focus your answer's of all questions on the same app.

9. For which platforms is your Cordova app available?

Multiple selection is possible

Android

iOS

Windows

Blackberry

Other:

10. Which providers did you include in your Cordova app?

Please name up to 4 providers.

OAuth/OpenID Connect Provider

1.

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.5: Example Cordova/Google (Part 1)

Cordova App

In the following we will ask you some questions about your Cordova app you have implemented once.

In case of multiple Cordova apps, please focus your answer's of all questions on the same app.

Google

The following questions are about your OAuth implementation with Google in your Cordova App.

11. Which concept did you choose to implement OAuth/OpenID Connect in your Cordova app?

Between which components did you implement the authorization requests. It also means which components handled authorization codes and tokens?

Requests were exchanged between...

- ... Cordova app and Google API directly (without any backend service)
- ... self-implemented backend service and Google API
- ... BaaS provider (e.g. Auth0, Firebase Auth) and Google API

... others:

- I can't remember.
- I don't know.

12. Which tools did you use to implement OAuth/OpenID Connect for Google?

Multiple selection is possible. Please add the name(s) of the used tools.

- Cordova plugins:
- client-side self-implementation
- official native SDK of Google
- server-side self-implementation
- BaaS provider (e.g. Firebase Auth, Auth0,...):
- others:

- I can't remember.

Figure A.6: Example Cordova/Google (Part 2)

13. Why did you choose the tools?
Multiple selection is possible. Please add the name(s) of the used resources.

I knew the tools from:

I found them via a search engine (e.g. Google):

I found them on Stack Overflow

I found them on another online forum:

Other reasons:

I can't remember.

14. Which grant flow did you use for Google?

Implicit Flow

Authorization Code Flow

Provider Authentication Flow

Resource Owner Password Flow

Client Credentials Flow

Hybrid Flow

I can't remember.

I'm not sure, because the tool(s) did that for me.

I don't know what a grant flow is.

15. Which user-agent did you use?
The end-user has to authenticate himself with his user credentials and grant your Cordova app access to the Google API. This interaction between end-user and provider is executed via the user-agent.
Please select one answer.

Webview

In-App browser tabs (e.g. Chrome CustomTabs, SafariViewController)

System Browser

Others:

I can't remember.

I'm not sure, because the tool(s) did that for me.

I don't know.

Figure A.6: Example Cordova/Google (Part 3)

16. How did you construct the redirect-URI?
The redirect-URI is used to direct the user-agent back to the app after interaction between end-user and provider is finished. Moreover, the URI contains sensitive data such as authorization codes, access and id tokens.
Please select one answer.
Attention: You are not asked to enter your real redirect-URI, only the pattern you used.

loopback interface – http://localhost/xxx

loopback interface – http://ipaddress:port/xxx

custom URI scheme – myapp://xxx

https app-URI scheme – https://com.example.myapp/xxx

Others:

I can't remember.

I'm not sure, because the tool(s) did that for me.

I don't know.

17. Did you use PKCE (Proof Key for Code Exchange) in your implementation?
Please select one answer.

Yes

No

I can't remember.

I'm not sure, because the tool(s) may have done that for me.

I don't know what PKCE is.

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.6: Example Cordova/Google (Part 4)

Cordova App

In the following we will ask you some questions about your Cordova app you have implemented once.

In case of multiple Cordova apps, please focus your answer's of all questions on the same app.

18. Why did you implement the selected grant flow?

Multiple selection is possible.

I used Authorization Code Flow, because

- the tool(s) use this flow.
- I know, it is the best flow for mobile apps.
- I tried to implement this flow at first and it worked.
- I read in a documentation it should be used. Which one?
- I read on Stack Overflow it should be used.
- I read in/on , it should be used.
- of another reasons:

- I can't remember.
- I don't know.

19. Why did you use the selected redirect-uri?

Multiple selection is possible. Please describe which documentation you used.

I used "https app-URI scheme – https://com.example.myapp/xxx" as redirect-uri, because

- the tool(s) described this pattern to use.
- I tried this pattern at first and it worked.
- I read in a documentation it should be used. Which one?
- I read on Stack Overflow it should be used.
- I read in/on , it should be used.
- of another reasons:

- I can't remember.
- I don't know.

Figure A.7: Example Cordova/Google (Part 5)

20. Why did you choose the selected user-agent?
 Multiple selection is possible. Please describe which documentation you used.

I used In-App browser tabs (e.g. Chrome CustomTabs, SafariViewController) as user-agent, because

the tool(s) use it.

I tried it at first and it worked.

it is embedded in my mobile app.

it runs in a sandbox environment, separated from my mobile app.

I read in a documentation it should be used. Which one?

I read on Stack Overflow it should be used.

I read in/on , it should be used.

of another reasons:

I can't remember.

I don't know.

21. Which information resources did you use most frequently?
 Please rank the options by importance, which means by frequency of use. (At least 2 ranks)
 1 – most used
 9 – hardly/never used

Search Engines	Stack Overflow	1
Tool Docs	Google Docs	2
Other Online Forum	Colleagues	3
Books	Others	4
No Info Required		5
		6
		7
		8
		9

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.7: Example Cordova/Google (Part 6)

22. Do you want to tell us anything else about your experiences with OAuth/OpenID Connect?
e.g. problems, issues you were faced with; what was disturbing, what did you like?

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.8: Example Cordova/Google (Part 7)

23. Finally, we'd like to ask you for some details about yourself.

Gender:

female male

I prefer not to say.

Age:

< 15 years 25 – 29 years 40 – 44 years 55 – 59 years
 15 – 19 years 30 – 34 years 45 – 49 years > 60 years
 20 – 24 years 35 – 39 years 50 – 54 years

I prefer not to say.

Country No answer

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.9: Example Cordova/Google - Demographics

Scenario

Please imagine, you develop a new mobile app with lots of fancy features for your end-users. Most of your users already have user accounts on the platforms of big and well-known providers. Therefore you want to provide your users the ability to login in your mobile app with their existing credentials from other services.

In the following, we will ask you about your ideas and methods to implement such a third-party login based on OAuth/OpenID Connect in a mobile app.

9. Which information resources would you use to get to know about OAuth/OpenID Connect and how to implement it in your app?

Please rank the options by importance, which means by frequency of use. (At least 2 ranks)

1 – most used

9 – hardly/never used

Search Engines	Stack Overflow	1
Tool Docs	Provider Docs	2
Other Online Forum	Colleagues	3
Books	Others	4
No Info Required		5
		6
		7
		8
		9

10. Which providers would you probably include in your app?

You can name up to 6 providers.

OAuth/OpenID Connect Provider
1. <input type="text"/>

Figure A.10: No Experience (Part 1)

11. Which concept would you choose to integrate OAuth/Open ID Connect in a mobile app?

Between which components would you implement the authorization requests. It also means which components would handle authorization codes, tokens and session data?
(OAuth provider might be Google, Facebook, Twitter,...)

Authorization requests should be exchanged between...

... mobile app and OAuth provider API (with native SDK of the provider)

... self-implemented backend service and OAuth provider API

... BaaS provider (e.g. Auth0, Firebase Auth) and OAuth provider API

... other:

I don't know.

12. Which user-agent would you use?

The end-user has to authenticate himself with his user credentials and grant the mobile app access to the provider API. This interaction between end-user and provider is executed via the user-agent.
Please select one answer.

Webview

In-App browser tabs (e.g. Chrome CustomTabs, SafariViewController)

System Browser

Others:

I don't know.

13. Which statement about user-agent do you agree with?

The end-user has to authenticate himself with his user credentials and grant your mobile app access to the provider API. This interaction between end-user and provider is executed via the user-agent.

A secure user-agent should...

be embedded in my mobile app.

run in a sandbox environment, separated from my mobile app.

I don't know.

Figure A.10: No Experience (Part 2)

14. How would you construct the redirect-URI?

The redirect-URI is used to direct the user-agent back to the app after interaction between

end-user and provider is finished. Moreover, the URI contains sensitive data such as authorization codes, access and id tokens.

Please select one answer.

Attention: You are not asked to enter a real redirect-URI, only a pattern you would use.

- loopback interface – http://localhost/xxx
- loopback interface – http://ipaddress:port/xxx
- custom URI scheme – myapp://xxx
- https app-URI scheme – https://com.example.myapp/xxx

Others:

- I don't know.

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.10: No Experience (Part 3)

15. Finally, we'd like to ask you for some details about yourself.

Gender:

female male

I prefer not to say.

Age:

< 15 years 25 – 29 years 40 – 44 years 55 – 59 years
 15 – 19 years 30 – 34 years 45 – 49 years > 60 years
 20 – 24 years 35 – 39 years 50 – 54 years

I prefer not to say.

Country No answer

Next

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.11: No Experience - Demographics

Thank you for completing our survey!

We would like to thank you very much for helping us.

Your answers were transmitted, you may close the browser window or tab now.

Lisa Leonhartsberger, BSc, TU Vienna – 2017

Figure A.12: Finished

List of Figures

3.1	Architecture of a Cordova Application (cf. [50])	12
3.2	Xamarin - Shared Code (cf. [67])	14
3.3	Architecture of Xamarin.Android (cf. [69])	15
3.4	Architecture of Xamarin.iOS (cf. [66])	15
3.5	Architecture of React Native (cf. [33])	16
4.1	Stack Web Viewer UI - Search Input Screen	25
4.2	Stack Web Viewer UI - Result Screen	26
4.3	OAuth 2.0 Protocol in Mobile Apps (cf. [29])	29
5.1	Threat Models of WebViews (cf. [35])	36
5.2	Number of Auth-Questions per Year and Framework	47
5.3	Auth-Question View Counts by Creation Date	49
5.4	Distribution of OAuth Protocols in Xamarin.Auth Code Snippets	53
5.5	Number of OAuth Grant Types in C3	56
5.6	Developing Experience of OAuth/OpenID Connect	65
5.7	OAuth/OpenID Connect Providers	67
5.8	Concepts	68
5.9	User-Agent	69
5.10	Redirect URL	70
5.11	Resources Ranking	71
A.1	Intro Text	81
A.2	First question	82
A.3	Skills and Experiences (Part 1)	83
A.3	Skills and Experiences (Part 2)	84
A.4	Experiences OAuth/OpenID Connect (Part 1)	85
A.4	Experiences OAuth/OpenID Connect (Part 2)	86
A.5	Example Cordova/Google (Part 1)	87
A.6	Example Cordova/Google (Part 2)	88
A.6	Example Cordova/Google (Part 3)	89
A.6	Example Cordova/Google (Part 4)	90
A.7	Example Cordova/Google (Part 5)	91
A.7	Example Cordova/Google (Part 6)	92

A.8 Example Cordova/Google (Part 7)	93
A.9 Example Cordova/Google - Demographics	94
A.10 No Experience (Part 1)	95
A.10 No Experience (Part 2)	96
A.10 No Experience (Part 3)	97
A.11 No Experience - Demographics	98
A.12 Finished	99

List of Tables

4.1	Number of SO Tags	21
4.2	Post Type Frequency	21
4.3	Posts across Framework Tags	22
4.4	Rate of (Un-)Answered Questions across Tags	23
4.5	Authentication Topics	27
5.1	Cordova - Storage APIs	35
5.2	Security Topics in Official Documentations	43
5.3	Meta Topics	44
5.4	Number of Topics across Frameworks	44
5.5	Participant Characteristics from the Online Survey	64

Bibliography

- [1] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, May 2016.
- [2] V. Ahti, S. Hyrynsalmi, and O. Nevalainen. An Evaluation Framework for Cross-Platform Mobile App Development Tools: A Case Analysis of Adobe PhoneGap Framework. In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*, CompSysTech '16, pages 41–48, New York, NY, USA, 2016. ACM.
- [3] M. Ali, M. E. Joorabchi, and A. Mesbah. Same App, Different App Stores: A Comparative Study. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '17, pages 79–90, Piscataway, NJ, USA, 2017. IEEE Press.
- [4] L. An, O. Mlouki, F. Khomh, and G. Antoniol. Stack Overflow: A Code laundering platform? In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 283–293. IEEE, 2017.
- [5] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.
- [6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [7] D. Buhov, M. Huber, G. Merzdovnik, and E. Weippl. Pin It! Improving Android Network Security At Runtime. In *IFIP Networking 2016*, May 2016.
- [8] D. Buhov, M. Huber, G. Merzdovnik, E. Weippl, and V. Dimitrova. Network Security Challenges in Android Applications. In *10th International Conference on Availability, Reliability and Security (ARES 2015)*, Aug. 2015.
- [9] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft. What information about code snippets is available in different software-related documents? An exploratory study. In *2017 IEEE 24th International Conference*

- on Software Analysis, Evolution and Reengineering (SANER)*, pages 382–386, Feb. 2017.
- [10] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague. OAuth Demystified for Mobile Application Developers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14*, pages 892–903, New York, NY, USA, 2014. ACM.
 - [11] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein. Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 323–328, July 2013.
 - [12] W. Denniss and J. Bradley. OAuth 2.0 for Native Apps. BCP 212, RFC Editor, Oct. 2017.
 - [13] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba. Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal*, 8(2):163–190, 2017.
 - [14] Facebook Inc. Props. <http://facebook.github.io/react-native/docs/props.html>, July 2016. Accessed: 2017-01-22.
 - [15] Facebook Inc. State. <http://facebook.github.io/react-native/docs/state.html>, July 2016. Accessed: 2017-01-22.
 - [16] Facebook Inc. Tutorial. <http://facebook.github.io/react-native/docs/tutorial.html>, July 2016. Accessed: 2017-01-22.
 - [17] Facebook Inc. AsyncStorage. <https://facebook.github.io/react-native/docs/asyncstorage.html>, Dec. 2017. Accessed: 2018-01-10.
 - [18] Facebook Inc. Generating Signing APK. <https://facebook.github.io/react-native/docs/signed-apk-android.html>, Dec. 2017. Accessed: 2018-01-10.
 - [19] Facebook Inc. Images. <https://facebook.github.io/react-native/docs/images.html>, Dec. 2017. Accessed: 2018-01-10.
 - [20] Facebook Inc. Introducing JSX. <https://facebook.github.io/react/docs/introducing-jsx.html>, Jan. 2017. Accessed: 2017-01-22.
 - [21] Facebook Inc. Networking. <https://facebook.github.io/react-native/docs/network.html>, Dec. 2017. Accessed: 2018-01-10.
 - [22] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan. Android Security: A Survey of Issues, Malware Penetration, and Defenses. *IEEE Communications Surveys Tutorials*, 17(2):998–1022, Secondquarter 2015.

- [23] F. Fischer, K. Böttinger, H. Xiao, C. Stranksy, Y. Acar, M. Backes, and S. Fahl. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *IEEE Symposium on Security and Privacy*, 2017.
- [24] R. Francese, C. Gravino, M. Risi, G. Scanniello, and G. Tortora. Mobile App Development and Management: Results from a Qualitative Investigation. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '17, pages 133–143, Piscataway, NJ, USA, 2017. IEEE Press.
- [25] M. Georgiev, S. Jana, and V. Shmatikov. Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks. In *NDSS Symposium*, volume 2014, page 1. NIH Public Access, 2014.
- [26] Google Developers. OAuth 2.0 for Mobile & Desktop Apps. <https://developers.google.com/identity/protocols/OAuth2InstalledApp>, Sept. 2017. Accessed: 2018-01-03.
- [27] S. Guthrie. Microsoft to acquire Xamarin and empower more developers to build apps on any device. <https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device>, Feb. 2016. Accessed: 2016-12-21.
- [28] S. Habchi, G. Hecht, R. Rouvoy, and N. Moha. Code Smells in iOS Apps: How Do They Compare to Android? In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '17, pages 110–121, Piscataway, NJ, USA, 2017. IEEE Press.
- [29] D. Hardt. The OAuth 2.0 Authorization Framework. Technical Report 6749, RFC Editor, Oct. 2012.
- [30] H. Heitkötter, S. Hanschke, and T. A. Majchrzak. *Evaluating Cross-Platform Development Approaches for Mobile Applications*, pages 120–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [31] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri. Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 66–77, New York, NY, USA, 2014. ACM.
- [32] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real Challenges in Mobile App Development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24, Oct. 2013.
- [33] M. Konicek. Under the Hood of React Native. <http://www.reactnative.com/under-the-hood-of-react-native/>, Nov. 2015. Last Accessed: 2018-01-22.

- [34] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk. An exploratory analysis of mobile development issues using stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 93–96, May 2013.
- [35] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin. Attacks on WebView in the Android System. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 343–352. ACM, 2011.
- [36] B. Martin, M. Brown, A. Paller, D. Kirby, and S. Christey. 2011 CWE/SANS top 25 most dangerous software errors. *Common Weakness Enumeration*, 7515, 2011.
- [37] M. Martinez and S. Lecomte. Towards the Quality Improvement of Cross-Platform Mobile Applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 184–188, May 2017.
- [38] A. K. McCallum. Mallet: A machine learning for language toolkit. 2002.
- [39] Mono Project. About Mono. <http://www.mono-project.com/docs/about-mono/>, Oct. 2016. Accessed: 2016-12-28.
- [40] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna. A Large-Scale Study of Mobile Web App Security. In *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.
- [41] OWASP. Mobile Top 10 2016-Top 10. https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10, Mar. 2016. Accessed: 2016-12-05.
- [42] OWASP. HTML5 Security Cheat Sheet. https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet, Nov. 2017. Accessed: 2018-01-09.
- [43] I. Ristić. SSL and TLS Deployment Best Practices. <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>, May 2017. Accessed: 2018-01-24.
- [44] C. Rosen and E. Shihab. What are mobile developers asking about? A large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, June 2016.
- [45] N. Sakimura, J. Bradley, and N. Agarwal. Proof Key for Code Exchange by OAuth Public Clients. RFC 7636, RFC Editor, Sept. 2015.
- [46] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*, 2014.
- [47] J. Solanky, K. Patil, and G. Patel. Resemblance of PhoneGap and Titanium for Mobile Application Development. *International Journal of Computer Applications*, 144(10), 2016.

- [48] Statista. Leading mobile app development SDKs worldwide 2016. <https://www.statista.com/statistics/742418/leading-mobile-app-development-sdks/>, Dec. 2016. Accessed: 2017-11-23.
- [49] M. Steyvers and T. Griffiths. Probabilistic topic models. *Latent Semantic Analysis: A Road to Meaning*. T. Landauer, D McNamara, S. Dennis, and W.Kintsch, eds. Laurence Erlbaum, 427(7):424–440, 2006.
- [50] The Apache Software Foundation. Cordova Architecture. <https://cordova.apache.org/docs/en/6.x/guide/overview/index.html#architecture>, Apr. 2016. Accessed: 2016-12-15.
- [51] The Apache Software Foundation. Cordova Plugin Whitelist. <https://cordova.apache.org/docs/en/6.x/reference/cordova-plugin-whitelist/index.html>, Aug. 2016. Accessed: 2016-12-12.
- [52] The Apache Software Foundation. Privacy Guide. <https://cordova.apache.org/docs/en/6.x/guide/appdev/privacy/index.html>, Apr. 2016. Accessed: 2016-12-09.
- [53] The Apache Software Foundation. Security Guide. <https://cordova.apache.org/docs/en/6.x/guide/appdev/security/>, Apr. 2016. Accessed: 2016-12-09.
- [54] The Apache Software Foundation. Storage. <https://cordova.apache.org/docs/en/6.x/cordova/storage/storage.html>, May 2016. Accessed: 2016-12-09.
- [55] The Apache Software Foundation. Whitelist Guide. <https://cordova.apache.org/docs/en/latest/guide/appdev/whitelist/index.html>, Oct. 2016. Accessed: 2017-01-19.
- [56] The PostgreSQL Global Development Group. PostgreSQL 9.6.3 Documentation - 8.11 Text Search Types. techreport. Available: <https://www.postgresql.org/docs/9.6/static/datatype-textsearch.html> Accessed: 2017-06-09.
- [57] P. K. Venkatesh, S. Wang, F. Zhang, Y. Zou, and A. E. Hassan. What Do Client Developers Concern When Using Web APIs? An Empirical Study on Developer Forums and Stack Overflow. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 131–138, June 2016.
- [58] H. Wang, Y. Zhang, J. Li, and D. Gu. The Achilles Heel of OAuth: A Multi-platform Study of OAuth-based Authentication. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications, ACSAC '16*, pages 167–176, New York, NY, USA, 2016. ACM.

- [59] H. Wang, Y. Zhang, J. Li, H. Liu, W. Yang, B. Li, and D. Gu. Vulnerability Assessment of OAuth Implementations in Android Applications. In *Proceedings of the 31st Annual Computer Security Applications Conference, ACSAC 2015*, pages 61–70, New York, NY, USA, 2015. ACM.
- [60] S. William and B. Lawrence. *Computer Security: Principles And Practice*. Prentice Hall, 2nd edition, Jan. 2012.
- [61] M. Willocx, J. Vossaert, and V. Naessens. Comparing Performance Parameters of Mobile App Development Strategies. In *Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16*, pages 38–47, New York, NY, USA, 2016. ACM.
- [62] Xamarin Inc. About. <https://www.xamarin.com/about>. Accessed: 2016-12-21.
- [63] Xamarin Inc. An Introduction to Xamarin.Forms. <https://developer.xamarin.com/guides/xamarin-forms/getting-started/introduction-to-xamarin-forms/>. Accessed: 2016-12-23.
- [64] Xamarin Inc. Android - WebView. https://developer.xamarin.com/guides/android/user_interface/web_view/. Accessed: 2017-01-12.
- [65] Xamarin Inc. Introduction to Portable Class Libraries. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/. Accessed: 2016-12-23.
- [66] Xamarin Inc. iOS Architecture. https://developer.xamarin.com/guides/ios/under_the_hood/architecture/. Accessed: 2016-12-22.
- [67] Xamarin Inc. Sharing Code Options. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/. Accessed: 2016-12-21.
- [68] Xamarin Inc. Working with Files. <https://developer.xamarin.com/guides/xamarin-forms/working-with/files/>. Accessed: 2017-01-12.
- [69] Xamarin Inc. Xamarin.Android - Architecture. https://developer.xamarin.com/guides/android/under_the_hood/architecture/. Accessed: 2016-12-22.
- [70] Xamarin Inc. Xarmarin.Mobile. <https://components.xamarin.com/view/xamarin.mobile>, Oct. 2015. Accessed: 2016-12-28.

- [71] Xamarin Inc. Android Data Access - Introduction. https://developer.xamarin.com/guides/android/application_fundamentals/data/part_1_introduction/, Sept. 2016. Accessed: 2016-12-30.
- [72] Xamarin Inc. App Transport Security. https://developer.xamarin.com/guides/ios/platform_features/introduction_to_ios9/ats/, Sept. 2016. Accessed: 2016-12-28.
- [73] Xamarin Inc. Authenticating a RESTful Web Service. <https://developer.xamarin.com/guides/xamarin-forms/web-services/authentication/rest/>, Sept. 2016. Accessed: 2017-01-12.
- [74] Xamarin Inc. Authenticating Acces to Web Services. <https://developer.xamarin.com/guides/xamarin-forms/web-services/authentication/>, Sept. 2016. Accessed: 2017-01-12.
- [75] Xamarin Inc. Authenticating Users with an Identity Provider. <https://developer.xamarin.com/guides/xamarin-forms/web-services/authentication/oauth/>, Sept. 2016. Accessed: 2017-01-12.
- [76] Xamarin Inc. Authenticating Users with Azure Mobile Apps. <https://developer.xamarin.com/guides/xamarin-forms/cloud-services/authentication/azure/>, June 2016. Accessed: 2017-08-28.
- [77] Xamarin Inc. Introduction to Web Services. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/web_services/, May 2016. Accessed: 2016-12-28.
- [78] Xamarin Inc. iOS - WebViews. https://developer.xamarin.com/guides/ios/user_interface/uiwebview/, 2016. Accessed: 2017-01-12.
- [79] Xamarin Inc. iOS Data Access. https://developer.xamarin.com/guides/ios/application_fundamentals/data/, Sept. 2016. Accessed: 2016-12-30.
- [80] Xamarin Inc. iOS Data Access - Introduction. https://developer.xamarin.com/guides/ios/application_fundamentals/data/part_1_introduction/, Sept. 2016. Accessed: 2016-12-30.
- [81] Xamarin Inc. Platform Features of Android. https://developer.xamarin.com/guides/android/platform_features/, Aug. 2016. Accessed: 2016-12-28.
- [82] Xamarin Inc. Platform Features of iOS. https://developer.xamarin.com/guides/ios/platform_features/, Sept. 2016. Accessed: 2016-12-28.
- [83] Xamarin Inc. ProGuard. https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/proguard/, Dec. 2016. Accessed: 2017-01-12.

- [84] Xamarin Inc. Protect the Application. https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_application/part_1_-_preparing_an_application_for_release/#protect_app, Dec. 2016. Accessed: 2017-01-12.
- [85] Xamarin Inc. Security and Privacy Enhancements. https://developer.xamarin.com/guides/ios/platform_features/introduction-to-ios10/security-privacy-enhancements/, Sept. 2016. Accessed: 2017-01-12.
- [86] Xamarin Inc. Xamarin.Android Data Access. https://developer.xamarin.com/guides/android/application_fundamentals/data/, Sept. 2016. Accessed: 2016-12-30.
- [87] Xamarin Inc. Securely Store Credentials. <https://developer.xamarin.com/recipes/cross-platform/xamarin-forms/general/store-credentials/>, Aug. 2017. Accessed: 2018-01-09.
- [88] Xamarin Inc. Transport Layer Security (TLS). <https://developer.xamarin.com/guides/cross-platform/transport-layer-security/>, Oct. 2017. Accessed: 2018-01-10.
- [89] S. Xanthopoulos and S. Xinogalos. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA, 2013. ACM.
- [90] R. Yang, W. C. Lau, and S. Shi. *Breaking and Fixing Mobile App Authentication with OAuth2.0-based Protocols*, pages 313–335. Springer International Publishing, Cham, 2017.
- [91] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology*, 31(5):910–924, Sept. 2016.
- [92] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang. An Empirical Study on Stack Overflow Using Topic Analysis. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 446–449, Piscataway, NJ, USA, 2015. IEEE Press.