Developers Are Users Too: Helping Developers Write Privacy Preserving and Secure (Android) Code

Duc Cuong Nguyen
Saarland University
s9ddnguy@stud.uni-saarland.de

Yasemin Acar, Sascha Fahl
CISPA, Saarland University
acar,fahl@cs.uni-saarland.de

Michael Backes
CISPA and MPI-SWS, Saarland University
backes@cs.uni-saarland.de

Abstract

Despite security advice in the official documentation and an extensive body of security research about vulnerabilities and exploits, many developers fail to write secure code for their Android apps. Oftentimes, Android code fails to adhere to secure best practices, leaving the apps vulnerable to a multitude of attacks. Our approach focuses on identifying these weak links in Android code and offering developers secure options in an Android Studio plugin similar to a spell-checking mechanism known from text documents. Instead of trying to identify vulnerable Android apps in existing markets, we can effectively prevent an app from being accidentally dangerous right at the development stage, before it reaches users' hands. We propose a formal and direct support for developers in building secure Android applications. The usability of the tool is established in an on-site developer problem discovery study, and its real world use is put to the test in a telemetry study after its rollout to interested Android developers.

Author Keywords

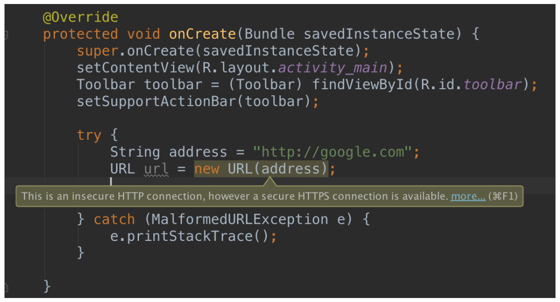
Android Best Practices; Developer Security; Usable Security for Developers, Developers Support, Android Security

Introduction

Whenever developers are anything less than cautious and knowledgeable and act accordingly, they put users' privacy and security at risk. Even in the absence of malicious intentions by developers, benign failure to write privacy preserving or secure code can lead to applications that leave user data vulnerable to leaks and attacks. Previous research has found that many mobile apps have poorly implemented privacy and security mechanisms, potentially because developers are inexperienced, distracted or overwhelmed [5]. Developers tend to risk users' privacy and security by requesting more permissions than actually needed [11, 12], not using TLS [9, 10] or cryptographic APIs [7] correctly, often using dangerous options for Inter Component Communication [6], and fail to store sensitive information in private areas. This behaviour frequently leads to the leakage of private user information [8]. In addition to educating and supporting end users with helpful transparent tools to help them take action to preserve their privacy and protect their security, we are convinced that it is important to admit that developers are users too – and oftentimes lack the time or understanding to always put their users' privacy and security first. By helping developers during the implementation process, we directly improve the privacy and security of users in ways that save them time and attention, offering them solutions that are usually only accessible to power users: Making code privacy preserving and secure.

Supporting developers in writing better code

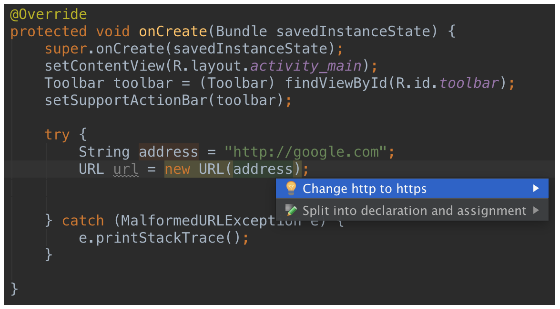
We are developing a plugin for Android Studio. This plugin guides Android application developers in scenarios that require them to make privacy or security relevant decisions. We extracted pitfalls from the large body of research we address in our Systematization of Knowledge [4] and privacy and security best practices from Android's official documentation. Using static analysis techniques on our body of all (some 1,4 million) free Android apps, we identify which privacy and security pitfalls occur in the wild. We map these dangerous code choices to privacy and security preserving



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    try {
        String address = "http://google.com";
        URL url = new URL(address);

    } catch (MalformedURLException e) {
        e.printStackTrace();
    }

}
```

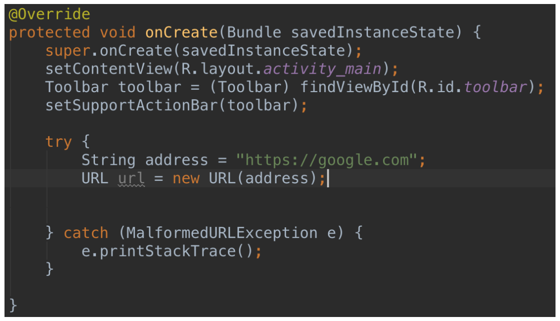
This is an insecure HTTP connection, however a secure HTTPS connection is available. more... (⌘F1)

Figure 1: Our plugin detects a bad practice, highlights and warns the developer.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    try {
        String address = "http://google.com";
        URL url = new URL(address);

    } catch (MalformedURLException e) {
        e.printStackTrace();
    }

}
```

Figure 2: Our plugin suggests a quick fix.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    try {
        String address = "https://google.com";
        URL url = new URL(address);


    } catch (MalformedURLException e) {
        e.printStackTrace();
    }

}
```

Figure 3: The developer uses our suggested quick fix.

code. In our Android Studio plugin, we scan the developers' input for code that translates to less-than-ideal privacy and security. Additionally, we detect whenever they paste a code snippet and use our existing knowledge of freely accessible code snippets online to give our users unobtrusive feedback about the privacy and security impact of their chosen code snippet. With a usability similar to spell-checking, we inform developers whether their choices are dangerous (cf. Figure 1) and offer a preview of possibly better alternatives (cf. Figure 2) that developers can choose to replace the concerned code with by using a simple shortcut i. e. as a quickfix (cf. Figure 3).

Evaluation plan

We offer users of our plugin the choice to opt into our telemetry feature that we use to quantify the usability and helpfulness, but also the failures and limitations of our plugin. More precisely, we design our plugin to gather bad practices' information anonymously (e.g How many bad practices happen in a project, does the developer ever read our warning message, how long did he spend to read, was a provided quickfix applied). From this information, we will gain a better understanding of how developers interact with our plugin. Our prototype was well received by the research and IT community present at Cebit 2016, a major international IT fair that conveniently takes place in our vicinity. When we asked developers to interact with our plugin prototype, we witnessed firsthand the relief developers experienced when they were exposed to a usable and unobtrusive way to making their code more privacy preserving and secure. We are in close contact with the developers interested in our plugin, and use their feedback to evaluate and continuously improve the usability and feature set of our plugin. Further more, during development, we are conducting expert reviews with developers of Android apps. Additionally, we are preparing a field study with Android application developers to quantify our plugin's contribution in real world circumstances.

Android Studio Plugin

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA [2]. Therefore, we decided to implement our approach as an Android Studio plugin, instead of a plugin for Eclipse [1] or another IDEs. The IntelliJ IDEA platform provides APIs so we can inspect developers' code in many ways:

- Inspect new statement (e.g new URL("http://google.com");)
- Inspect declaration statement (e.g `Random random;`)
- Inspect assignment statement (e.g `int x = random.nextInt();`)

Having these APIs helps us easily detect whenever developers write dangerous code in terms of security and privacy. For more complicated combinations of code, we use secure information flow to back/forward track insecure code. For example, when developers read the string from a password field and store the password on the external SD card, our plugin will detect the dangerous code and give a warning message against this combination. Whenever a mistake is detected, beside a warning message, we also provide developer a concrete solution to correct their mistake. By leveraging the quick-fix mechanism [3] of the IntelliJ platform, we offer developers the option to use a single simple short cut to replace the existing dangerous code by secure code (e.g replace a TrustManager that accepts all certificates by certificate pinning).

Our contributions

Our Android Studio plugin helps developers to easily implement privacy and security best practices. With our plugin, we are:

- Protecting app users against leakage of personally identifiable information by careless developers (e. g. upgrading to secure and privacy preserving network connections whenever possible).
- Supporting (laymen) developers in selecting libraries including but not limited to advertising, crash reporting and analytics libraries that provide an adequate level of privacy protection for end users.
- Supporting developers in choosing secure and privacy preserving cryptographic primitives and protocols (e. g. using a strong symmetric cryptographic algorithm, with the correct cipher mode and padding).
- Preventing privacy leaks caused by unsafe storage of sensitive user information (e. g. preventing the unencrypted storage of sensitive user information on an external storage such as SD cards).
- Protecting developers against the use of dangerous code snippets pasted from StackOverflow and other information sources.
- Helping developers to write apps that protect their users against user fingerprinting attacks (e. g. using randomized identifiers instead of unique (hardware) tokens).
- Providing transparent interfaces and workflows to help developers make effective privacy decisions.

REFERENCES

1. 2016. Eclipse. `https://eclipse.org`. (2016). [Online; accessed: 2016-06-06].

2. 2016. IntelliJ. `https://www.jetbrains.com`. (2016). [Online; accessed: 2016-06-06].

3. 2016. Quick-Fixes for Code Issues. `https://www.jetbrains.com/help/resharper/10.0/Code_Analysis__Quick-Fixes.html`. (2016). [Online; accessed: 2016-06-06].

4. Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. 2016a. SoK: Lessons Learned From Android Security Research For Appified Software Platforms. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP '16)*.

5. Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016b. You Get Where You're Looking For: The Impact Of Information Sources On Code Security. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP '16)*.

6. Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. 2011. Analyzing Inter-application Communication in Android. In *Proc. 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*. ACM. `DOI`: `http://dx.doi.org/10.1145/1999995.2000018`

7. Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*. ACM. `DOI`: `http://dx.doi.org/10.1145/2508859.2516693`

8. William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. 2011. A Study of Android Application Security. In *Proc. 20th Usenix Security Symposium (SEC'11)*. USENIX Association. `http://www.enck.org/pubs/enck-sec11.pdf`

9. Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*. ACM. `DOI`: `http://dx.doi.org/10.1145/2382196.2382205`

10. Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. 2013. Rethinking SSL Development in an Appified World. In *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*. ACM. `DOI`: `http://dx.doi.org/10.1145/2508859.2516655`

11. A. Porter Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. 2011. Android Permissions Demystified. In *Proc. 18th ACM Conference on Computer and Communication Security (CCS'11)*. ACM.

12. Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steve Hanna, and Erika Chin. 2011. Permission Re-Delegation: Attacks and Defenses. In *Proc. 20th Usenix Security Symposium (SEC'11)*. USENIX Association.