Mobile Security for Banking on Android Platform

Lukas Aron Petr Hanacek
Faculty of Information Technology
Brno University of Technology
Bozetechova 2 Brno, Czech Republic
{iaron, hanacek}@fit.vutbr.cz

ABSTRACT

This paper discusses the proposal of protecting mobile banking applications on an Android platform. The mobile platform surpasses computers in its popularity in many aspects of ones daily routine. One of the basic feature is its connection to the internet and the possibility of transferring money using the mobile device. Despite threats that are available on mobile devices like security breaches, data leakage and other types of malware, the popularity of using mobile banking is more popular than a few years ago. Mobile banking applications have the same security protection as any other mobile application, thus without additional specific protection for mobile banking supported by an operating system. This paper proposes novel protection for applications which checks if the mobile banking application communicates to the current bank company and the data (usually passwords, money amount etc.) are not sent to someone else. There is a discussion about another control mechanism: permissions which are used to access mobile device hardware.

KEYWORDS

Mobile Banking, Protection, Android, Malware, Android security.

1 INTRODUCTION

Almost everything that users can use on their desktop can also be accessed on their mobile devices, in particular on smart phones. These devices have become powerful with capabilities that desktops did not even have several years ago. Increased capabilities have come hand in hand with increased security threats. Nowadays, all our private and business data is accessible on our phones or tablets. Data can be stored on the device or accessed via Internet. Thus, these devices increasingly have become the targets of malicious attacks. The most successful and widespread operating system of mobile devices is Android [6]. Needless to say, the operating system plays a key role for the security and privacy of these devices.

Permission is the corner stone of applications security policy on the Android operating system. Giving permission to applications can lead to data leaks because it may, for example, allow these applications to provide access to personal contacts, private messages or any other kind of sensitive data saved on the current device with an ability to access the Internet. A malicious application may thus send all sensitive data to a server on the other side of the world.

Fine-grained permission setting is not yet possible in Android (it should be possible since Android 6 [1] is in use). Due to the popularity of mobile devices and the use of an internet connection everywhere, internet banking is getting usual routine of the day. However these applications work with users money, they have the same security protection as any other application. There is no existing additional protection of these applications than using secure protocols, passwords or two-sided verification.

Moreover, applications for using bank accounts are usually developed by third party companies without the exact knowledge of the security in this banking area. This paper introduces an additional protection for mobile banking applications based on Android. The basic idea behind the proposal is to wrap the application with an additional protection layer. This layer sets additional checking rules for filtering the connection between an application and the server. There is real-time filtering if the application communicates with a bank server and there is no data leakage. When data or an information leakage attempt appears, the user is warned and the also user has the ability to confirm or deny this connection. In the next part of this paper the basic introduction into Android security architecture is documented followed by related work. The next section describes the main concept of the proposal that can be implemented on each version of the Android operating system. The final part is the conclusion aimed towards future research possibilities.

2 ANDROID SECURITY OVERVIEW

This part contains the basic introduction into Android security architecture. In order to understand the main idea of this paper, the necessary security basics are covered here. For another source of deep dive in Android security principles read [2]. Android is a privilege-separated (or permission-separated) operating system, in which each application runs with a distinct system identity (Linux user ID and group ID). Parts of the system are also separated into distinct identities. Linux thereby isolates applications from each other and from the system. Additional finer-grained security features are provided through a permission mechanism that enforces restrictions on the specific operations that a particular process can perform, and per-URI permissions for granting ad-hoc access to specific pieces of data.
The central design point of the Android security architecture is that no application, by default, has permission to perform any operation that would adversely impact other applications, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another applications files, performing network access, keeping the device awake, etc. Because each Android application operates in a process sandbox, applications must explicitly share resources and data. They do this by declaring the permissions they need for additional capabilities are not provided by the basic sandbox. Applications statically declare the permissions they require (declaration is setup during the development of the application), and the Android system prompts the user for consent at the time the application is installed.
The application sandbox does not depend on the technology used to build an application. In particular, the current virtual machines (runs the byte or native code) Dalvik/Art VM [12] are not a security boundary, and any application can run a native code. All types of applications - Java, native, and hybrid are sandboxed in the same way and have the same degree of security from each other.

2.1 User IDs and File access

During installation, Android assigns each package a distinct Linux user ID. The identity remains constant for the duration of the packages (applications) life on a current device. This Linus ID can be different for the same package (application) on another device. The most important fact is that this number is unique per device. Because security enforcement happens at the process level, the code of any two packages cannot normally run in the same process. Any data stored by an application will be assigned that applications user ID. Android does not allow direct access to the data of one application from another.

2.2 Using permissions

A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely impact the user experience or any data on the device. In order to make use of protected features of the device, the developer of the application has to include in the specific file AndroidManifest [8] line with one (or more) `<uses-permission>` tag declaring the permissions that the current application requires.
The package installer grants the permission during the installation of the application based on the AndroidManifest file. No checks with the user are done while the application is running: the application is either granted a particular permission when installed, and can use that

feature as desired, or permission is not granted and any attempt to use the feature fails without prompting the user.
In almost all cases, however, a permission failure will be printed to the system log. However, in a normal user situation (such as when the application is installed from the official Google Play Store) an application cannot be installed if the user does not grant the application each of the requested permissions. This principle should be changed since Android 6 Marshmallow is now available. So there is no need to worry about runtime failures caused by missing permissions because the mere fact that the application is installed means that the application has been granted its desired permissions.
The application has a defined permission as was described above. There is a basic schema of the flow of communication between the application and operating system, as it is implemented in Android OS (see Figure 1). In the schema is the application which uses its permission defined during its development. These permissions are mapped into operating system calls. There are also user's files (usually documents, pictures, etc.) stored on the current mobile device.

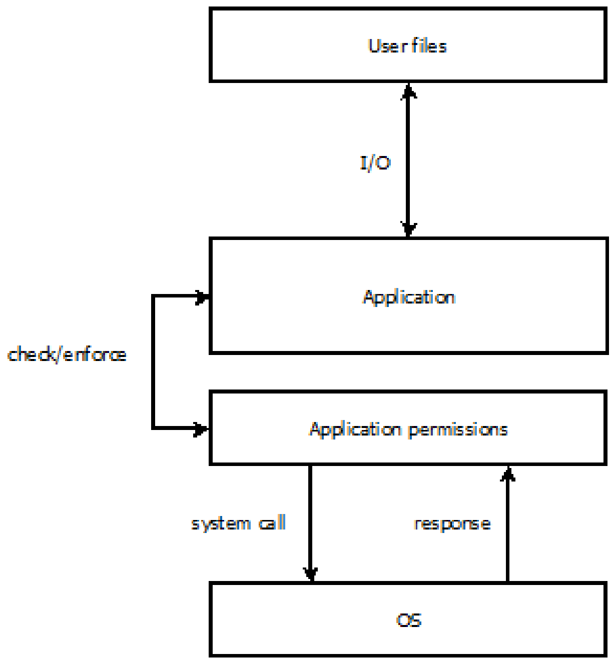

Figure 1. Application communication flow

If the user of the device wants to work with the files through the application, system calls are raised for input/output operations which are needed to communicate with the application.

3 RELATED WORK

During the evolution of the Android operating system a lot of additional features were defined. There is an enormous list of updates, recommendations by community or novel application approaches to upgrade this system. In this section is a small group of the extension to the Android which is related to the topic of security. Diverse threats are identified from mobile services, including permission bypass [13], [15], and fraudulent money transfers with online banking [7], [14]. The following work is more related to implementing improvements of Android operating system to cover more security principles than its basic version.
First of all there is the project Aurasium [16] which is the cornerstone of this proposal. Aurasium introduces the solution that bypasses the need to root the device when modification of the Android OS is required. The behavior of the application can be modified or the flow of the information can be followed. This project automatically repackages arbitrary applications to keep the sandboxing mechanism and policy enforcement code, which closely watches the behavior of security and privacy intrusions such as attempts to retrieve a users sensitive information, etc. Aurasium has the ability to detect and prevent cases of privilege escalation attacks. Experiments show that these principles can apply this solution to a larger scale of malicious applications with a near 100 percent success rate, without significant performance and space overhead.
Dr. Android and Mr. Hide [4] Fine-grained security policies on unmodified Android is another approach for checking an Internet connection. They presented a concept for replacing coarse Android platform permissions with finer-grained permissions that lower needed privilege levels, decreasing the potential threat from malicious apps. The system contains two novel parts: Mr. Hide and Dr. Android. Mr. Hide is an Android service that

protects sensitive device capabilities with fine-grained permissions which the service dynamically enforces. For example, Mr. Hide includes an API for accessing the Internet but protects this API with a new permission called `InternetURL(d)`, which only grants access to the Internet domain d. The Google search application can use Mr. Hide to access the Internet by replacing Androids full Internet permission with the much weaker `InternetURL(www.google.com)` permission, providing increased confidence to both the developer and users. The second part is Dr. Android which stands for Dalvik Rewriter for Android. This concept is very useful and has a similar target as our concept. The difference between our concept and Dr. Android and Mr. Hide is that we do not modify any Android permissions and we are focusing only on protecting a specific region of applications. The weak point of this solution (Dr. Android and Dr. Hide) is that it is only focused for domains and unfortunately does not cover the connection directly to the IP address.

TaintDroid [3] is another extension to the Android platform that tracks the flow of privacy sensitive data by third-party applications. TaintDroid supposes that downloaded applications are not trusted, and tracks in real-time the way these applications access users data. Their primary goals are to detect when sensitive data leaves the system by third-party applications and to simplify analysis of applications by mobile device users or external security services. TaintDroid implementation modifies the Dalvik VM of Android in order to introduce variable level taint tracking. For tracking taint sources, TaintDroid does variable level tracking for an interpreted code, method level tracking for a native code, message level tracking for Inter Process Communication and File level tracking for secondary storage files.

TrustDroid [17] implements the famously known principle of BYOD - Bring Your Own Device [9]. This principle is related to privacy protection of leakage from corporate data. It can operate on a server (offline) and also real-time on the mobile device based on Android. The static analysis is performed offline. Static semantic analysis on a compiled byte code is done for data flow tracking. From the aspect of semantic analyzing of byte code it is converted to a tree structure, which is again processed to generate call graph. This call graph is processed to show the source to sink paths. In order to generate the tree structure, a parser is built based on an open source third-party solution ANTLR parser generator. The TrustDroid tracking is engine composed of a pair of definition (source and sink), file sniffer and the glue between these components. This engine explores the output of the semantic analyzer by using the file sniffer and then performs taint tracking. TrustDroid usually works as a standalone application with permission to access file system (which needs a specific permission or root privilege) and scan installation packages (apk files). Restrictions of this principle are an inability to provide an analysis of a dynamic loaded code and Java Native Interfaced [5] code.

4 PROTECTION OF MOBILE BANKING APPLICATION

This part is the main content of this paper. It includes the description of the proposal of protection for mobile banking applications on an Android platform. Protection for a mobile banking application is related to login data leakage. This leakage usually has the following characteristic. The corrupted bank application has injected own domain or IP address. A user logins his username or login number with a password and this information is sent to the third party server which collects this data and then redirects the communication to the correct bank server. Now the hacker has the ability to login to the bank account. Another additional protection, such as SMS confirmation, is usually useless, because if the user has corrupted the mobile banking application then it has the permission to read the SMS messages and can hide the incoming confirmation.

This paper proposal has been built on top of Aurasium project [16], which is designed for user application on the Android device and to track all system calls that the application raises. During real-time execution the user can see the calling permissions, execution of I/O operations or using internet connection. This proposal uses the basic features of the Aurasium project such as decompiling the apk package of application and adding related hooks to be able to track the flow of the application. The additional rules or behavior is out of the Aurasium project and is a part of this proposal.

Currently we have the possibility to track an internet connection call. This has been done through system functions: `getaddrinfo()` and `connect()`. These functions are responsible for DNS [10] resolving and socket connection, thus the proposal can track all Internet connections related to domains and also direct IP address calls. More information about Linux system calls can be found in [11]. Intercepting them allows us to control the applications Internet access. The possibility to check the internet calls is the main power of checking if the bank application communicates with the corresponding server. The proposal solution has two parts: a mobile banking application which can be corrupted, infected by malware or it can be malware free, and the information about the website domains or IP addresses of the current bank company. The bank company can provide this list to its users. We have a list of website domains and IP addresses of the 5 biggest banks in the Czech Republic. This list is necessary to provide correct checking. If the list is not available or the bank does not provide it, the implementation will still work, but the user will be informed about every internet call and then he has to decide to confirm or deny the call. The recommendation is to use the verified list of bank domains which can be provided by the current bank.

The following steps are necessary to have additional control over the data flow. At first step we need the package file of the application (for mobile banking) and this package has to be repacked (unzipped) by our project. This means that the application goes through the process of unpacking the package, adding additional hooks for system calls, inserting the prepared list of website domains or IP addresses (as was discussed above) and creating the package with a signing process.

A modified version of the package is ready to install and the results of the installation are the same working application that they were before, but there is an additional feature to track the flow of the application calls to the kernel. This can be used to track all important calls, especially for tracking calls like: `getaddrinfo()` and `connect()`. On top of tracking internet communication there is the matching mechanism to control the use of the permitted list of domains or IP addresses.

The proposal of mobile banking protection is about creating the hooks and then track all internet calls of the application and provide feedback to the user if the application uses only permitted domains provided by bank company. In this case the proposal can detect the leakage of the data to different website domains that is defined in the information of the bank.

The correct behavior of protection against the infected application is shown by the warning to the user when there is an attempt to access a different website domains which is not allowed (is not covered by the list from the bank). The next step is up to user if he continues the process of the application or terminates it.

There is a graph of a proposal of the solution for protection of a mobile banking application (see Figure 2).

Figure 2 contains a few parts: the biggest one is the mobile banking protection which is based on a modified application, a list of allowed bank websites or IP address or a combination thereof. Inside this box is mobile banking application which can have saved credentials or keys. All saved files are transmitted to the application through I/O calls which is the whole application context with all necessary files. The next part of the proposal is the user who plays the role coordinator or decision maker. He has the ability to deny or confirm access to the website domain or IP address which is not on the list of allowed address.

In this case the application permissions contains at least the Internet connection request

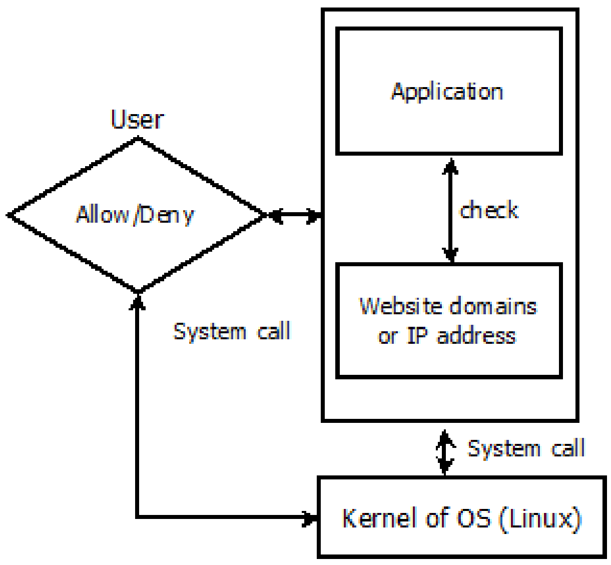

Figure 2. Proposal of protection

or some hardware information permission. As was proposed the solution needs two inputs: the application which is in the cover and the bank information. Afterwards when the user runs the application the mobile banking protection checks all the system calls related to the website domain of the bank according to the bank information. This graph is the same for all versions of Android operating system without the need to use root. This proposal can be modified to other kinds of protection against data leakage. The mobile banking applications needs only Internet connection permission; however, they ask during the installation process for more than that. There can be another warning when the application tries to access more than just the Internet connection, such as a phone/email contacts, in order for example.

5 EXPERIMENTS

We have chosen five banks in the Czech Republic for which we have a list of website domains or IP addresses. We have downloaded the corresponding applications to communicate with the bank using a mobile device based on the Android operating system. We did modifications of these applications in order to send the entire communication to our testing server. Afterwards, we provided the list from the related bank to a related mobile application and setup the proposed protection. While we were trying to login to the current bank company we were warned about the disallowed website or IP address that we were trying to connect. This protection works on each application that we were tested. We have not provided the list of bank companies in this proposal.

6 CONCLUSION AND FUTURE WORK

The security issues in the mobile world has become more and more sophisticated and their detection is more complicated. The Main responsibility is on the user, even if he uses the mobile device to work. The user is the weakest point in the whole string of the security chain. The proposal of protecting the mobile banking application is in trying to ease the user responsibility to check everything that the application does in the background. This approach could save the user (or corporation) money and should deny the leakage of password from the device. This proposal shows another type of protection against malware infection. The user can detect a possible malfunction of the application related to the connection between banking application and the server. This protection can be extended by additional features to protect the application in more ways than the track system calls which are only responsible for solve the DNS records. The next step of this work will need more mathematical models and additional tracking features in order to protect users against data or money leakage.

7 ACKNOWLEDGEMENTS

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by the project CEZ MSM0021630528 Security-Oriented Research in Information Technology and by project FIT-S-11-1 Advanced Secured, Reliable and Adaptive IT.

REFERENCES

[1] DEITEL, P. J., WALD, A., AND DEITEL, H. M. Android 6 for programmers.

[2] DRAKE, J. J., LANIER, Z., MULLINER, C., FORA, P. O., RIDLEY, S. A., AND WICHERSKI, G. *Android hacker's handbook*. John Wiley & Sons, 2014.

[3] ENCK, W., GILBERT, P., HAN, S., TENDULKAR, V., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS) 32*, 2 (2014), 5.

[4] JEON, J., MICINSKI, K. K., VAUGHAN, J. A., FOGEL, A., REDDY, N., FOSTER, J. S., AND MILLSTEIN, T. Dr. android and mr. hide: fine-grained permissions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices* (2012), ACM, pp. 3–14.

[5] LIANG, S. *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley Professional, 1999.

[6] LIU, W., ZHANG, G., CHEN, J., ZOU, Y., AND DING, W. A measurement-based study on application popularity in android and ios app stores. In *Proceedings of the 2015 Workshop on Mobile Big Data* (2015), ACM, pp. 13–18.

[7] MARFORIO, C., KARAPANOS, N., SORIENTE, C., KOSTIAINEN, K., AND CAPKUN, S. Smartphones as practical and secure location verification tokens for payments. In *NDSS* (2014).

[8] MEIER, R. *Professional Android 4 application development*. John Wiley & Sons, 2012.

[9] MILLER, K. W., VOAS, J., AND HURLBURT, G. F. Byod: Security and privacy considerations. *It Professional*, 5 (2012), 53–55.

[10] MOCKAPETRIS, P., AND DUNLAP, K. J. *Development of the domain name system*, vol. 18. ACM, 1988.

[11] NEGUS, C. *Linux Bible 2010 Edition: Boot Up to Ubuntu, Fedora, KNOPPIX, Debian, openSUSE, and 13 Other Distributions*, vol. 682. John Wiley & Sons, 2010.

[12] OH, H.-S., KIM, B.-J., CHOI, H.-K., AND MOON, S.-M. Evaluation of android dalvik virtual machine. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems* (2012), ACM, pp. 115–124.

[13] ONGTANG, M., MCLAUGHLIN, S., ENCK, W., AND MCDANIEL, P. Semantically rich application-centric security in android. *Security and Communication Networks 5*, 6 (2012), 658–673.

[14] REAVES, B., SCAIFE, N., BATES, A., TRAYNOR, P., AND BUTLER, K. R. Mo (bile) money, mo (bile) problems: analysis of branchless banking applications in the developing world. In *24th USENIX Security Symposium (USENIX Security 15)* (2015), pp. 17–32.

[15] WU, L., DU, X., AND ZHANG, H. An effective access control scheme for preventing permission leak in android. In *Computing, Networking and Communications (ICNC), 2015 International Conference on* (2015), IEEE, pp. 57–61.

[16] XU, R., SAÏDI, H., AND ANDERSON, R. Aurasium: Practical policy enforcement for android applications. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)* (2012), pp. 539–552.

[17] ZHAO, Z., AND OSONO, F. C. trustdroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on* (2012), IEEE, pp. 135–143.