Masters Proposal

Richard Allen
STUDENT NUMBER: 998184

PROPOSAL SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF RESEARCH IN LOGIC AND COMPUTATION
DEPARTMENT OF COMPUTER SCIENCE
SWANSEA UNIVERSITY

Thesis committee:

Markus Roggenbach

Swansea University
Department of Computer Science
January 2020

Masters Proposal

Richard Allen

1. Introduction

There are 3.5 billion smart phone users world wide (citation required). Smart phones are becoming deeply integrated into our every day lives. We use them to navigate, hail taxis, order goods online and even to pay for goods using contactless payment. But there are dangers associated with the adoption of these devices. When a person installs an application from an app store how can they be certain it is not performing some malicious activites that they are unaware of?

Smart phones communicate to cell towers using a protocol called GSM. There is a related telecomms protocol called TETRA (citation require) that is used by emergency services for radio communication. The TETRA standard allows a special call type to be made to TETRA radios that the radio will answer immediately without ringing and without the users intervention. The microphone on the radio and transmits the ambient noise around the radio user to the caller without the user knowing this is taking place. The emergency services are currently transitioning from TETRA radios to GSM smart phones that have greater data bandwidth. This enables them to use body-mounted video cameras to stream a video feed to the headquaters.

How can a person be certain that it is not possible to use their smart phone to eavesdrop on their conversations? How can a person be certain that an application they installed is not relaying their GPS location to an observer without their knowledge? How can a person be sure the forward facing camera on their phone is not being used to surreptitiously send images of them to an unknown observer?

For the sake of peoples privacy and the prevention of crime the user must always be aware when installed software is using sensitive features of their device.

Of those 3.5 billion smart phones users, 2.5 billion are using Android devices. Android does provide a security system that prevents applications from accessing sensitive features of the device without having been given permission by the user but it is conceivable that this security system could be circumvented. If an application that has been granted permissions to read the users contacts were to pass that information to another application that has permission to contact an outside party then the two could collude with one another to expose the users contacts without their knowledge.

The project aims to increase the security of Android devices and the confidence that people have in using them. This benefits people in our everyday lives and people in specialised areas of work. As the police services transition to smart phones for communication between police officers and police control rooms, the privacy of that communication is paramount to the saftey and effectiveness of their work.

The proposal is to use techniques from runtime verification to recognise when operating system calls are being use to read sensitive devices on the phone such as the camera, GPS, and microphone or sensitive data held on the phone such as the address book. The techniques will be used further to look for collusion between installed apps that are granted security privaleged and those that are not.

The project will achieve the overall security aim by logging the activity of all installed applications and monitoring that log for indications of collusion. Suspect logs will have tell-tale signs within the entries that we will call properties of the log. Those properties can be defined using liner time temporal logic (LTL) and software can be generated that will monitor the log an evaluate when those properties are present.

The monitor will be realized as an android application and there will be a period during the project where the monitor is evaluated to discover if it can recognise collusion between two applications that are known to collude.

2. Related Work

There are existing studies into Android security that the project intends to build upon.

The paper "Detection of app collusion potential using logic programming (1)" details how to perform static analysis of application code to identify when there is possible collusion between applications. Application code is analysed to identify an applications ability to send and receive data and to use protected resources. These abilities are encoded as Prolog facts and Prolog rules operate on these facts to identify where collusion may be taking place.

The work into static analysis is continued in the following paper "Software Model Checking for Mobile Security – Collusion Detection in K (2)". This paper investigates another approach that is model-checking. The framework K is a methodolgy for designing and analysing programming languages. It is used to give semantics to the bytecode instructions of an application and then generate an abstract model of that application. The possible executions of that model are checked for indications of collusion.

These are both static analysis techniques. The proposed project intends to extend the work into the field of dynamic analysis through runtime verification.

Dynamic analysis has been proposed in the paper "Runtime verification meets Android security (3)". In this paper the authors suggest the construction of monitors from LTL to monitor an applications permission requests for protected resources. The technique is similar to the work propsed here but we intend to look for signs of collusion between applications rather than just malicious activity by individual applications.

In the paper "Predictive Runtime Verification of Timed Properties (4)" the authors investigate the use of monitors to predict the violation of properties with real-time constraints. They describe how to use static analysis to give some a priori knowlege of the system being monitored and then to use that prior knowlege to extend events being monitored to discover when the property will be violated in future.

3. Technical Background - Fundamental Domain Principles

3.1 Android security

There are a number of different security mechanisms provided by the Android operating system. Firstly, at its core, is a Linux kernal that enforces the POSIX security features of sandboxing and file access permissions.

The sandboxing feature gives each installed android package a unique userId and a Linux process can only execute code for single userId. This means different packages must run in different processes. Linux uses a memory management unit to seperate processes into their own address spaces. By doing this a process is unable to reference another processes address space so they cannot corrupt each others memory, read, write or in any way communicate with each other directly via memory. They are also unable to reference memory owned by operating system processes that run in protected mode, thus they are unable to insert they own code into an operating systems memory space in order to have it executed in protected mode.

The second feature provided by the Linux kernal is that of file access permissions. Files are associated with an owners userID (UID) and a groupID along with three tuples of read, write, execute permissions. The first tuple is the owners permission, the second is the permissions granted to the group the owner is a member of and the final tuple is the permissions for all other users. Considered settings for these tuples controls all access to files and can be used to prevent unwanted access by unknown users. The Linux kernal extends this security mechanism to hardware drivers by exposing them as pseudo-files. This means the Linux files access rules can be used to quite effectively restict access to resources when configured to do so.

These two features stem from the use of userIDs that are unique per package and the rule that a process can only execute code owned by a single UID. But as is often the case with security there is a built-in way to circumvent this by sharing userIDs. The Linux kernal allows a package to share a UID with another package but it must be specifically requested by both packages and they must be signed by the same party. So for the most part packages are seperated from each other in sandboxes but the shared UID feature does open the door to malicious activity via inter-application communication within the same sandbox.

Further security is provided by a permissions system that is specific to the Android operating system. Access to protected features of the operating system such as the ability to connect to the internet, make calls, use the GPS or camera are restricted to applications that requested permission to use them during the installation of the application.

There are four protection levels for each feature. Normal and dangerous protection level are granted to an application if the user agrees to allow that during installation. The Signature protection level requires the signature of the package being installed to match the signature of the package that created the permission. The signature-or-system level is less restrictive in that it will also allow the package to be installing in the system image. Once an application has been granted permissions they will be denied further permissions at runtime. While this system does attempt to provide some security it suffers from the typical problem that the application is unlikely to function if it does not get permission to use all the features it requests. So if the user decides they do not want to allow a certain permission they will have to abandon installation of the app. When the user is faced with making this decision they often agree to the requests regardless of the dangers simply because they have no alternative but to use the software. This is akin to agreeing the terms of a licence without reading the lengthy document simply because it is inconvienient, unintelligible or it leaves the user with no option if they do not like certain parts of the agreement. Furthermore once the user has agreed permissions and app has been installed, there is no way to tell what the features are being used for. For the most part the use will be benign and any malicious use will be infrequent enough to go unnoticed. This renders permission based security next to useless.

An Android app can provide services to other apps by exposing components for external use. Components have an 'exported' property that can be set to false to prevent apps with different UIDs from using that component. This system has a weaknesses due to apps having a default encapsulation policy that will dictate whether a property is exported when the developer has not given any specific instruction. Components may be exported by default if the developer has not purposefully set the property.

3.2 Inter-application communication

The need to keep applications and their components seperate is key to security and robustness and is therefore a necessary consideration but it is at the same time contrary to the need to provide a feature rich experience to the user. If applications and the components within them have the ability to collaborate with each other then the device becomes a more powerful tool. Therefore Android has overt communication channels specifically for communication between applications. One of these channels is via a system of intents. An intent can be thought of as a command that is sent to a particular receiver or as an event that notifing listeners that something has taken place. If an SMS message is received then an Android component can publish an intent informing listeners of the event. Listeners subscribe to intents and can take action when they receive notification of an event.

In addition to the intent system Android provides three other inter-process communication methods: A remote procedure call system called Binder, Services that have interfaces accessable via Binder and a content provider.

Below Android, the Linux kernal provides methods of communication via sockets and files where the permissions allow them to be referenced my many applications.

3.3 Collusion

Application collusion is where the result applications cooperating with each other achieves a malicious objective. To cooperate requires communication between applications and that communication may be across the overt channels provided the platform such as an intent or a service interface or it might be by exploiting parts of the platform to create a covert channel.

Explain concept of collusion Get this from data analysis book

3.4 Runtime Verification

Explain runtime verification Use transformation picture + lightweight explaination from essay

3.5 xPosed framework

Write about how the exposed framework works

4. First Own Steps

5. Planning

Appendix A: First item

Here is the appendix. You can put anything you want here, with or without descriptions, and can refer to the sections by labelling them. Appendix 5.

References

[1] J. Blasco, T. Chen, I. Muttik, and M. Roggenbach, "Detection of app collusion potential using logic programming," *Journal of Network and Computer Applications*, vol. 105, 06 2017.

[2] I. Asăvoae, N. Nguyen, and M. Roggenbach, *Software Model Checking for Mobile Security – Collusion Detection in*

$\mathbb{K}$

K, pp. 3–25. 06 2018.

[3] A. Bauer, J.-C. Küster, and G. Vegliach, "Runtime verification meets android security," in *NASA Formal Methods Symposium*, pp. 174–180, Springer, 2012.

[4] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa, "Predictive runtime verification of timed properties," *Journal of Systems and Software*, vol. 132, 06 2017.