MOBILE SECURITY

Discussion on two papers:
- Analyzing Inter-Application Communication in Android by Erika Chin Adrienne Porter Felt Kate Greenwood David Wagner from University of California, Berkeley.
- CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities by Long Lu Zhichun Li Zhenyu Wu Wenke Lee Guofei Jiang from Georgia Institute of technology.

Inter process Security (IPC):

There exist IPC in traditional computers as well as in the mobile world. To see the threats and the vulnerabilities, we discuss the common forms of IPC that exist nowadays:

- Shared Memory: It is the basis of all the forms of IPC available. In order to have a communication between two apps we need to exchange data and shared memory comes into the picture. We can say it is the natural way of communicating between the application where in one app create a piece of memory for writing and reading the data and other app can access the data through that piece of memory.
- Pipes: It is another way of communication. Question here is who allocates space for applications? Two apps decide on what data to be shared and follow the semantics between them but it is OS who decides how much space to be allocated using pipes.
- Socket: IPC in some sense resembles network communication whether apps are talking on the same machine or the different one. It is a common mechanism that also plays important role in iOS and Android too.
- RPC (Remote Procedure call): Sharing of data between the two processes in which one function is capable to call the function in another process and communicate. One process can wrap up the arguments and after wrapping
can send it to another app which can execute and return the result. So it deals with function calls.

Security Problems when two program Communicate:-

- Unauthorized access to data: - suppose, one app allocates shared memory and do not enforce authorization as in which app can access what, and third malign app is able to access that can cause unauthorized access leading to vulnerabilities. So proper protection mechanism is required. For example, while writing kernel drivers, if kernel doesn’t perform enough checks, user land code can do a lot of damage for instance, cases of jail breaking and rooting concepts in iOS and Android respectively.

- Mobile Security (Android): -
  1. Component: - it is the essential component or building block for android application. Program paradigm for android is such that it lets you package the code in component. What is the use for it?
     - Code reusability: -for example, the case of contact manager, it can be used by different apps. It is kind of inter component communication.
     - Also, for recent mobile operating system, you are not allowed to work on the low level details of kernel. As a developer you cannot access IPC components for OS example: windows 8, windows 8.1 in phones. To overcome it, OS provides low level interfaces to embed into high level interfaces. That is termed as Component level communication.

So, Apps do not have API for shared memory, but provides interface for processes to communicate over the secure interface or we can say by enforcing security. Example in iOS, if we try opening a web page, request goes in the form of string. OS dispatches request to which app can handle this URI request.

So, here the finding is that App cannot make a low level change. They should not be worried about, how much memory to allocate in shared memory, where to allocate. It is not an app concern. OS handles it by providing the interface thus enforcing the security.

2. Intent in Android: -
   - Example: app managing the contact information. If app encounters a request, homepage entry is made. User clicks on the link means
he want to open that. So, App either handles the request or delegate it to the other app with the help of the intent.

- Now, how does OS decide where to send these request? It is a security flaw which can be exploited. For example: - There are two app on android phone, which opens a pdf file. Now two files can handle that. So when the adobe reader is used, which app uses it is an issue. Design of android enables user to decide which app to proceed as in pop up can be displayed by the OS. It is a design flaw. Third party app can change the icon that comes on the pop up with the malicious app icon which resembles the original app and thus attacker can **hijack the intent/ app activities**.

**What message first paper delivers?**

- First Paper talks about the different attacks by examining the android application interaction and identifying the security risks. But data set paper uses to demonstrate that is not quite large as discussed in the class. Example, it uses 100 apps only. They are doing most of the work manually so we suppose that they did less due to the human factor. There was no automatic detection method.
- Also, this paper tells what the problem is, but does not tell about the solution which the second paper does.
- Paper delivers a message that even though the mistakes are easy to avoid, but average developer don’t know how to do that. There are mechanism to protect it but we as average developer are not aware of them.

**Discussion on second paper:** -

Second paper deals with analyzing the code to automatically vet Android apps for component hijacking vulnerabilities. They model these vulnerabilities from a data-flow analysis perspective.

CHEX: - Why do we need new tools to analyze the mobile apps when there exist traditional ones to analyze the traditional systems?

- Answer is New Challenges. Apps are developed using java and compiled to 
apk. It is a change from traditional java code compiling into java byte code.
- Also, it limit the mobile devices in terms of memory, computing speed and size of the files. Yes, small file sizes executes faster on mobile CPU, but what challenges does it pose.
CHALLENGES: -

1. Small file sizes result in the removing of some important information that might need in analyzing the code. Let us see the flow diagram to get what it means.

![Flow Diagram]

2. Programming paradigm is completely different from the normal java code. It uses event driven execution. There are multiple entry points in android applications not one like main in the traditional code. They have on create, on start method and many more. Also, there exist asynchronous start, power limitations which make a big challenge. Mobile applications are written in collection of event handlers. Without any blocking, OS can return back to the app.

So, the author’s method of static analyzing the code flow was introduced.

Static Analysis: -

- Built call graphs: - To build this, we need to know that where the start of the program is and in android there are many placed to start an application as discussed previously. Call graphs are most basic building block of program analysis. Possible call traces starting from one point to where the program can possibly flow or branch can occur and which path is taken in the end. It can all be done without executing the program.

- Data Flow analysis: - In the paper, they present new data flow analysis. Due to security problem data from 3rd party app, data flows after reaching path to some place reading the content from place where it is not intended to.
CHALLENGES ADDRESSED: - Very Concise entry point discovery algorithm. It is a heuristic based algorithm without human effort. As the evaluation states in the paper, there were 50 entry points observed per average app. So they were not just on create and on start methods.

3 general categories of entry points: -

- Life cycle related: - Event happen, OS reacts and send the response. Managing of event handlers. For example, On start for OS to start, to suspend, OS sends notification. On resume etc.
- Asynchronous execution: - create a new thread off the main thread to execute asynchronous without affecting main thread. So, now the new entry point is the thread created.
- Other miscellaneous entry points are also discussed in paper.

Now, once Entry point is identified, one can analyze data flow.

SINGLE STATIC ASSIGNMENT FORM: -

In this form you create the program into IR. Paper uses the IR in SSA form. By converting the code into SSA form, code can be analyzed in a much more easy way. Most of the common compiler do use SSA for code analysis and code checking for example GCC.

Let us see how it does operation using simple program example: -
Now, we have to know what type of data is flowing through the code. It is termed as inter component data analysis. This was the limitation of the first paper.

This approach recovers all possible different components whether they are inter, overlapped component. It is basically determining where the data could possibly go.

**IT has some FALSE CASES or we can say detections. SO we can say the analysis is SOUND but not complete.** It is because goal of static analysis is code quality and code detection which shows false alerts in some cases. In reality, its okay to have some false alerts. Also, permission also imposes false alerts in the process. Component and other things require permissions which the author didn’t consider.

Even malicious code could evade the static analysis due to obfuscations.

**HOW WAS THE PROCESS CARRIED OUT?**

- Data flow relativeness analysis. If no two apps related to each other, then they don’t have to share the data. For example, if a class A and B are there
and class B uses some variables in class A, so we say that there is a data flow from A to B, asking B to carry out some work.

After Dataflow analysis, they identified the vulnerabilities. For instance, they found vulnerabilities in one of the famous games which accidently exposes the interface for the game. Any malicious attacker can access that and change the score of the game. This attack was made on the principle, that one component can receive URL and other component can receive the data. So if attacker can get hold of that, it can make network proxy without the permission.

Also, the case of internal sensitive data (data flow)

- Data will go out of the application. for example, another Instant Messaging app leaked credentials to the public storage. So malicious attacker can enable the write password to the SD card and enable the most serious security flaw. This is known as hijacking enabling flaws.

NOTE: - Analysis doesn’t consider about the algorithm app uses, only data flow analysis is performed. So static analysis is used.

Class Concluded.