Lecture Notes: 10/21/13

Paper1: Analyzing Inter-Application Communication in Android
IPC(Inter Process Communication) Security

Threats are discussed for mobile devices as well as for personal systems.

Mechanisms of IPC:
1. Pipes
2. Shared memory- basis of all the forms. The method for the processes to communicate with each other is that to allocate memory chunk and share it with another one.
3. Socket

The concept of IPCs resembles network communication. Through this different programs can communicate whether they are on the same machine or on different machines.
So socket is another communication mechanism for IPC.

So socket plays an important role for IPCs.

In terms of security, the problems when two programs talk to each other are:
1. Unauthorized access to data- If the communication is limited to only two parties, then the problem will be if third party can read and write the data.
2. It often causes problem if someone is writing code in kernel level. When every process is running in OS and security switching happens, if the kernel codes wants to share memory with user programs then security threats can happen.
3. There are limited ways for breaking into OS called rooting or jail-breaking the mobile OS. Most techniques rely on design flaws so that user code can be injected into the kernel code.

Background in mobile OS

Some important concept in Android:
1. Component- is essential for any android apps. They are the building blocks of any android app. This is because of following reasons:
   a) Code usability: if you have a component that does a very specific thing i.e. lookup contacts on the phone but it can also be used by the other apps. This is one form of IPC or inters component communication. In recent application development in mobile devices they reshape the old concept of old IPC. The user level application does not have the capability to do any of these. The low level concept is not exposed to user level application. The reason was that is due to security on one hand they all to use apps, second is the security impact is negative. That’s one reason the mobile application does not use IPC mechanism.

   So as a developer, one application will talk to each other by a concept called component level communication. The OS provide low level interfaces. So in order to allowing two programs to talk to each other when one program wants to talk to each other but the OS provide the support of conveying very short
message to each other, but that provides the need and guarantees security. Like open a webpage by iOS, now the app does not know how to do this, they do this by issuing a request by issuing a string (looks like URI) and then OS dispatches request and select which app can handles this. So there is no need to allocate the memory and etc. This simple design can also cause security problem.

There is another IPC called RPC (Remote Procedure Calls). RPC shares data among processes and it causes the programs to use functions that are declared in another process.

The same concept is followed in Android. Different apps can talk to each other but in a strained environment. They can talk to each other through –intent. For e.g. if the user is looking through the contacts and then user touches the browser then the previous app can open the browser(if it can do so) or it can delegate to another app. In Android this thing happens is that if there is already an app that can do this thing then that task is delegated to another app through the use of intent. The context manager will sent an intent and send it to the OS and then OS will decide where to sent this request. There are some problems associated with this:

a) Unauthorized Intent Receipt

Suppose we open a pdf file in Android and there are two applications, which support that file. There will be a pop up message for the user to select which one of the app he wants to open it. The app that the user touches will send a request to the OS to allow it to access the pdf.

So there was a design flaw in this pop up message. There is an icon in front of this app that handles this task. The third party can change the icon in front of its name and can change it to adobe reader. Also it changes the icon of the original one. If user wants to use adobe reader so he touches the fake adobe reader.

One imp message this paper delivers is that- the mistakes can be easy to avoid but then a developer does not know how to prevent them. Average developer does not know these features. So this is because of armature and inexperienced developer. They just try to make money and give up quality, assurance etc.

How do we make sure that these apps do not get to app stores is a critical issue.

Dis-advantage of this paper:

1. They rely on human analysis that limits the scalability and the size of then data set.
2. The data center of this paper is not large.

Second Paper: CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities

The second paper is about the static analysis of code.
It can handle not the small volume but also the big volume of apps. A very large number of apps come to apple and android store. There are all new apps, so the detection and analysis method has to solve the scalability issues so that there is no threat. The second paper we used a lot of more than 5k apps and depending on human verification. The first paper does not have automatic method.

**Background**

Static analyzing code is not a new area so why we need a new tool. As there are some unique challenges. Android apps are written in Java. When they are compiled and packages are called apks.

![Diagram of Java and Apk relationship]

This is different from the java code, as mobile devices have limited storage, limited computation power. This type of byte code has smaller file size and faster when run in mobiles. So the challenge is if you have to make file smaller make it more secure so this is the problem.

![Diagram of Java and Apk relationship]

**Challenges:**

1. So the main challenge is to reverse this process and how to recover this info as much as possible.
2. Security challenge- writing an app is different than writing a java program. These mobile devices follow an event driven module. The app itself does not have say in it life cycle, execution, sequences, order, etc.

In mobile devices there is no single entry point but there are multiple. This causes power constraints because OS have to have immediate control, as if the battery is running low than OS has to kill the low propriety apps. Due to this mobile apps are written in the ways collection of event handlers.

**Static analysis we first have to build:**

1. **Call graph**
It is the most basic of building block of any program method. It tells about the call traces, branches it might go through, execution, function it calls. It tries to model the device without executing the program.

We have very precise entry point discovery. In android app there are various places the graph can start. In the previous paper the human effort tell where to start the program. Here we have a heuristic based algorithm without any human interference.

2. **Data flow analysis:**
   Gives understanding of the app that what variables can do and what shall be the outcome. This paper follows data flow analysis. We want to know amount of data that possibly come from third party app can come into this app and reach at some place that they are not supposed to reach. So this information is tell by this data flow analysis.

**There are the categories of entry points:**

1. **Life cycle entry points** - any event happens then the event handler will notify the OS about the event. There is large number of event handlers like event handle will tell the OS when mobile starts, etc.

2. **Entry points that have to do asynchronous execution**
   If the malware wants to create new threat from the main thread so that it can handle something asynchronously but not blocking the execution of the main thread. At this points the starting points will be that of new thread.
   All these complication when come together its very diff to tell where is the entry points.

Once all the entry points are identified when we knw where to start. It is very imp for static analysis i.e. call graph.

![Diagram](image)

**Intermediate Representation (IR)** – they have all these necessary information inferred but not as friendly java language. Here it is not relying on human analysis. There is a concept called single static assignment- SSA, which is a form that you convert a program into that every variable is assigned once. The IR that we use is in SSA form. Analyzing code is lot easier than regular code representation. SSA is used by GCC, LLVM. They use this for code checking. One advantage is every variable is assigned once. For e.g.
Example Program:

```c
int main()
{
    int a;
    a=0;
    printf(a);
    a=b
    -
    -
    -
}
```

SSA Conversion:

```c
int main()
{
    int a;
    a1 = 0; //it is already defined so we rename it by a1
    printf(a1);
    a2=b;
    -
    -
    -
}
```

This can be complicated by message function calls, functions, etc.

Having in SSA form we can track the origin but that is not enough. For that data flow analysis is required.

Data flow analysis solves- inter component data analysis:
We want to recover all the possible interactions. We want to be sure that were this data could possibly go. For static analysis the disadvantage is that it causes false positives. But the good point is that it never misses anything.
This analysis is sound but not complete that causes false positive.

But in reality it is tolerable to have false positive. Static analysis is rarely seen on malicious software. We cannot detect if a program is doing malicious behaviors. The static analysis can be fooled that a program path is not feasible. The malicious code can easily pass the security check of static analysis.

Analysis among different component is done by: This is by doing some kind of data flow relating analysis. If one component is always executed after another so it will share the
data flow paths. If two components are not related to each other then the data flow paths cannot be change. For e.g. funcA calls funcB so there is a data flow between them.

With analysis results the vulnerability we are looking for: The external data overwrites: the external data overwrites the internal data. For e.g. The highest score of a game can be changed by the third app.

Another type of data flow is IM- instant messaging apps that accidently release code. This is called highjack-enabling flows.