Today Discussion:

Today's discussion involved the discussion for three topics:

1) Project Topic discussion
2) Few clarification regarding deputy
3) Model curing code (Not covered today)

Project discussion:

There are 3 project options for the students. Detailed information about the project is available on the class website.

The projects will involve Coding + evaluation criteria+ demo. The distribution for project will be 60% report + 40% Code/Product

1) **User-driven disclosure for Android** - First of them is a bit ambitious project– it lead to publication. Scheduled publication is due end of February. There a 2 major contribution to this project. So for this project, one can do either of them by the end of semester. Inspired by two papers:-

a) The first one is System call taint droid paper. It is modified version of dalvik virtual machine which is essentially run on android. What it does is run time taint tracking of data inside the dalvik machine. As Prof Sekar discussed in class. It maintains for every byte of memory inside a virtual memory a Tag or another byte. The value of the other byte tells you about the source of data. So if the program reads the string of input from the network,
It looks like Array of byte that was read from network, so taint droid will maintain the shadow array of bytes. This data is not accessible by the program, it is the metadata that droid taint tracks. Taint droid can keep track of information way more than just byte of data. Suppose if application is trying to send GPS information, it can keep the track of it.

Why the taint droid like application was developed was to answer questions like what type of private information is sent by the application over the network? If application query your GPS location, we can log that the information has been transferred to specific place. Similarly can be audio-video data also.

- Taint droid tool is not an enforcement tool. It is a measurement tool, to know what app is doing with personal data. So the Goal of project-1 is to turn taint android into enforcement tool.
- So we can have the policy like allow to read the GPS information, but not send it over the network.

b) Second part describes it may send it where it is described by the second paper. In many apps today we get a permission access notification like, need access to photos, GPS location etc. Suppose we
have app which needs location, like restaurant locator apps. So this part involves significant hacking on virtual machine, some way to notify VM the app is allowed to get standard enforcement tools. Then implement the buttons (OS managed).

The project involves the following components:

- Implementing the OS-managed UI components.
- Implementing policy-enforcement within Taint Droid.
- Connecting the OS-managed UI elements to Taint Droid’s policy enforcement.
- Writing a small sample application that uses the new system.

The goal of this project is to produce a publication, ideally in time for the USENIX Security 2014 deadline (February 27, 2014) as told before. You do not have to complete the entire project by the end of the semester. You only need to complete one of the major components by the end of the semester (i.e. the Taint Droid policy-enforcement system or the OS-managed UI components).

We can make use of taint droid for jelly beans.
2) Memorable but incommunicable passwords:

- This project is inspired by recent paper. It is actually a Weird paper at Stanford.
- PhorceField: A Phish-Proof Password Ceremony, Michael Hart, Claude Castille, Manoj Harpalani, Jonathan Toohill, and Rob Johnson.
- They come up with method for authenticating user, but want to be secure against rubber hose crypt analysis. It means beating up the person until they tell you the password, who knows the password. Which seems impossible. Idea was, the way user will login is that user would have trained on game and would have skill in game that no one would have as game would be customized to that user.
- This project envisions extending the ideas in the PhorceField paper to see if it is possible to create graphical passwords that users can remember and use to login, given a prompt, but cannot easily communicate to another person. The PhorceField paper demonstrated that users cannot easily communicate their password when using a search-based interface, but it is not clear if they cannot communicate their password using any interface.

Project is to collect a corpus of images of different types. They could be concrete nouns like Landscape, faces etc. Create website to do a user study it. Recruit students as participants. Study will involve:

1) Can people remember them, can they describe them to other people.
2) Small password system on the bases of classification. Some users get faces, some get landscape as classification of passwords.
3) Then we have to test it every day or in a week time’s or after some day how well they remember there password.
4) How long or how easy is it to communicate the password to the fisher by victim? Try think ways so that you give as much power as possible to fisher other than simply pulling the password images from user’s computer and mailing to them.
5) Point here to discover the system that can authenticate people without telling directly the description of the password.

6) Between groups comparison, you take your subjects who were given passwords of image type 1 and subjects with image type 2. And you will measure the success rate after logging into the system after a week. And then we have to say, whether these success rates are successfully distinguishable. Professor can help on this part of the statistics.

7) Participant 12 per type of image, 24 for two.

8) Compute average distribution of time, mean of the time and perform the statistical test.

3) **Android Fuzzing Tool**

- This project is much Shorter and have vague description.
- We have to build a fuzzer for android API. A fuzzer is tool for testing other piece of software, it generates input randomly and feeds to other software and observes how it handles that. Popular technique for testing network server, were you build only client that generate the random bytes and send it over network and see whether it crashes or how it behaves on those condition. We monitor that.
- Example: Fuzzer for Linux kernel, trinity. Link to this fuzzer is provided on the project description page for detailed description. Randomizing the input won’t cause many problems. There could be a bug deep within the input processing/ or there is poor code coverage.
- Write a tool that uses fuzzing to test the Android APIs. Note your tool is not intended to fuzz android applications, it is intended to fuzz the Android APIs/OS.
- You may write a fuzz tester for a subset of the Android APIs. You should clearly define the subset you are testing and the security properties you intend to test with your fuzzer.
- You may also use the Java Reflection API, as in the JavaFuzz tool.
- You should implement the fuzzer and run it on an android system. Investigate any crashes or bugs provoked by your fuzzer. Report any real bugs you find to the Android Developers. Try to obtain feedback from developers confirming your bugs.
Clarification on Deputy Concepts: -

Deputy was a scary paper where we talked about dependent types system. In deputy we can write annotations like:

myfunc (int * COUNT(n) p , int n)
{
    //deputy might insert assertion like below if we have P[i] like assignement
    assert(0<=i && i <n);
    P[i] =....
}

The reason we could write code like this and link it to the code that does not run through deputy is because all the extra code that deputy inserts on the fly is simply assertions like described above and assertion like this do not change the state of program. They either terminate the program or leave the program in the exact same state. It is different from Jones and Kelly's because it has potential issues when you link Jones and Kelly's compiled code and to non-compiled code because it has extra metadata about the bounds. And if you link that code with that updated metadata, things might not work. But here all the inserted code here either terminate the program or leave it in that state. One clarification involved this concept.

So, Deputy has two phase:

1) Insert assertion/ annotations can make the program safe.
2) Then optimizing it by eliminating the assertion by analyzing the loop and by understanding the bounds. So it is straight up compiler optimization stuff. This optimization is not related to security it is just eliminating the stuff.

Last thing to clarify is regarding the updating of these variables.

If we have a parallel assignment like code below: -

myfunc(int * COUNT(n) p , int n)
{

p=p+1;
n=n-1;
}

Now, here we know `p: ptr (n) int` is the condition which says p is the pointer n int, deputy would like to prove that the new value of p and n, p+1 and n-1 would also be true for this condition (satisfy this constraints). That means it holds after the parallel assignment is done.

```
p+1: ptr (n-1) int    // this should be true
p: ptr (p, p+n) int  // when we have pointer in integers, it remembers like this.
```

Low        High    // that's what Count in that function means

So below we see how it handles annotation.

Class Concluded