The lecture covers the final project and few topics. The detailed discussion about the projects are given on the link below: http://www.cs.stonybrook.edu/~cse509/projects.html

There are 3 project options for the students. The demo is 40% and report is 60% of the grade.

**Project1: User-driven disclosure for Android**

It’s a very ambitious project and will lead to a publication. Two major components that needs to be done. It is inspired by two papers.
1. The first one is based on TaintDroid. https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Enck.pdf
   It is a modified version of dalvik virtual machine that is used on android. It does a run time taint tracking of data inside the virtual machine. It maintains for every byte of memory inside a virtual machine, a Tag of another byte. The value of the other byte tells you about the source of data. So if the program reads a string of input from the network. TaintDroid makes a shadow list.

```
Read from network ➔
Shadow list ➔
```

Program reads a string of input from the network, so it is an array of byte that is read from the network. Then taint droid will maintain the shadow array of bytes. This array of bytes is not accessible by the program, it is the metadata that droid taint tracks. Taint droid can keep track of more information than that. For example: if application wants to read the GPS location, which is considered to be private data. So if the application reads a string of input from the network. TaintDroid makes a shadow list.

```
Read from network ➔
Shadow list ➔
```

TaintDroid answers question like what kind of personal information can this application send over the network. If the application queries our GPS location, we can log that the information has been transferred to specific place. Similarly can be audio-video data also.
It answers questions like what kind of info can be sending in the network.

Note:
Taint droid tool is not allowed to do enforcement. It is a measurement tool, to know what app is doing with personal data. So in project-1 is to **turn taint android into enforcement tool**. So we can have a policy that can allow reading the GPS information, but not sending it over the network.

In android if we install an app then it will ask for some information.

Second part describes it may send it where it is described by the second. 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).

There was a paper on any app there is a button that initiates action. Like the button below shows recording button. It initiates recording. The idea behind this paper, this button is drawn by the app itself. It is managed by trusted OS then we can grant no privileges to the app for reading and writing. But when we click this button the OS knows that the user clicked this button and the button has microphone drawn on it. Then we will give the app permission to read microphone data. In iOS the case is something different. The OS does not know that there is a microphone button on the app. When you click on the rec button then the app will take permission from OS and it will pop up a message saying ‘Do you want the app to grant permission to record something?’ Then we click ok and the app get.

This project is to stick these two projects together.
Suppose there is an app that gives the location of all the restaurants nearby. So there is a GPS button that is managed by the OS and not by the application. It will grant the app:
1. Privilege to read the GPS
2. Can write to the network
3. The app is running inside the VM, which does not allow any bit of information to go out of network. But here we can that VM should allow.
So this requires hacking of VM and allows telling the VM that this app can get around with normal enforcement rules.

So there are two parts of it:
   a) If you modify the taint droid system to do enforcement.
   b) You can build infrastructure where OS receives input from the user and then makes dummy calls to modified VM.

**Project 2: Memorable but incommunicable passwords**

This is also inspired by some papers.

This is based on the research by some guys from Stanford. They were trying to come up with a method to authenticate users. They want it to be secure against rubber host crypt analysis. This means that beating up a person until they tell you the password. They want the password to be secure against the person who is trying to get the password by trying to beat the persons. They came up with an idea that the way the user login i.e. the user is trained on a game and it will have a skill in the game that no one will have.

There was another paper which was published in which we are trying to come up with another password system that if you were able to visit the right website then you can enter your password very easily but if you visit a wrong website then you cannot enter your password. You do not know how to do it. The right website is given access to a small set of 12 images that is specific to you. So if you see your images then you can enter your password. But if we visit a wrong website then that website does not have access to the images so it will guess randomly from a 100,000 image and show from them. So if you do not find your images so you will not use that websites.

So here you will use images for storing password. Here the concept is it is easy to select the images but difficult to describe to someone else. For example, faces. Faces can be great to build the password. If someone asks us describe your face or describe the landscape. So that would be difficult and hard to describe. So the project will be to collect the large number of images like random images and then we write a small website and recruit each other as participants. There are two things that we can track in this and can describe to other people easily. We set a small password system for these images. Your subject of experiment will have to select a set of images. Some of the users get faces, some get landscapes and then we have to ask them after 2 weeks and 3 weeks whether they remember their passwords or not.
We also have to setup a system to test how comfortably people can communicate their passwords to other people. You also have to set up how easily people can describe to a fisher. I will give a fisher as much power as compared to giving the image. The user cannot see the real image but we have to describe the image. You have to test whether the set of images can be easily communicable or not and how long it takes for the victim to describe his password to the fisher. If we decide to participate then you will get extra credit (2 lecture notes). Then see how quickly we can compromise the security of our own password. So that’s the idea behind the project. Back to the paper. The point will be to have a system to prevent the fishing attacks. The upper limit for this project is 70 participants.

Here we have to draw the curves (distribution) for the person who has used concrete nouns v/s people who have used abstract nouns for selecting their images. Every project will get 12 participants and a person can take part in more than 2 teams.

**Project 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 another piece of software. It works like it generates input randomly and then feeds to other software and observes how it handles that particular input. This is a 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.

Example of fuzzers: Fuzzer for Linux kernel called trinity. Link to this fuzzer is provided on the project description page for detailed description. Trinity makes random system calls with random arguments and see whether kernel crashes or not. The thing is that you run it inside a virtual machine and we record what we are doing and it reports when it crashes. There could be a bug deep within the input processing/ or there is poor code coverage. This is the problem behind random fuzzing.

In this project our main goal is to write a tool that uses fuzzing to test android APIs. Our tool is not intended to fuzz android applications (they already exist); 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:  

In deputy paper we talked about the dependent type system. In deputy we can write annotations like:

```c
int myfunc(int *COUNT(n) p,int n)
{
    assert(0<= i && i< n)  // this insertion has been inserted
    p[i] = ....
}
```

Let's talk about compatibility. 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 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 where the program counter points to. That 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. There is some code that John’s and Kelly was inserting that update the current state of the program. But here all the inserted code here either you have to terminate the program or leave it in exact same state. The binary code is not effectively unchanged.

The other thing was that we talked about trying to eliminate these assertions. That is compiler optimization stuff.

So there are two phases of the deputy:

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**.

Suppose we have a parallel assignment:-

```c
Int myfunc(int *COUNT(n) p,int n)
{
    p = p+1;
    n = n-1;
```
p is a pointer to n ints.
P : ptr(n) int

What deputy would like to prove is that the new value of p and n, would also be true for this condition i.e. satisfy this constraints. That means it holds after the parallel assignment. So we have to prove \( \rightarrow \) p+1 : ptr (n-1) int

\[
p+1 : \text{ptr} (n-1) \text{ int}
\]

\[
p : \text{ptr}(p, p+n) \text{ int} \quad \text{high}
\]

\[
p+1 : \text{ptr}(p+1, p+n) \text{ int} \quad \text{low}
\]

\[
p+1 : \text{ptr}(p+1, p+1 + n-1) \text{ int}
\]

RULE:
P : ptr(a, b) int a<=c<=d<=b
P : ptr(c, d) int

so using this rule if eq 1 and 2\textsuperscript{nd} are true than 3\textsuperscript{rd} is also true,
P+1: \text{ptr}(p, p+n) \text{ int} \quad \ldots(1)

\[
p+1 : \text{ptr}(p+1, p+n) \text{ int} \quad \ldots(2)
\]

\[
p+1 : \text{ptr}(p+1, p+1 + n-1) \text{ int} \quad \ldots(3)
\]

Another Rule:
P : ptr(a, b) int k:int
P+k : ptr(a, b) int

So with this I get:

Annotation of p:
P:ptr (p, p+n) int \quad 1:int
p+1: ptr(p, p+n) int
p+1: ptr(p+1, p+n) int
p+1: ptr(p+1, p+1 + n-1) int

Class Concluded.