CSE 484 / CSE M 584 (Spring 2012)

Asymmetric Cryptography

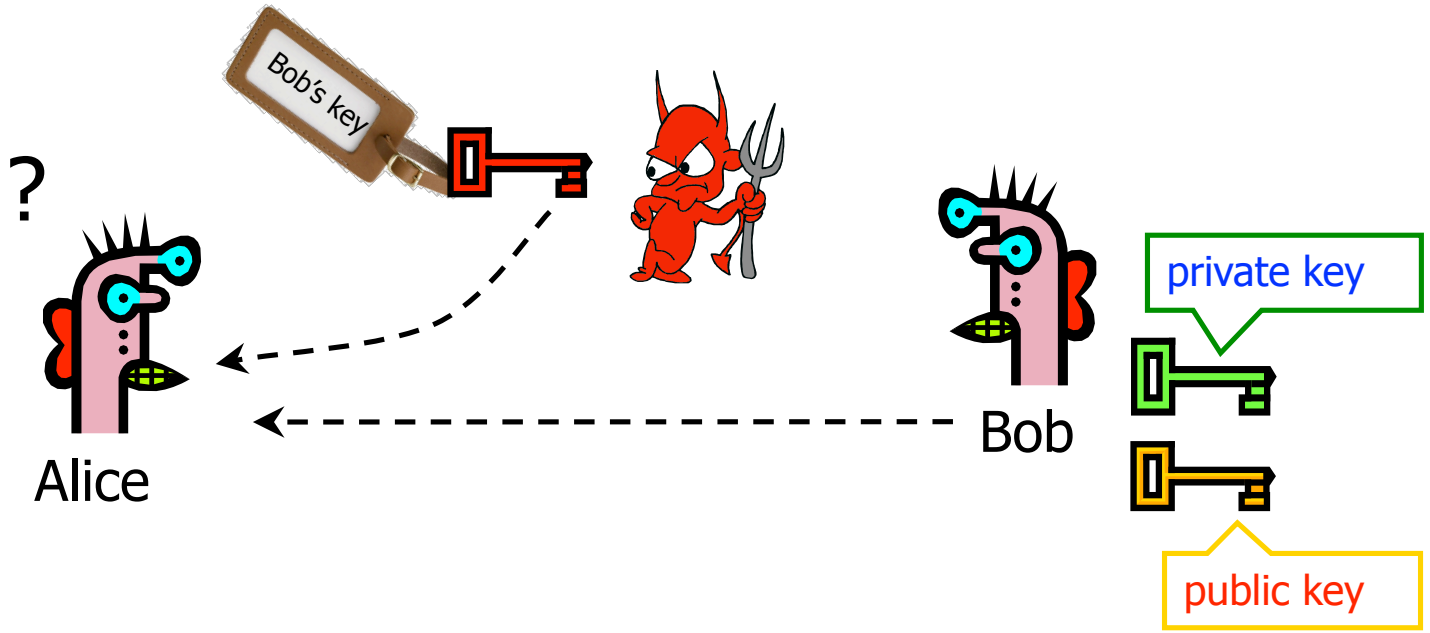
Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Goals for Today

- Asymmetric Cryptography

Authenticity of Public Keys



Problem: How does Alice know that the public key she received is really Bob's public key?

Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{CA}("Bob", PK_B)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA's certificate for the public key (offline)
  - Every computer is pre-configured with CA's public key

Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority
  - For example, Verisign
  - Everybody must know the public key for verifying root authority's signatures
- Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  - Instead of a single certificate, use a certificate chain
    - $sig_{Verisign}(\text{"AnotherCA"}, PK_{AnotherCA})$, $sig_{AnotherCA}(\text{"Alice"}, PK_A)$
  - What happens if root authority is ever compromised?

Many Challenges

Spoofing URLs With Unicode

Posted by timothy **on Mon May 27, '02 09:48 PM**
from the **there-is-a-problem-with-this-certificate** dept.

Embedded Geek writes:

> "Scientific American has an interesting article about how a pair of students at the Technion-Israel Institute of Technology registered "microsoft.com" with Verisign, using the Russian Cyrillic letters "c" and "o". Even though it is a completely different domain, the two display identically (the article uses the term "homograph"). The work was done for a paper in the **Communications of the ACM** (the paper itself is not online). The article characterizes attacks using this spoof as "scary, if not entirely probable," assuming that a hacker would have to first take over a page at another site. I disagree: sending out a mail message with the URL waiting to be clicked ("Bill Gates will send you ten dollars!") is just one alternate technique. While security problems with Unicode have been noted here before, this might be a new twist."

http://it.slashdot.org/story/08/12/30/1655234/CCC-Create-a-Rogue-CA-Certificate
http://www.win.tue.nl/hashclash/rogue-ca/

Many Challenges

CCC Create a Rogue CA Certificate

Posted by CmdrTaco on Tue Dec 30, 2008 12:14 PM
from the **they-even-faked-this-dept** dept.

t3rmin4t0r writes

> "Just when you were breathing easy about Kaminsky, DNS and the word hijacking, by repeating the word SSL in your head, the hackers at CCC were busy at work making a hash of SSL certificate security. Here's the scoop on how they set up their own rogue CA, by (from what I can figure) reversing the hash and engineering a collision up in MD5 space. Until now, MD5 collisions have been ignored because nobody would put in that much effort to create a useful dummy file, but a CA certificate for phishing seems juicy enough to be fodder for the botnets now."

DigiNotar Hacked by Black.Spook and Iranian Hackers

Posted by Mikko @ 09:05 GMT | Comments

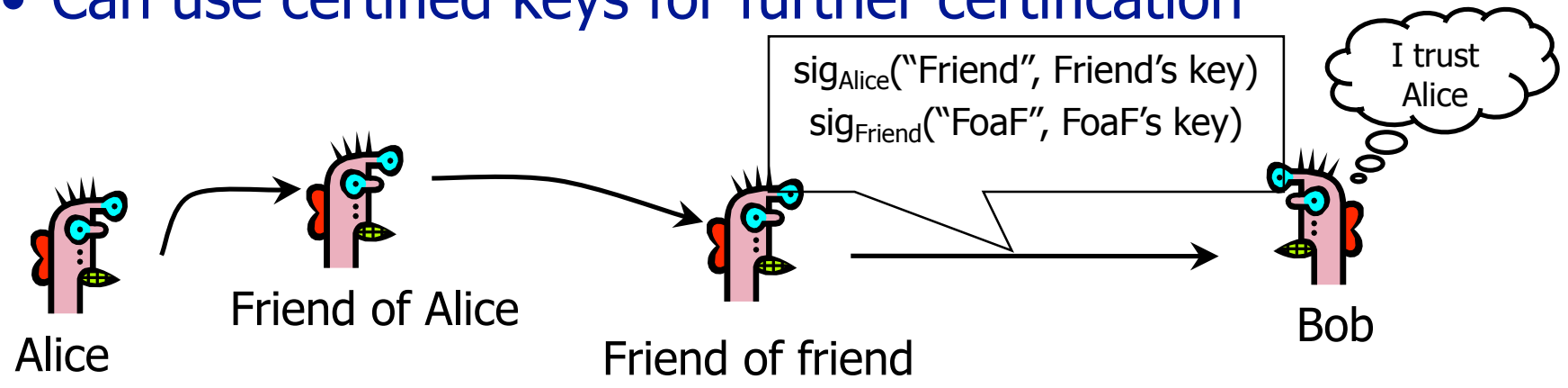
DigiNotar is a Dutch Certificate Authority. They sell SSL certificates.

Somehow, somebody managed to get a rogue SSL certificate from them on **July 10th, 2011**. This certificate was issued for domain name **.google.com**.

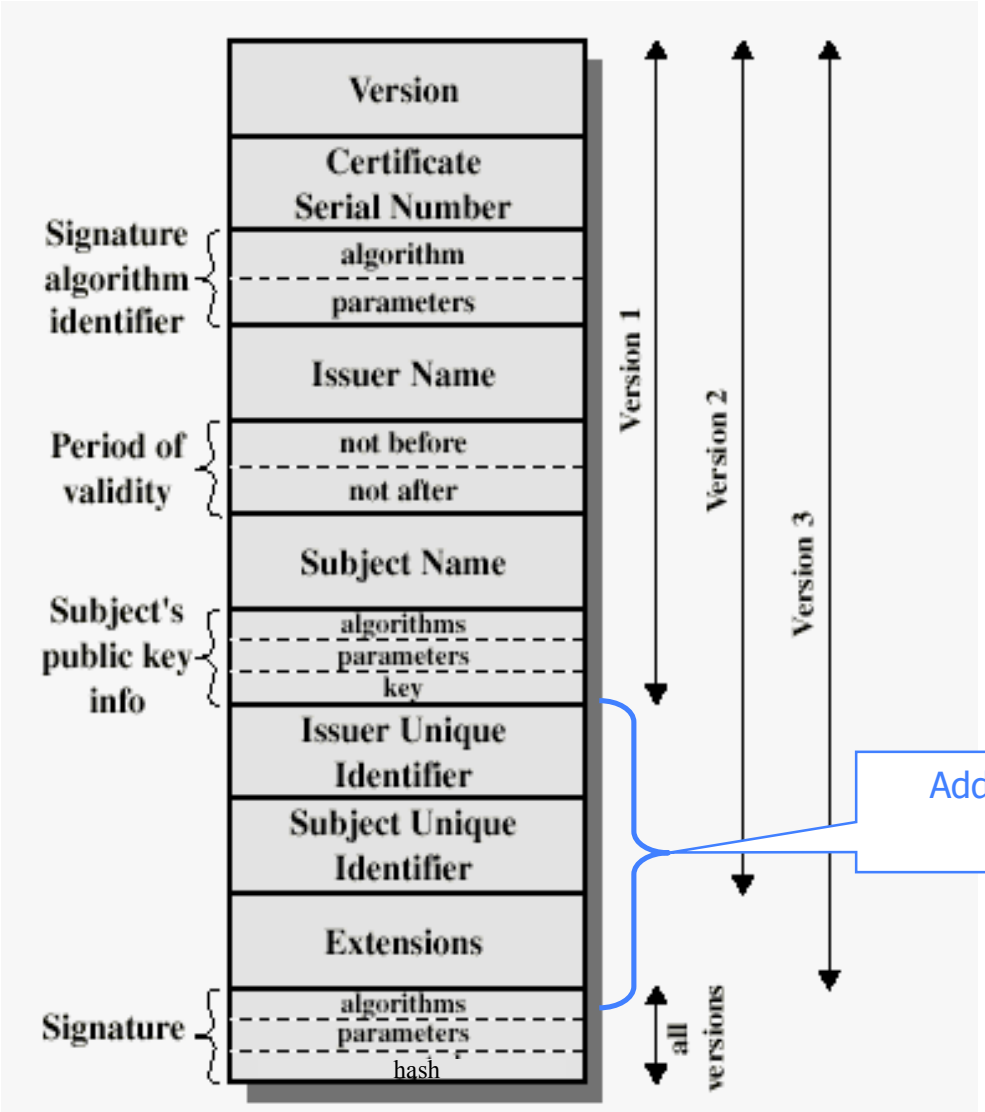
What can you do with such a certificate? Well, you can impersonate Google — assuming you can first reroute Internet traffic for google.com to you. This is something that can be done by a government or by a rogue ISP. Such a reroute would only affect users within that country or under that ISP.

Alternative: "Web of Trust"

- Used in PGP (Pretty Good Privacy)
- Instead of a single root certificate authority, each person has a set of keys they "trust"
  - If public-key certificate is signed by one of the "trusted" keys, the public key contained in it will be deemed valid
- Trust can be transitive
  - Can use certified keys for further certification



X.509 Certificate



Added in X.509 versions 2 and 3 to address usability and security problems

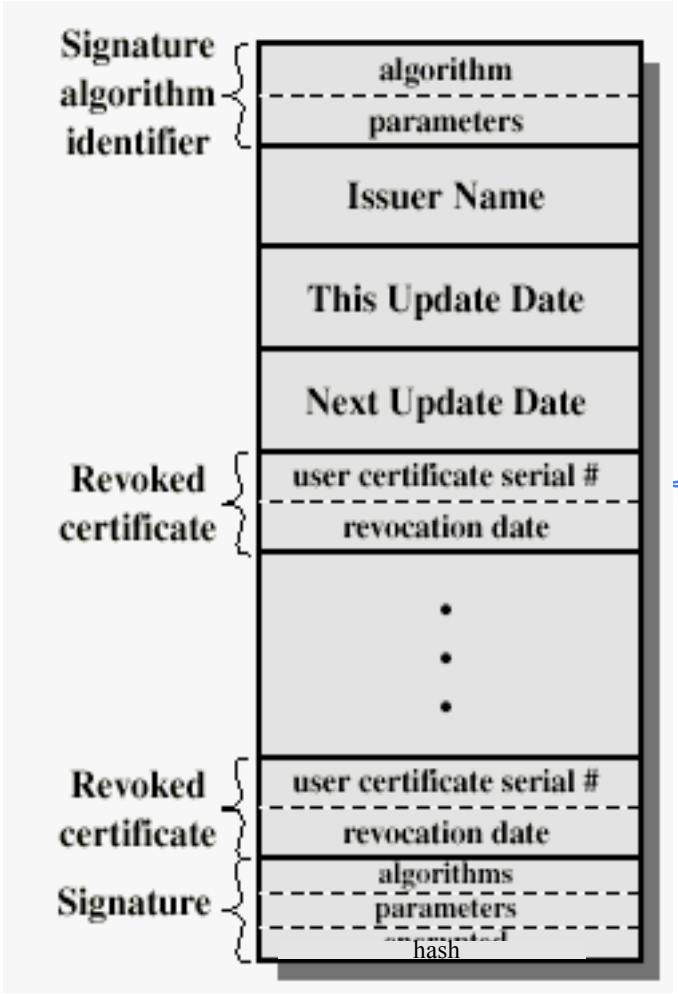
Certificate Revocation

- Revocation is very important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  - CA's private key has been compromised!
- Expiration is a form of revocation, too
  - Many deployed systems don't bother with revocation
  - Re-issuance of certificates is a big revenue source for certificate authorities

Certificate Revocation Mechanisms

- Online revocation service
  - When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    - Like a merchant dialing up the credit card processor
- Certificate revocation list (CRL)
  - CA periodically issues a signed list of revoked certificates
    - Credit card companies used to issue thick books of canceled credit card numbers
  - Can issue a "delta CRL" containing only updates

X.509 Certificate Revocation List

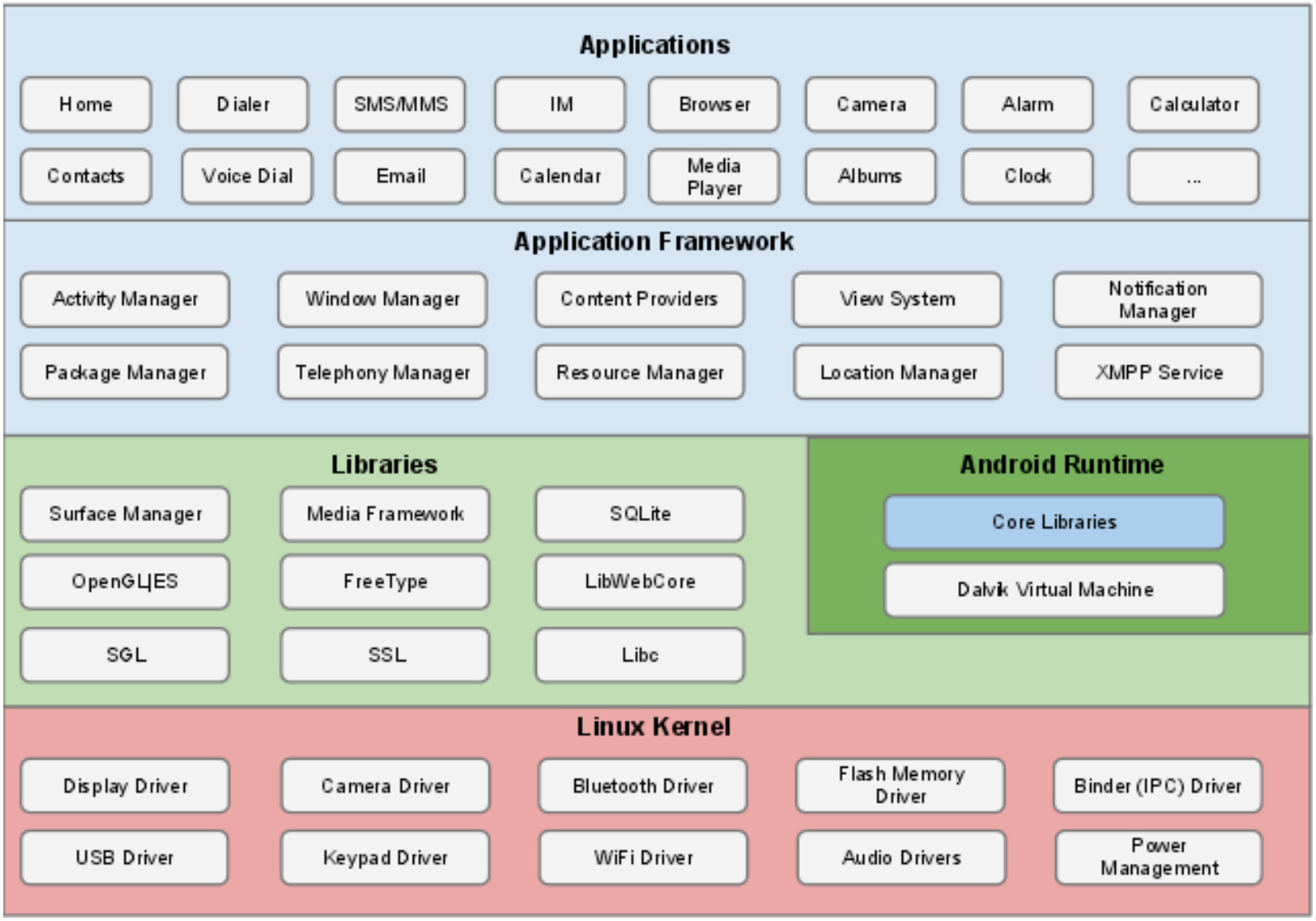


Because certificate serial numbers must be unique within each CA, this is enough to identify the certificate

Mobile Device Security (Android)

- Android
  - Based on Linux
  - Layers:
    - Android Application Runtime (generally written in Java, run in the Dalvik virtual machine; sometimes native applications or native libraries)
    - Android OS
    - Device Hardware
  - Applications
    - Pre-installed
    - User-installed
      - Via app stores
      - Via over the air (OTA) updates.

Android Software Stack



http://source.android.com/tech/security/index.html

Application Sandboxes

- Based on Linux: Has clear notion of users and permissions
- Each application
  - Assigns unique user ID (UID)
  - Runs as that user in a separate process
  - **Different than traditional operating systems** where multiple applications run with the same user permissions

Application Sandboxes (II)

- Desktop browser sandbox: language specific
- Android sandbox: baked into the OS, via the kernel
  - No restriction on how applications are written
  - Native code
  - Java code
- Conventional systems: memory corruption errors lead to complete compromise
- Android: memory corruption errors only lead to arbitrary code execution in the context of the **particular** compromised application
- (Can still escape sandbox -- but must compromise Linux kernel to do so)

File permissions

- Files written by one application cannot be read by other applications
  - Not true for files stored on the SD card
- It is possible to do full filesystem encryption
  - Key = Password combined with salt, hashed with SHA1 using PBKDF2.

Memory Management

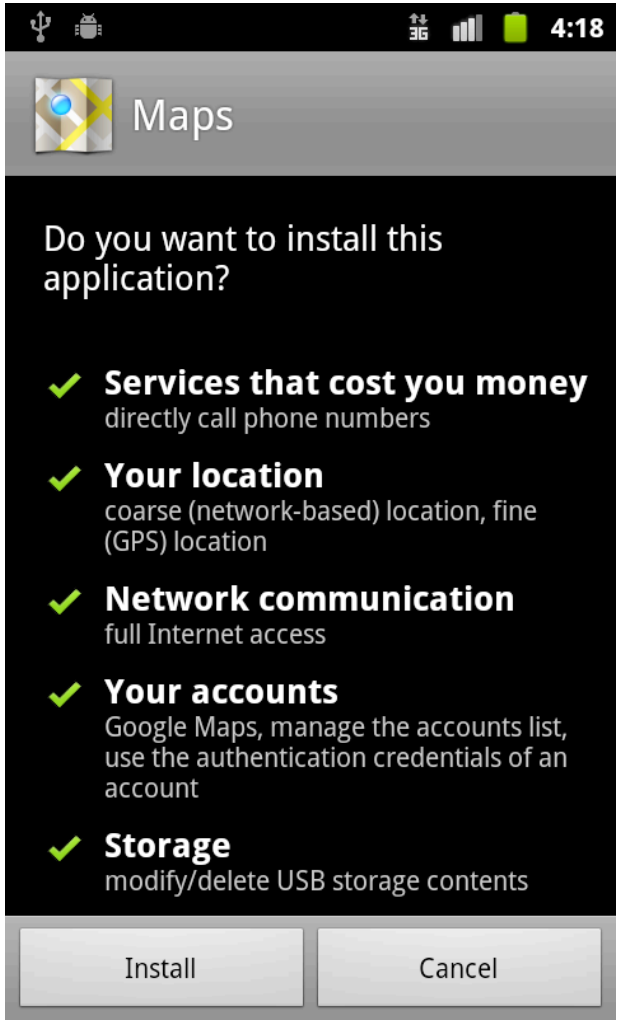
- Address Space Layout Randomization to randomize addresses on stack
- Hardware-based No eXecute (NX) to prevent code execution on stack/heap
- Stack guard derivative
- Some defenses against double free bugs (based on OpenBSD's dmalloc() function)
- ...
- (See http://source.android.com/tech/security/index.html)

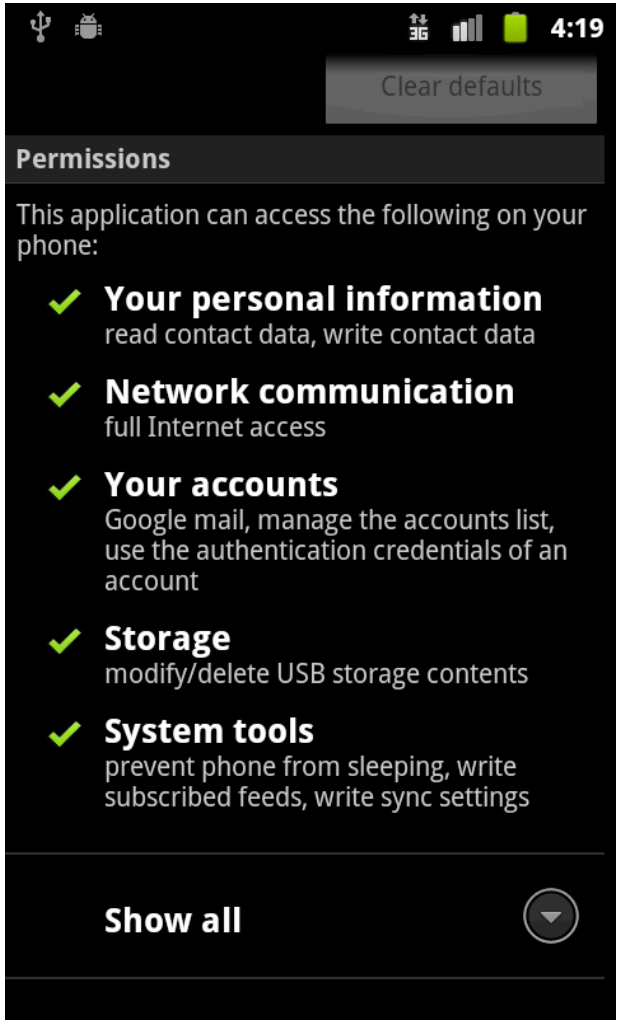
Applications

- Activity: Code for single, user-focused task
- Services: Code that runs in the background
- Broadcast Receiver: Receive Intents (messages from other applications)

- AndroidManifest.xml
  - Overall information about application (activities, services, ...)
  - Also specifies which **permissions** are required by applications

Permissions / Manifests





http://source.android.com/tech/security/index.html

Permissions

- Example permissions
  - Camera
  - Location (GPS)
  - Bluetooth
  - SMS functions
  - Network capabilities
- Cannot grant / deny individual permissions
- One accepted, users not notified of permissions again
- Security exception thrown if attempt to access resource not declared in manifest

Obtaining User Consent for Permissions

- General options:
  - At install time (manifests)
  - At time of use (prompts)
- Why manifests
  - Users are evaluating the application, the developers, etc, to see if they want the app
  - Prompts slow down user; hinder user experience
  - Users may just say "OK" to all dialogs without reading them
- Why prompts
  - At time of resource access
  - Opportunity for user to be more in control of actual resource use (app with GPS permissions should only actually access the GPS when the user wishes -- but can't tell with manifest model)
- (Alternative: User-driven access control, Roesner et al (2012))

Application Signing

- Apps are signed
  - Often with self-signed certificates
- Signed application certificate defines which user ID is associated with which applications
  - Different apps run under different UIDs
- Shared UID feature
  - Shared Application Sandbox possible, where two or more apps signed with same developer key can declare a shared UID in their manifest

Shared UIDs

- App 1: Requests GPS / camera access
- App 2: Requests Network capabilities

- Generally:
  - First app can't exfiltrate information
  - Second app can't exfiltrate anything interesting
- With Shared UIDs (signed with same private key)
  - Permissions are a superset of permissions for each app
  - App 1 can now exfiltrate; App 2 can now access GPS / camera

Privilege Redeligation

- Permission redeligation:
  - App 1 does not have access to resource X
  - App 2 does have access to resource X
  - App 1 gains access to resource X via App 2
  - (App 1 and App 2 not signed by the same party)
- Video example:
  - https://plus.google.com/photos/110581955720098741626/albums/5638277509860549393/5638277512553016018

Regarding the previous video

- From the slides for "Permission Re-delegation: Attacks and Defenses" by Adrienne Porter Felt, Helen J Wang, Alexander Moshchuk, Steve Hanna, Erika Chin: