Mobile Security and Privacy
Guest Lecture by Serge Egelman
Nov. 14

Eric Love

November 21, 2012

1 Avoiding Mobile Malware

- Most mobile phone malware uses SMS capability (73%)
- Very few legit apps use it (3%)
- Can we just say "don't use apps that require SMS?"
- Installing an app:
  - Clicking "download" in app store lists permissions an app requires
  - Perm list could require scrolling to view completely
  - Determining if an app asks for unreasonable perms can take many steps!
- How can we make it easier to follow good security advice?
  - Iconize permissions in marketplace search results?
  - Aggregate permissions?
  - Make it easier to find all installed apps that use a specific permission – aids in finding source of malicious behavior
  - Present user a dialog box the first time an app tries to use a certain permission?

2 Presenting Permissions to Users

- Android has > 150 permissions!
- iPhone has 2, Windows 16
- With many perms, users habituated to accept, confused in general
- With too few perms, users can easily regret installing apps that have unpredictable access (e.g. an app might steal all users' contacts and upload to a server without their knowledge)

3 Android Comprehension Study

- Recruited users with AdMob, ran ad for 1 hour, got 308 respondents
- Multiple choice questions tested understanding of permissions
- Had to combat random responses from users:
  - Can compare to expected values for random guessing
  - $E[correct|randomguesses] = \frac{|Questions|}{\sum_{n=0}^{4} \binom{4}{n}}$
  - Even for users with low knowledge, $E[correct]$ was greater than this
  - User outlier detection algorithm of some kind (e.g. reject all responses 2σ below the mean)
- Other strategy: validate results by lab experiment
  - Can supervise users, ask follow-up questions
  - Interrogating users helps explain why comprehension so low
  - Have users install two apps, observe whether they look at perms,
  - See if they can answer "can the app send SMSs?" while they're looking at permissions
  - Results:
    - 82.5% don't look at perms, 40% unaware that perms exists, 40% don't care
    - Explanations: many too habituated to say yes (too many requests)
    - Many unaware that perms exists because they're presented too late in the install process
    - People could not identify perms from apps they use regularly
    - Perms not presented completely: "read SMS" could be reading incoming or outgoing or both

3.1 Permission-granting mechanisms

What about permissions other than install-time?

- Run-time warnings add contextual information when a resource is requested
- Potential pitfall is that runtime warnings seen too often, so users become habituated to click through

Curating the market:

- platform owner (such as Apple) must approve all apps
- lots of work: human beings must audit all app submissions
- review process is usually opaque
- can fail badly, as in the case of Path for iOS (stole contact data)

Have a trusted UI:

- Ex: Developer can specify a button to send an SMS, but only the OS can produce the actual panel that will send the SMS
- Must be careful to avoid clickjacking attacks
- Cannot grant things in the future ("join this WiFi network when its in range")
- Cannot handle asynchronous requests (i.e. user not initiating the request)

Are there other mechanisms? Open question...

4 How should different permissions be granted?

HotSec paper on this topic. Basic ideas

- Categorizes 83 permissions across Android, Windows Phone 7, iOS, Mozilla WebAPI
- Characteristics that might help decide how to grant:
  - Risk level?
  - Is there an incentive for abusing it?
  - Revertible – can bad effects be undone?
- Examples:
  - Remote wipe. High risk, but low incentive.
  - Changing time: low risk, easily revertible, low incentive. ⇒ seems not to need a dramatic run-time warning
- Implicit access:
  - Minimize habituation by not bothering user when not warranted
  - Give visible indication when a permission is being used
  - Make it easy to identify what apps are using a perm
- Flowchart:
  - Revertibility (can undesirable effects be undone?) Yes ⇒
  - Severity (if abused, is it just an annoyance?)
  - Initiation (did the user make the request?)
  - Alterable (can the action be altered by the user?)
  - Approval (does it need to work without immediate user approval?)

Some example permissions we discussed in class

- Factory reset: pretty severe, but infrequent, so you could use a trusted UI. What is the incentive to abuse this?
- Reading user's email inbox:

  - severity could depend on other capabilities. If the app has no internet access, then maybe it's not so bad. But almost all apps have internet access, so it's probably severe.
  - Do apps need this? Many do for ad analytics, which enables them to be free
  - Trusted UI: let user pick an email or mailbox to grant the app
  - Or could differentiate based on whether the app is intended to replace the system email app. In this case, use a general install permissions.

- Adding calendar events: pretty low severity, low incentive. Could defend against it by just adding a button in the calendar to delete all events added by a certain app.

- Sending SMS messages: main use case would be that the user is in an app, wants to initiate a specific SMS, so you could use a trusted UI. There are some cases where you would like asynchronous SMSs – the message should be sent at a fixed time in the future, or periodically as a reminder.