Android Ecosystem Integrity

Possible Malware Cross-Infection Vector

Tomáš Rosa

http://crypto.hyperlink.cz

November 2011, Prague

Abstract

- We first show the Screen Lock Bypass application at work.
  - This is an interesting forensics/hacking technique in itself.
- We then conclude by noting a possible way of an effective malware cross-infection.
  - The observation is trivial. Its impact, however, can really be dramatic.
  - Especially in the area of two-factor authentication applications.

Experimental Setup

- The proof of concept demo was exercised on Google Nexus S I9023XXKF1 with Android version 2.3.6, build GRK39F.

Screen Lock Bypass (SLB)

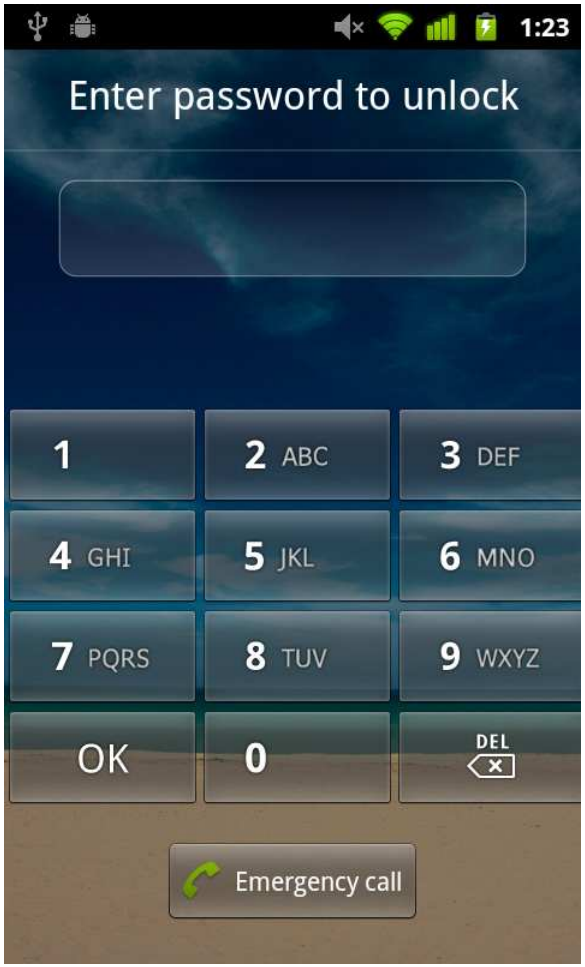
- Developed by Thomas Cannon [1], popularized by Andrew Hoog [2], and freely available on the Android Market [3].
- Its official purpose is to help users who accidentally forgot their screen lock gesture or PIN.
  - Anybody who knows the login name/password for the Gmail account associated with the particular Android device can use this application to try to unlock the screen.
  - The success ration may not be 100 %, but it is quite high anyway.
  - In particular, we did not encounter any problem during several trials we have made for this presentation.

The Dark Side

- As was already noted in [2], this application may be used *not only* by the legitimate device owner.
  - Just anybody, who knows the respective Gmail credentials can give it a try.
  - Obviously, the Gmail credentials seems to be quite “magic”.
    - And that is just the beginning…

The Screen (Un)Lock At Work

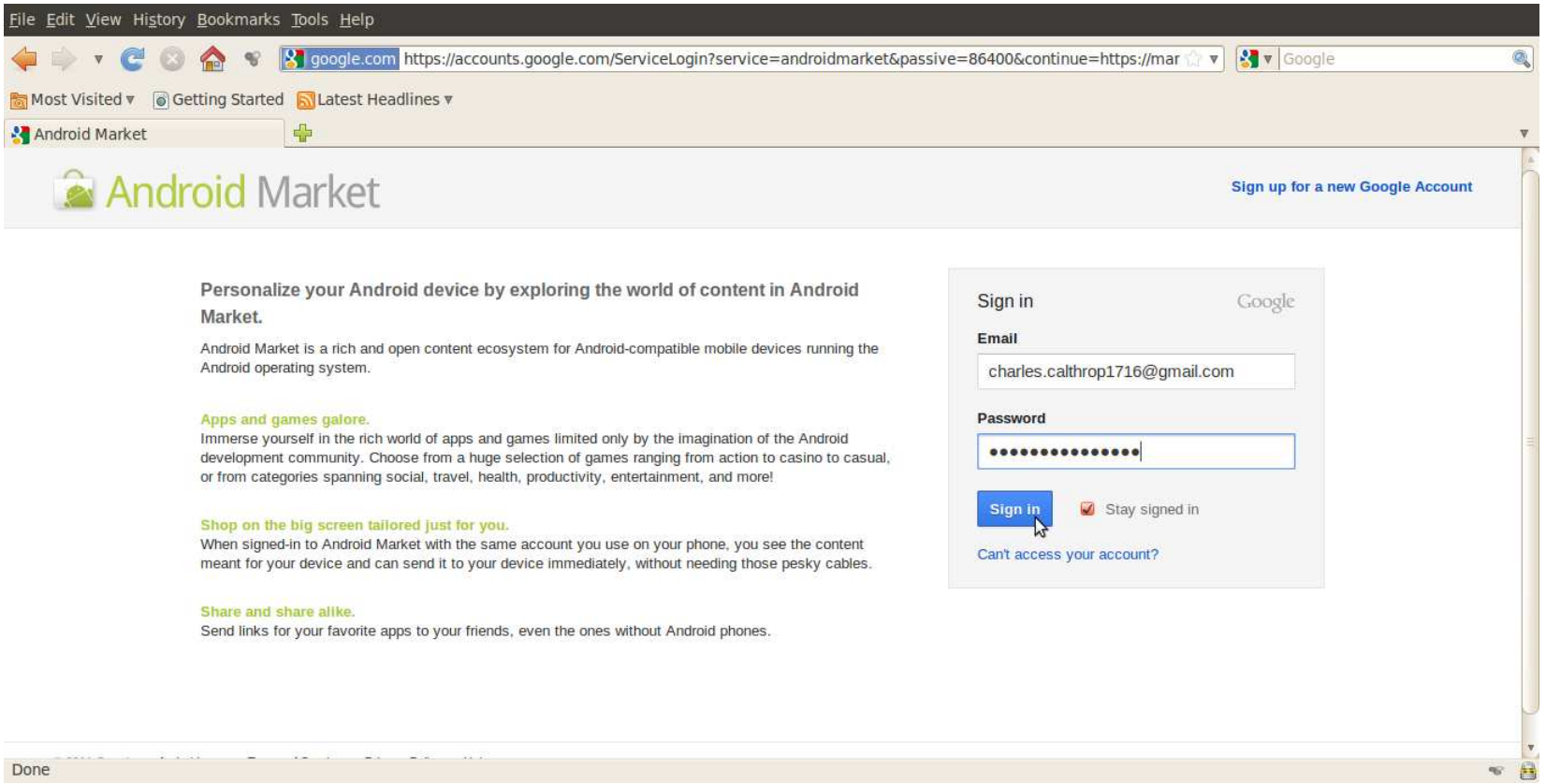
- Let us assume that the device display is locked by a PIN that we somehow cannot recall…



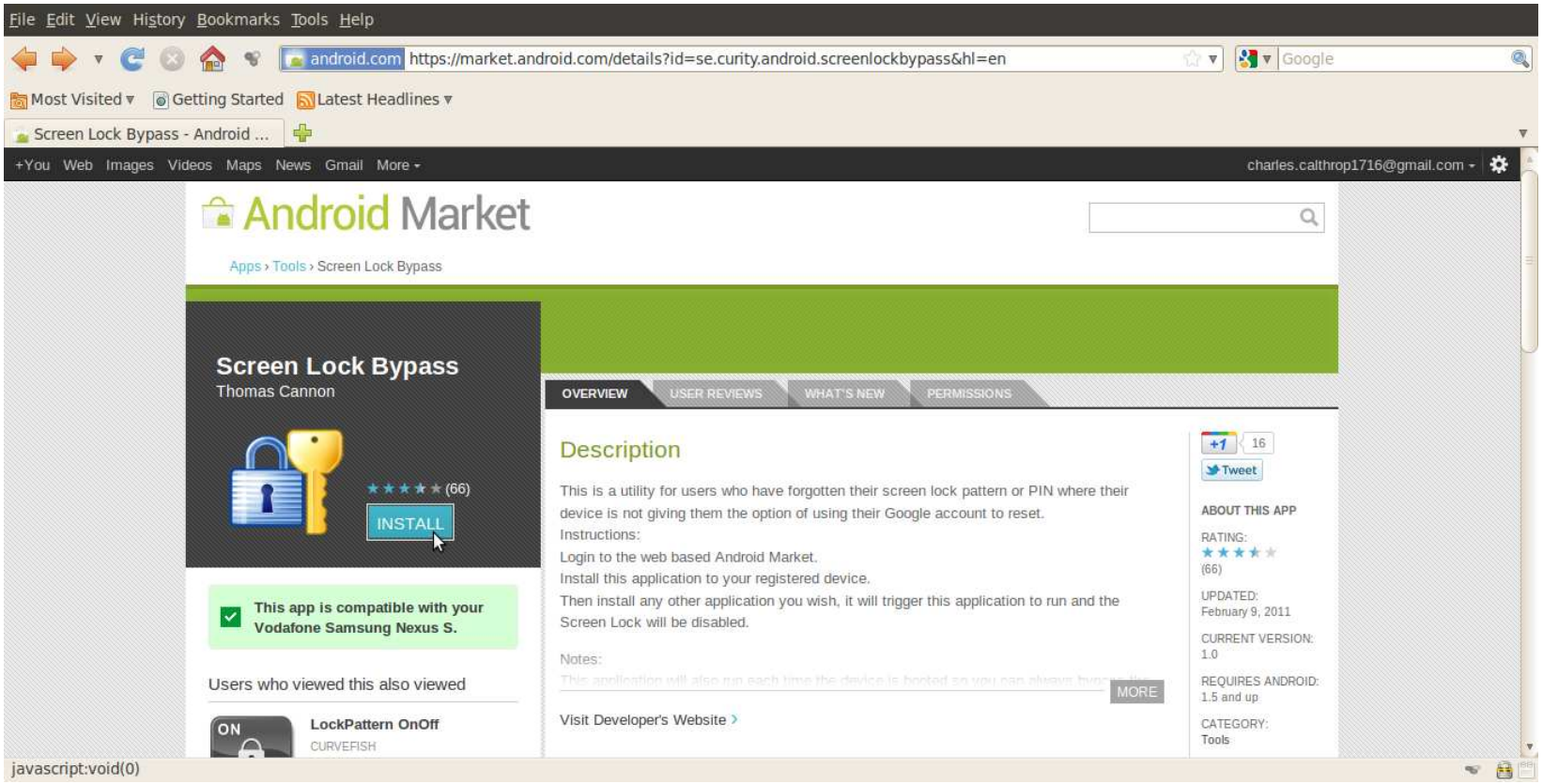
Gmail Account Sidekick

- Let us assume we somehow *can* recall the associated Gmail account login name/password…
- So, we do the following (from any PC/Mac)
  1. go to http://market.android.com
  2. use the name/pwd to log in – note the same credentials apply here as for that Gmail account
  3. find the “Screen Lock Bypass” application and let it install to the associated Android device

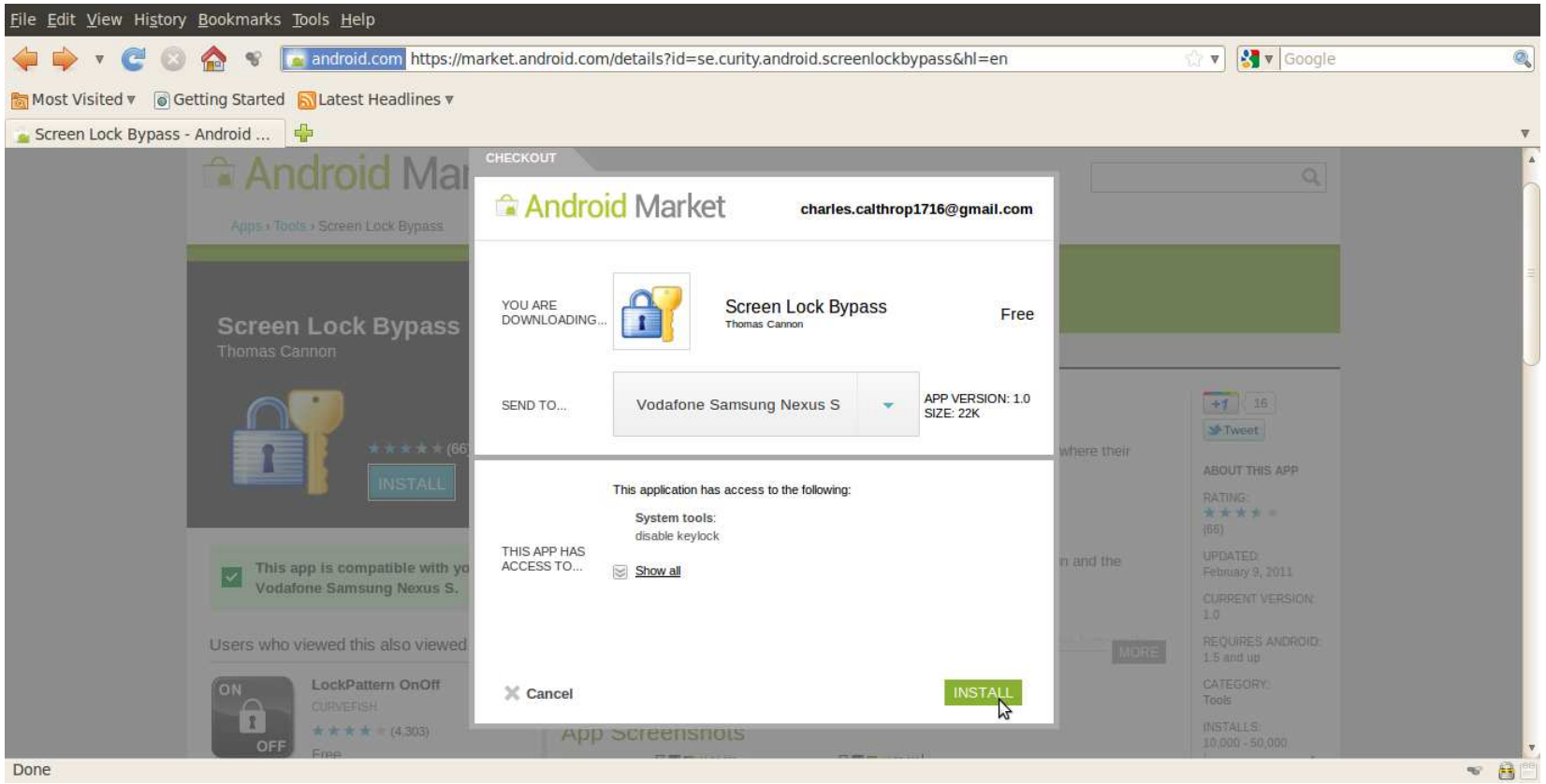
Android Market Login



Finding SLB Application



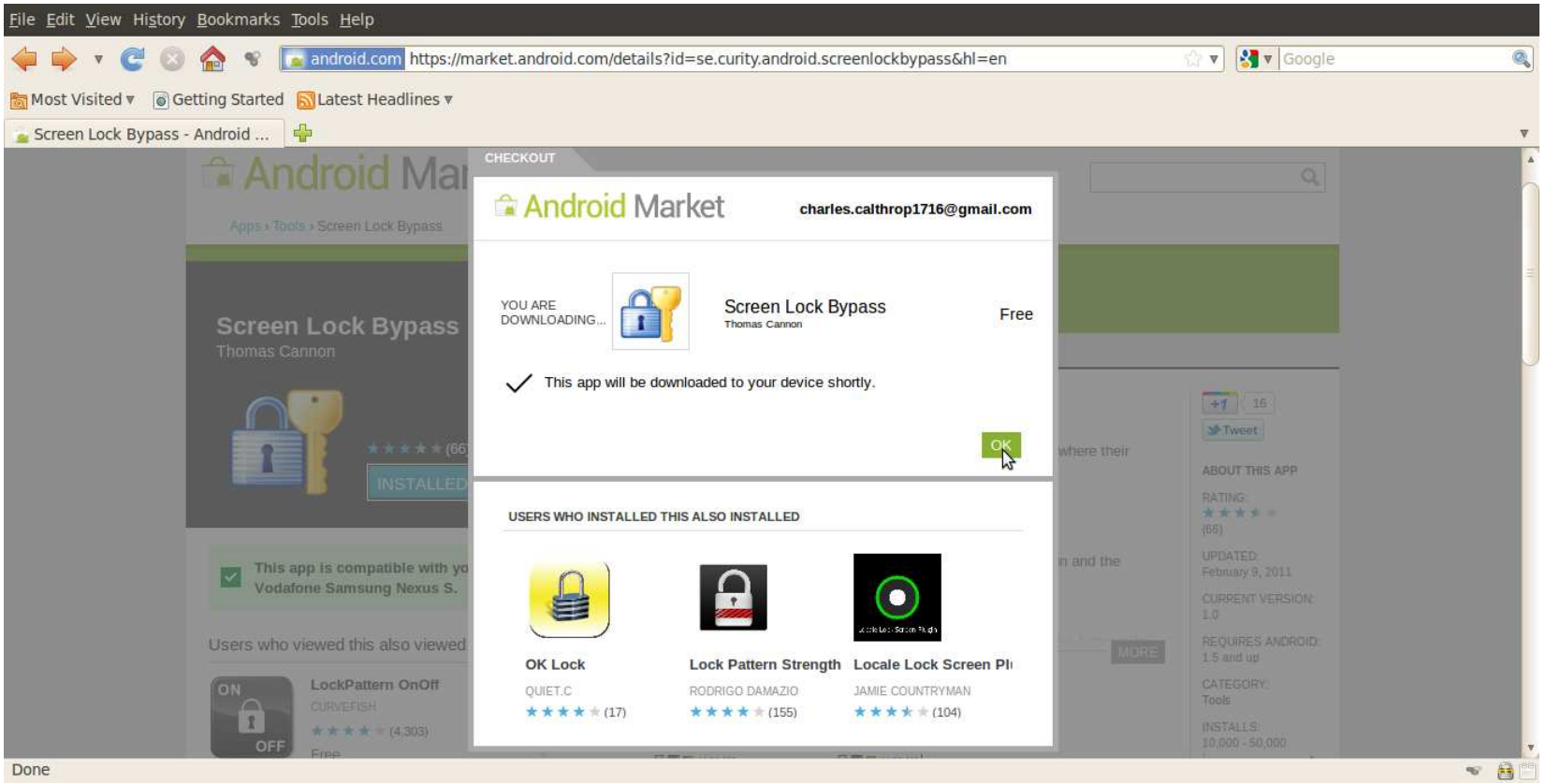
Starting SLB Installation



Telephone Number – Who Cares?

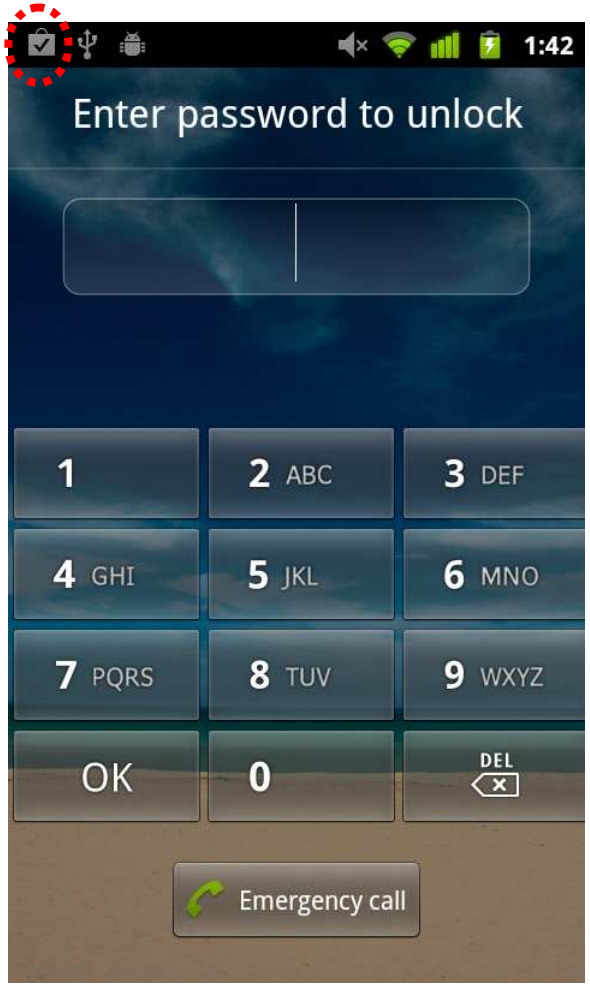
- We should emphasize it is unnecessary to know the telephone number of the target Android device.
- We either do not need to know any other a-priori identification of the device.
- This is because of Android Market offering us the list of associated devices automatically.
  - All we have to do is to choose a device from the list.

Installation In Progress



Meanwhile On the Device

- While the application is being installed, there is no user interaction required at the mobile device side at all.
- The name of the application flashes briefly in the status bar, leaving on just a tiny symbol of a successful installation.



Recall, OTA = Over The Air

- Note the SLB application was installed through a service channel that Google uses to silently manage Android devices worldwide.
  - This permanent data path is kept automatically by each Android device linked to the Android Market portal.
  - That means, we do not need to tweak the mobile phone in any way to start downloading.
  - It may be resting on a table as well as in somebody's pocket – just in any place with GSM/UMTS service coverage.
  - The display does not have to be turned on before the installation starts.
  - Well, this all really is a silent service…

Hands-Off Application Startup

- So, we have downloaded the (pirate) application on the Android device.
- The question is, however, how to make this code run?
  - Obviously, we cannot do that manually, since the screen is locked.
  - Unfortunately, the Android OS provides several reliable ways on how to do that.

Android Broadcast Receiver

- This is an application component [4] responsible for inter-process communication based on broadcast `Intent` mechanism.
  - Usually, developers use a `BrodcastReceiver` derivatives to hook up for asynchronous system events like:
    - `android.provider.Telephony.SMS_RECEIVED`
    - `android.net.conn.CONNECTIVITY_CHANGE`
    - `android.intent.action.PHONE_STATE`
    - etc.

`BroadcastReceiver` Setup

- To register a `BroadcastReceiver` component, it suffices to list it in the respective `AndroidManifest.xml`.
  - This xml file is stored in the application package and it gets processed automatically during the application installation [5].
  - Therefore, no single code instruction of our application needs to be run to hook up for a particular broadcast `Intent`.

Registration Example

- Remember – it is all done in a package configuration file.
  - We do not need to run our code to register for a broadcast `Intent`.

```
…
<receiver android:name=".SniffReceiver">
  <intent-filter android:priority="256">
    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
  </intent-filter>
</receiver>
…
```

Once Upon A Broadcast…

- When the particular broadcast is fired, the Android operating system invokes those registered receivers.
- This way our `onReceive()` method gets called and – yes, we have got it – our application code is up and running!
  - Actually, it is a bit complicated when it comes to the order of calling these receivers and possible event cancellation, but this is not important for use here.

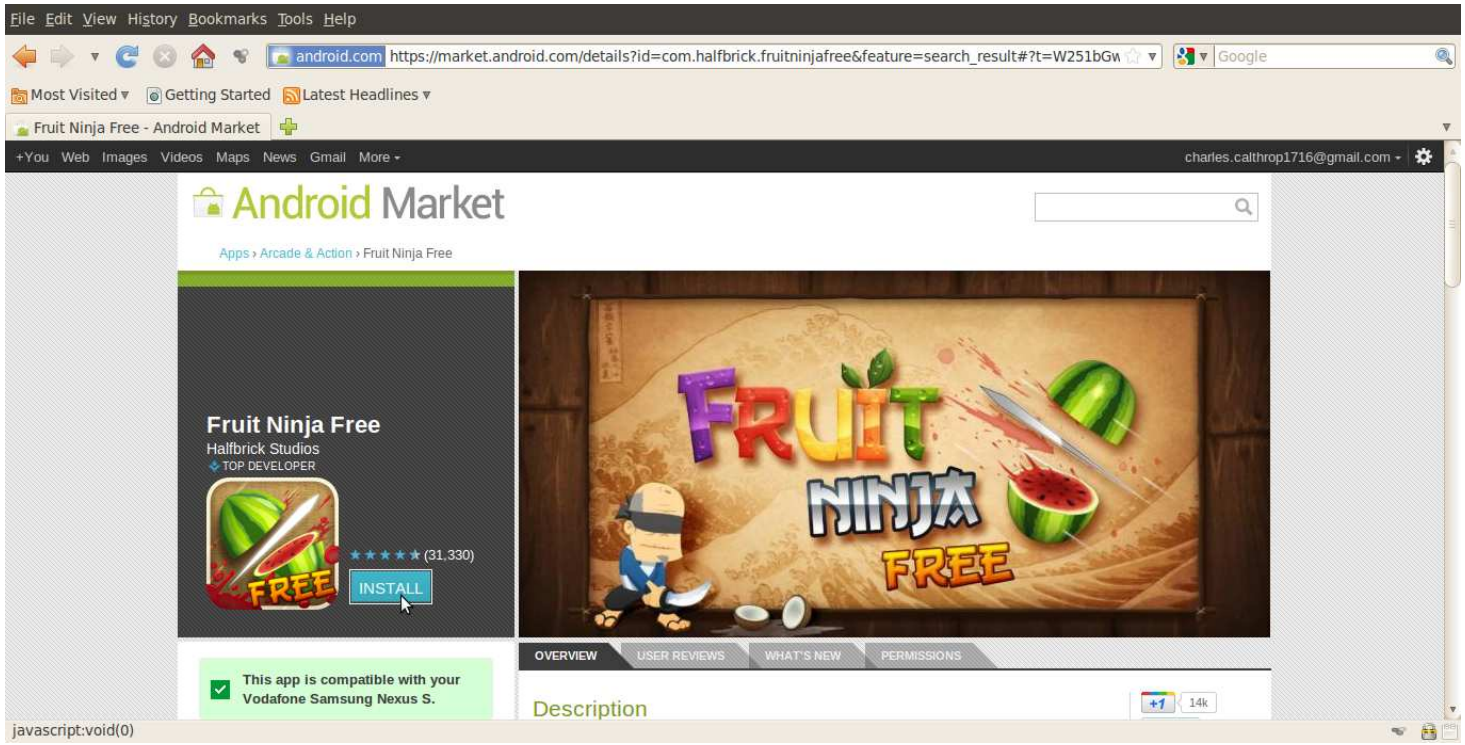
Back To SLB

- The Screen Lock Bypass, in particular, registers to the following broadcasts:
  - `android.intent.action.PACKAGE_ADDED`
    - Triggers when a new package is installed.
  - `android.intent.action.BOOT_COMPLETED`
    - Triggers after finishing OS boot and startup procedures.

Two Ways to Unlock

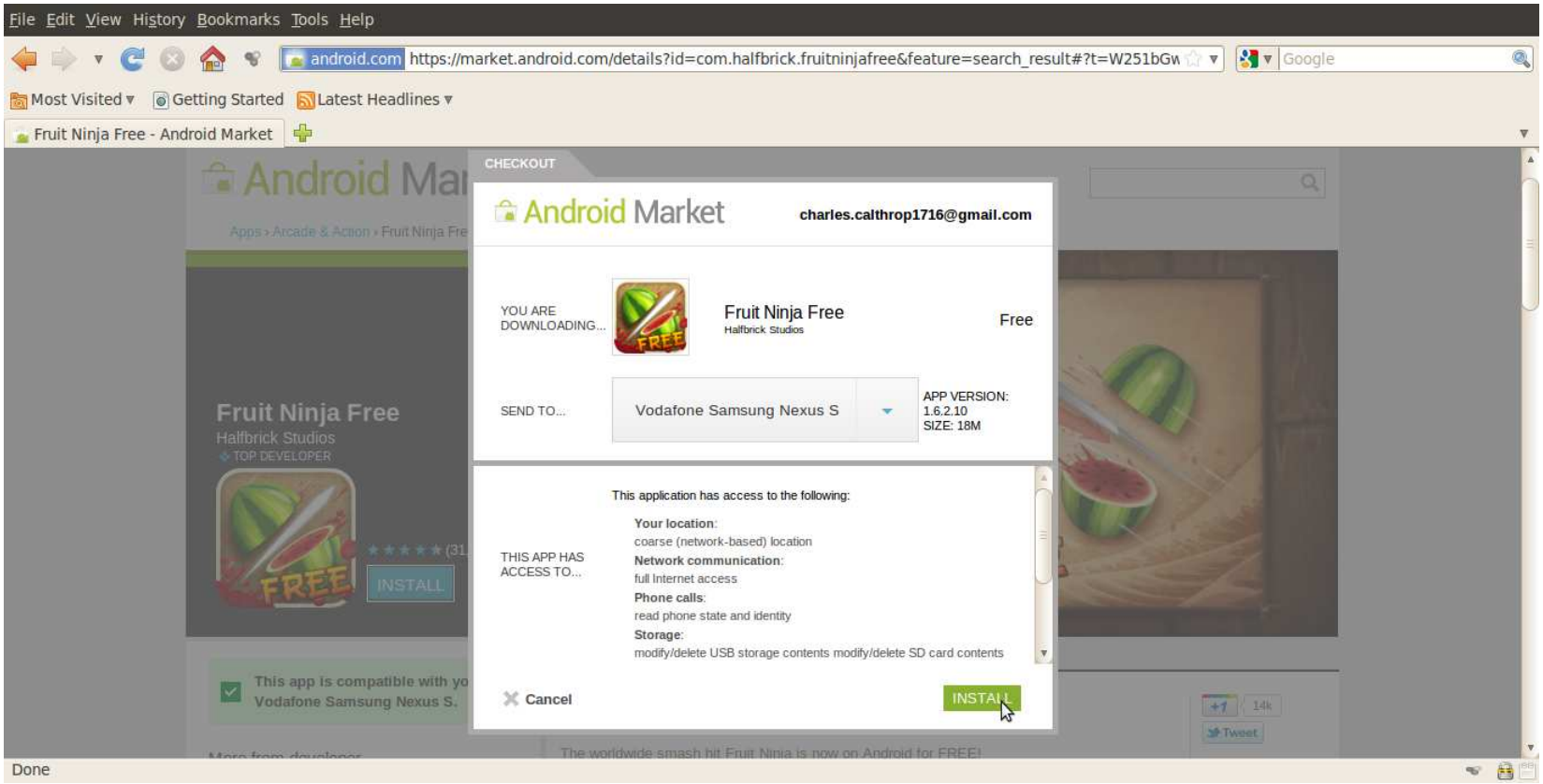
- According to the aforementioned events, there are basically two ways on how to trigger SLB activity.
  1. To install just another application package from the Android Market in the same way as we did for SLB itself.
  2. To switch off/on the device, hence triggering the `BOOT_COMPLETED`.
- We have verified both ways worked well in our experimental setup.

Going the First Way

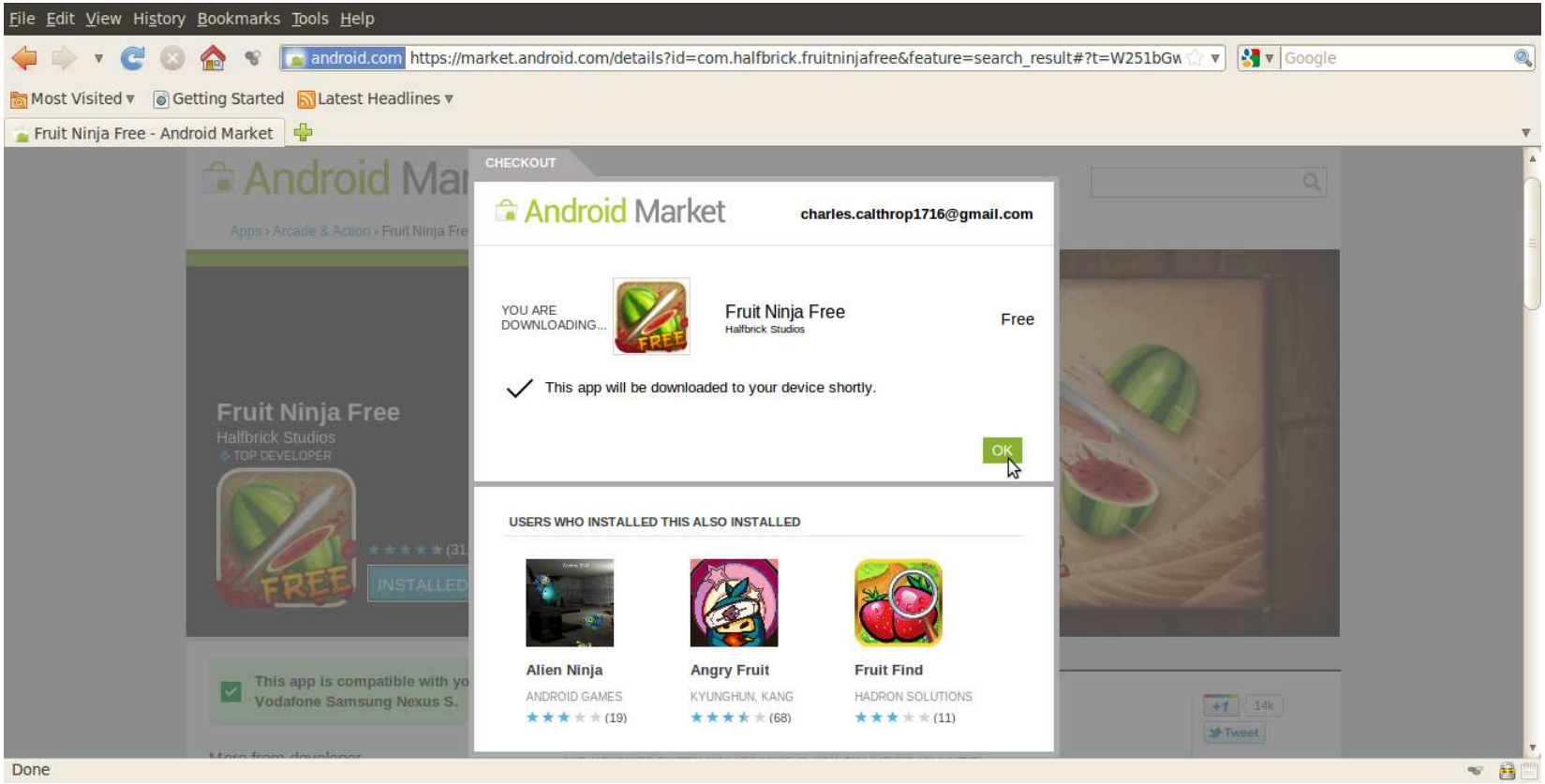


- It really does not matter what application we choose.
- Important is just the final event that triggers our `onReceive()`.

Installing Dummy Application

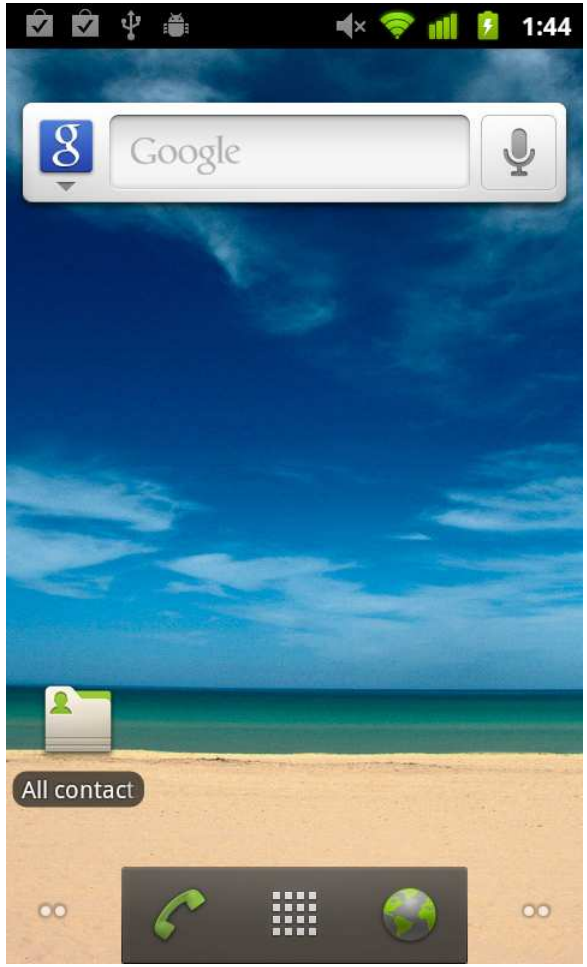


Installation In Progress



Having Finished Installation

- Successful installation triggers `PACKAGE_ADDED`.
- This in turn starts the SLB trap.
- Suddenly, the screen lock disappears…



Possibly

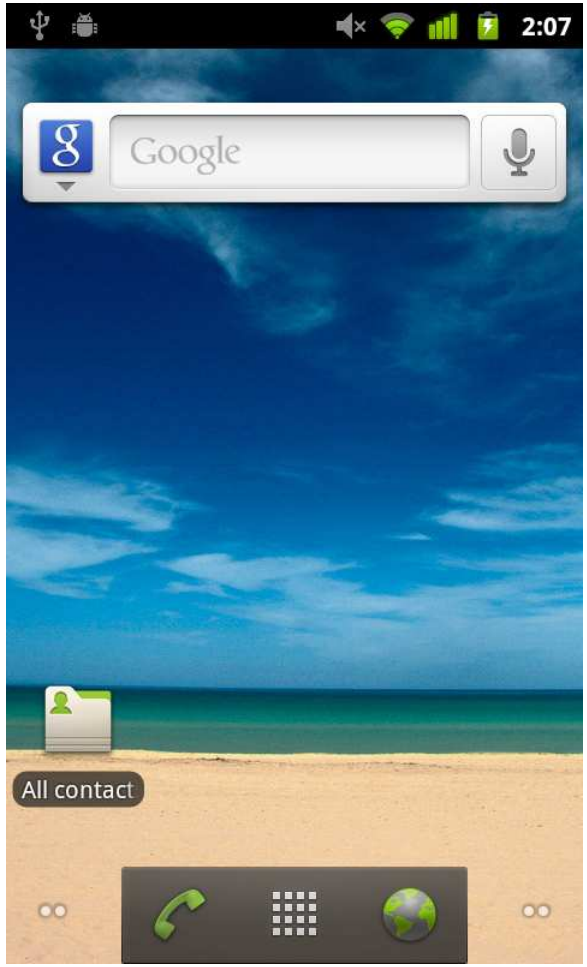
- Well, we can also enjoy playing Fruit Ninja.
- But we do not have to.
- Just for fun…

Remember… (regarding SLB)

- We have downloaded an application package on the Android device.
- We have granted any user permissions we needed to that package.
- We have run a code of that package.
- We did not need to directly operate with the mobile device in any way.
- The only thing we needed was an internet access and a valid login name/password for the associated Gmail account!

Working The Other Way

- By simply switching off/on the device, we can trigger `BOOT_COMPLETED`.
- This again runs a SLB code.
- Again, the screen lock disappears happily…



Remember Again

- The only thing we needed was an internet access and a valid login name/password for the associated Gmail account!
  - Well, this time we used the power off/on switch.
  - The attacker, however:
    1. Can use the former approach using a dummy package installation.
    2. Can just wait until users “recycle” their devices by themselves.

Access Rights Revisited

- The Android operating system relies mainly on user-granted permissions [6].
- During the application installation, the user is asked whether to allow or deny permissions required by the particular `AndroidManifest.xml` [5].
  - Well, this model itself is quite questionable as users may not be fully aware of the possible impact.
  - Furthermore, it is especially non-trivial to discover the risk of various permission synergy effects.
  - Anyway, this is not the topic we want to address here.

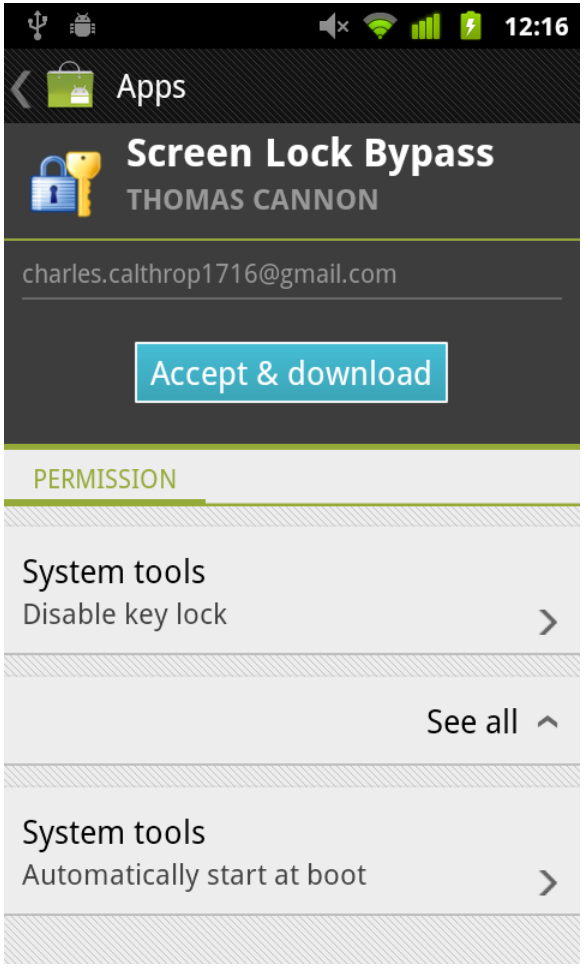
User-Granted Permissions Limits

- We should note that there are some privileges that cannot be granted even by explicit user confirmation.
  - For instance, it is not possible to directly grant root access to the underlying Embedded Linux core.
  - With user-granted privileges, we can, however, run a possible root exploit…
- On the other hand, the power of user-granted permissions is still considerable.
  - For instance, permissions needed by an SMS sniffer can be fully granted this way.

Let Us Experiment

- To see permission granting process at work, we can try installing SLB directly from the Android Market application running on the particular Android device.
  - Well, this does not make a sense, but we do this for another purpose.
  - We want to demonstrate how the user-granted permission mechanism works.

Illustrative Screenshots



As Bad As It Looks

- Well, but when we installed SLB through the web interface, we did not need to grant these permissions. Or did we?
  - We did, but that time it was granted through the web interface instead (cf. the former screenshots).
- Does it really mean…?!
  - Unfortunately, yes.
  - Provided we have respective Gmail credentials, we can choose any application form the Market, give it any user-granted permission, send it to the victim's device, and run it!

Cross-Infection Highway

- Time to time, users log to their e-mail accounts from “ordinary” computers, too.
  - What about if that PC/Mac is infected by a malware that steals Gmail login name/password?
  - The conclusion is immediate – such a malware can instantly spread to the associated Android device.
  - There is no need for any further user cooperation!
  - This all in fact effectively breaks those popular SMS-based two-factor authentication schemes…

Conclusion

- Provided the way Android ecosystem is currently managed, the following is true:
  - Compromised Gmail account implies compromised associated Android device.
  - This opens up a whole highway for malware cross-infection from PC/Mac to Android mobile.
  - Furthermore, this signalizes the emerging end of the two-factor mobile-based authentication as we know it…

Thank You For Attention

Tomáš Rosa, Ph.D.
http://crypto.hyperlink.cz

References

1. http://thomascannon.net/blog/2011/02/android-lock-screen-bypass/
2. Hoog, A.: *Android Forensics – Investigation, Analysis and Mobile Security for Google Android*, Elsevier, 2011
3. https://market.android.com/details?id=se.curity.android.screenlockbypass
4. http://developer.android.com/reference/android/content/BroadcastReceiver.html
5. http://developer.android.com/guide/topics/manifest/manifest-intro.html
6. http://developer.android.com/guide/topics/security/security.html