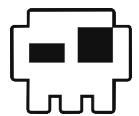
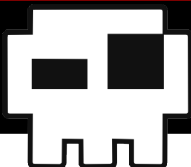




Hacking Android for fun & profit





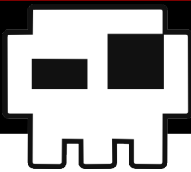
Plan (1/3)

Android System

- ☞ Features
- ☞ Permissions
- ☞ API & SDK
- ☞ Debugging mode

Overt & covert channels

- ☞ Overt channels overview
- ☞ Covert channels overview
- ☞ Lick everybody's asses to get access...
- ☞ ...and hide to be stealthy



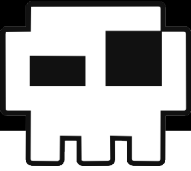
Plan (2/3)

Remote control & triggers

- ☞ Internet polling
- ☞ Short Messages (SMS)
- ☞ Class 0 Short Messages as a covert channel

Hacking Android's Java API

- ☞ Reflection is your best friend
- ☞ Go deeper and use what you need
- ☞ How to send Class 0 short messages with Android SDK ver. > 6

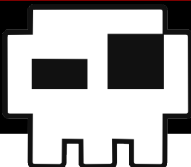


SpyYourWife

- ☞ Instant geolocation app.
- ☞ Class 0 SMS transport layer
- ☞ Geolocation tricks





Conclusion

- ☞ Android, the most awesome mobile phone of the world ?







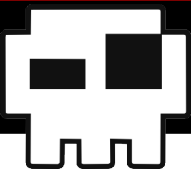
Plan (1/4)

Android System

-  Features
-  Permissions
-  API & SDK
-  Debugging mode

Overt & covert channels

-  Overt channels overview
-  Covert channels overview
-  Lick everybody's asses to get access...
-  ...and hide to be stealthy



Android

 OS for mobile phone and tablets

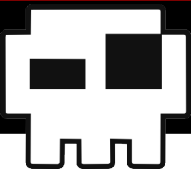
- ☞ Owned by Google Inc.
- ☞ Open-source (well, almost)

 Advantages







- ☞ SDK provided by Google
 - Dedicated development tools
- ☞ Code available
- ☞ Android emulator based on qEmu
- ☞ Specific Eclipse plugin

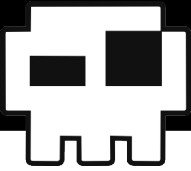
 <http://android.google.com>





Generic features (smartphones)

-  WiFi connectivity
-  GSM/CDMA connectivity
-  Global Positioning System
-  SMS/MMS capability
-  Internet connectivity
-  Multiple sensors (proximity, orientation, ...)

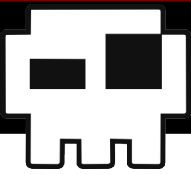


Security Model

- ☞ Based on « permissions »
- ☞ Permissions rule Android's world
 - Internet access
 - Sensor management
 - Telephony management

Each application runs in its own world

- ☞ Separated files
- ☞ Cannot interact with another app.



Android

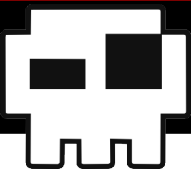
SDK

- ☞ Google provides us with a useful SDK
- ☞ Regularly updated
- ☞ Available on Windows & Linux
- ☞ Create APK files (Android app. package files)

Java API

- ☞ Android provides many useful components
 - Sockets
 - Multi-threading
- ☞ They are packed in android.jar
- ☞ Available from every application





Android

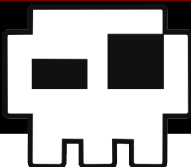
Debugging mode

- ☞ Allow application debugging through USB
- ☞ Allow application deployment through USB
- ☞ Anybody having a physical access to the phone can enable this mode

Unknown sources





- ☞ Dangerous option of Android
- ☞ Enable any application to be install from anywhere

 User is responsible of his/her own safety !







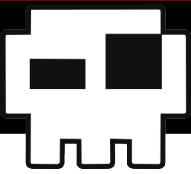
Plan (1/4)

Android System

-  Features
-  Permissions
-  API & SDK
-  Debugging mode

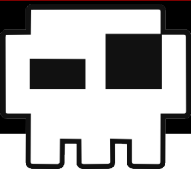
Overt & covert channels

-  Overt channels overview
-  Covert channels overview
-  Lick everybody's asses to get access...
-  ...and hide to be stealthy



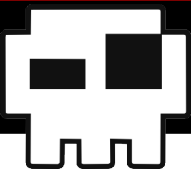
Overt & covert channels

- 🤖 Everything is locked or almost locked
- 🤖 How to transfer confidential information to the outside ?
 - ☞ Use generic communication channels
 - Internet through HTTP/S
 - Intent
 - SMS
 - Application logs
 - ☞ Use other communication channels
 - Light state
 - Active processes or threads
 - Sound, etc.



Android Intents

- 🤖 Android is based on « Activities »
 - ☞ Kind of process
 - ☞ An application can have one or more activities
- 🤖 Activities can send and receive « Intents »
 - ☞ An intent contains
 - A name
 - And extra params
- 🤖 It is a convenient way to transfer data between two activities





Covert channels

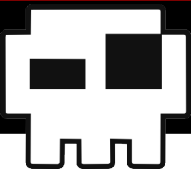
Covert channel

-  Can be use to transfer data between applications with different permissions
-  This is called « collusion »

Based on inoffensive channels

-  Light state used to transmit data between two applications
-  Modifying the number of running threads in order to transmit data

 The stealthier the covert channel is, the less data we can send



Overt channels

Communication channels


 They are used as usual

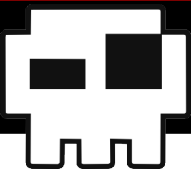
- HTTP requests
- SMS/MMS
- TCP connections

 They are easily detected

But user is very vulnerable

 Thanks to a bit of social-engineering, it is easy to convince the user to install our application

 Permissions are not checked by the user (non-technical)

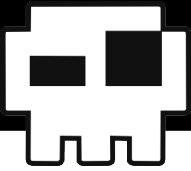


Lick everybody's asses ...

Overt channel based malware

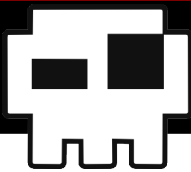
- ☞ Application is released on the Android Market
 - Requires READ_PHONE_STATE permission
 - Requires INTERNET permission
- ☞ In the Market, the application states that
 - It does not send private information over Internet
 - It uses the READ_PHONE_STATE permission to access only the phone state








... and hide to be stealthy

- 🤖 Overt channels can be easily monitored
 - ☞ TaintDroid
 - ☞ Intent-based communication
 - Easy: register an intent receiver for a specific intent
- 🤖 Let's make it harder :)
 - ☞ Use encryption with cryptographic API
 - ☞ Design a home-made encoding
- 🤖 In fact, once the application installed it's all fucked up






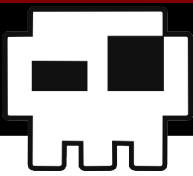
Plan (2/3)

Remote control & triggers

-  Internet polling
-  Short Messages (SMS)
-  Class 0 Short Messages as a covert channel

Hacking Android's Java API

-  Reflection is your best friend
-  Go deeper and use what you need
-  How to send Class 0 short messages with Android SDK ver. > 6



Remote control & triggers

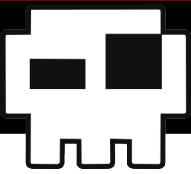
🤖 Once a malware is installed, we want to

- ☞ Take complete control of the phone
- ☞ Remote control the phone
 - Execute nasty actions
 - Send pr0n SMS/Email
 - ...

🤖 This can be done with:

- ☞ Internet polling
- ☞ Specific triggers

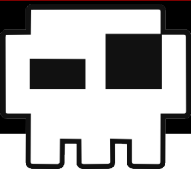




Internet polling

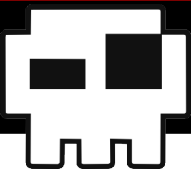
- 🤖 Based on regular HTTP requests
 - ☞ Requires Internet connectivity
 - Not always available
 - Bandwidth limited
 - Quotas set by many Telcos
 - ☞ Require a server-side script with a database
 - Costs money and time
- 🤖 Needs a running background application !
- 🤖 Well, not a good way to RC a phone ...





Triggers

- 🤖 Instead of polling,
 - ☞ Wait for an event to occur !
- 🤖 Many ways to trigger an action
 - ☞ SMS
 - ☞ phone call
 - ☞ Geolocation
- 🤖 SMS & phone calls can be easily intercepted by a dedicated application
- 🤖 No background application, the activity is loaded by the OS !



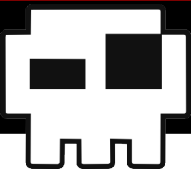
Triggers

Advantages:

- ☞ Easier to implement
- ☞ Still work when Internet connectivity is down
- ☞ Still work when phone is asleep
 - Polling requires the application to stay in background
 - Background application might be closed if unused

Coolest triggers

- ☞ SMS
- ☞ Phone call

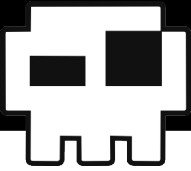


Triggers

SMS

- ☞ Can be intercepted on every Android device
- ☞ Contains only a hundred bytes of data (133 in 8bits encoding)
- ☞ Different classes of SMS
 - Class 0: SMS must be showed instantly and not saved on SIM or in the phone
 - Class 1: « normal » short message
 - Class 2: SM contains SIM data
 - Class 3: SM should be forwarded to an external device


 Short message of class 0 is normally never sent by a phone



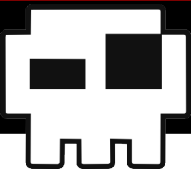
How to intercept SMS ?

 When the Android system receives an SMS, it broadcasts a specific Intent

☞ `android.provider.Telephony.SMS_RECEIVED`

 We can set in the `AndroidManifest.xml` file (in the app.) an Intent receiver that reacts on this Intent

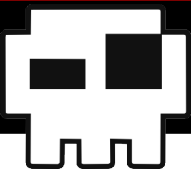
```
<receiver android:name=".BusterReceiver">
    <intent-filter android:priority="100">
        <action
            android:name="android.provider.Telephony.SMS_RECEIVED"
            />
    </intent-filter>
</receiver>
```

How to intercept SMS ?




- 🤖 The priority is important: the higher, the better
- 🤖 Android will launch the Intent receiver when a SMS is received
 - ☞ Our BroadcastReceiver will be the first notified of this SMS
 - ☞ We are able to avoid the broadcast of the event to the underlying broadcast receivers (lower priority)

```
private final String ACTION =
"android.provider.Telephony.SMS_RECEIVED";
public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(ACTION))
    {
        this.abortBroadcast(); /* avoid further broadcast */
    }
}
```






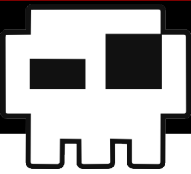
Plan (2/3)

Remote control & triggers

-  Internet polling
-  Short Messages (SMS)
-  Class 0 Short Messages as a covert channel

Hacking Android's Java API

-  Reflection is your best friend
-  Go deeper and use what you need
-  How to send Class 0 short messages with Android SDK ver. > 6



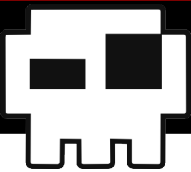
Hacking Android's Java API

Android Java API

- ☞ Contains every component needed by every android application
- ☞ Designed on an object model
 - Private classes, methods and properties
 - Public classes, methods and properties
 - Internals are hidden by methods and classes visibility and not directly available

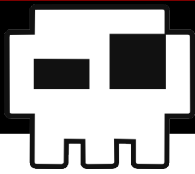
 Is there a way to access a private method from outside its class ?

☞ YAY !



Java Reflection API

- 🤖 See ya in a mirror
 - ☞ Reflection allows introspection and dynamic object manipulation
 - ☞ We can instantiate objects, invoke methods and get/set properties
- 🤖 The Android Java API is full of private stuff not intended to be used as-is
 - ☞ Is there a way to bypass restrictions and/or do some fun stuff ?
- 🤖 Yes, we can make a method public instead of private and use it !

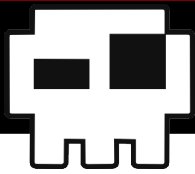


Go deeper and use what you need !

Android's Telephony layer

- ☞ Provides a `SmsManager` class
- ☞ This class contains the `sendTextMessage()` method
 - Can only send Class 1 SMS
- ☞ BUT also contains a *private* method called `sendRawPdu()`
 - Can send SMS in raw mode, with PDU encoding
 - PDU: Protocol Description Unit

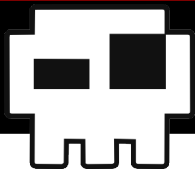
 Some bytes of the PDU-encoded SMS can be altered in order to make it Class 0 SMS =)



Go deeper and use what you need !

SMS PDU format

Offset	Size	Role
0	1	SMSC address size
1	1	Message type
2	1	TP-Message Reference
3	1	Address length (X)
X+3	1	Protocol Identifier (TP-ID)
X+4	1	Data coding scheme (TP-DCS)
...	...	





Go deeper and use what you need !

Data coding scheme

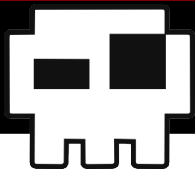
-  Bit 0-1: message class
-  Bit 2: Message coding

To force a PDU-encoded SMS to be Class 0:


-  Set bits 7-4 to 1
-  Set bit 1-0 to 0

TP-DCS byte to F0h is pretty easy

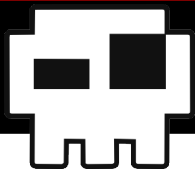
-  8-bit data (instead of 7-bit)



Go deeper and use what you need !

 First, grab a reference on the *sendRawPdu* method:

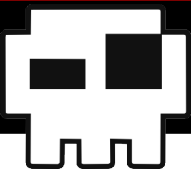
```
byte[] bb = new byte[1];
Method m2 =
SmsManager.class.getDeclaredMethod(
    "sendRawPdu",
    bb.getClass(),
    bb.getClass(),
    PendingIntent.class,
    PendingIntent.class);
```




Go deeper and use what you need !

 Then, make it accessible and use it:

```
m2.setAccessible(true);
SmsMessage.SubmitPdu pdu =
SmsMessage.getSubmitPdu(
    null, PhoneNumber,message,false
);
/* change class to Class 0 */
size = (int)pdu.encodedMessage[2];
size = (size/2) + (size%2);
pdu.encodedMessage[size+5] = 0xF0;
m2.invoke( /* Invoke */
    sm,
    pdu.encodedScAddress,
    pdu.encodedMessage,
    Null,
    null );
```

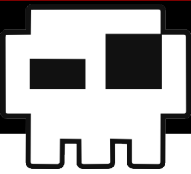


SpyYourWife

-  Instant geolocation app.
-  Class 0 SMS transport layer
-  Geolocation tricks

Conclusion

-  Android, the most awesome mobile phone of the world ?

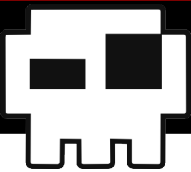


SpyYourWife

- ☞ Proof-of-concept using Class 0 SMS to transfer data between two mobile phones
- ☞ This app. (once installed on a target phone, through USB for instance) react on Class 0 SMS
- ☞ Orders are sent in Class 0 SMSes and intercepted by the app.

Using Class 0 SMS avoid SMS filtering by text

- ☞ False-positive reduction






Geolocation tricks

- ☞ Use only `ACCESS_COARSE_LOCATION`
 - `ACCESS_FINE_LOCATION` requires the GPS location provider
 - `ACCESS_COARSE_LOCATION` will only use Wifi networks and Tower cell ID to locate the phone (less visible)
 - `READ_PHONE_STATE` can help by providing the Cell ID
- ☞ Android keeps track of your location
 - Calling the *`getLastKnownLocation()`* method of Android's *`LocationManager`* allows you to get the last known location for the device
 - Useful when another application requires regular updates

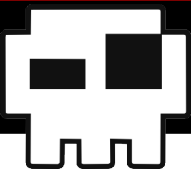


SpyYourWife

-  Instant geolocation app.
-  Class 0 SMS transport layer
-  Geolocation tricks

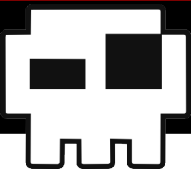
Conclusion

-  Android, the most awesome mobile phone of the world ?



Conclusion

- 🤖 Android users can decrease dramatically the security of their smartphones
 - ☞ They have to evaluate the permissions requested by each application
 - ☞ They have to know exactly what each permission implies
- 🤖 Android's Java API can be hacked through reflection
 - ☞ Dynamic code and access modification
 - ☞ Dynamic instantiation, method invocation, property tampering, etc.



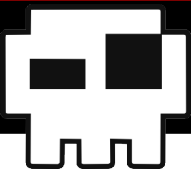
Conclusion

Covert channels

- ☞ They are damned amazing, but are they really useful ?
 - Applications can easily be installed with user's consent
 - Applications run in their own environment, so they cannot be easily monitored

Overt channels

- ☞ Easy way to transfer data through a medium
- ☞ Easily detected, but data can be encrypted to avoid detection
- ☞ A common and good way to leak information from the phone



Conclusion

Actual threats

Malwares

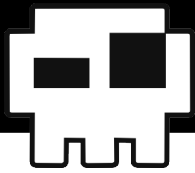
- Constantly growing
- DroidDream case
- Use covert channels to communicate between apps

Trojans

Still easy to drop a trojan on a smartphone

- USB debugging feature
- Social-engineering

Can use overt channels once the application is installed



Questions

Questions ?



Special thanks to

Heurs
@emiliengirault
@adesnos