RSAConference2016
San Francisco | February 29 – March 4 | Moscone Center

Connect to Protect

SESSION ID: MBS-W04

Finding Triggered Malice in Android Apps

Christopher Kruegel
Chief Scientist
Lastline, Inc.

#RSAC

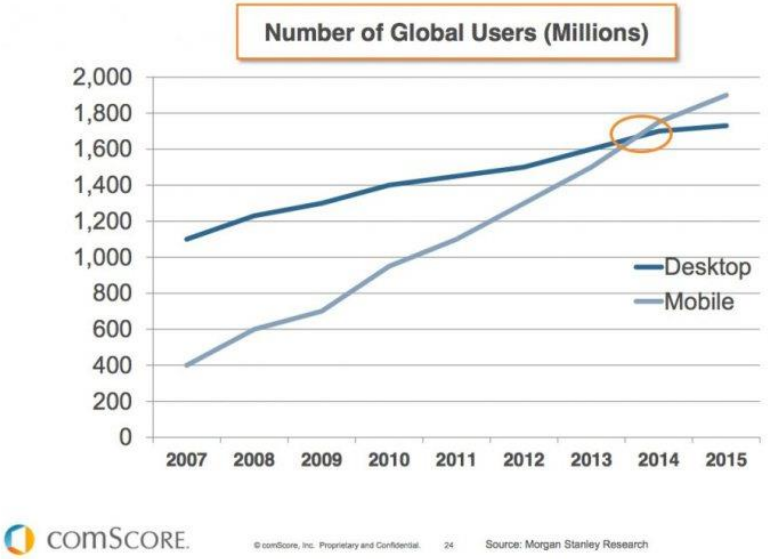
Who am I?

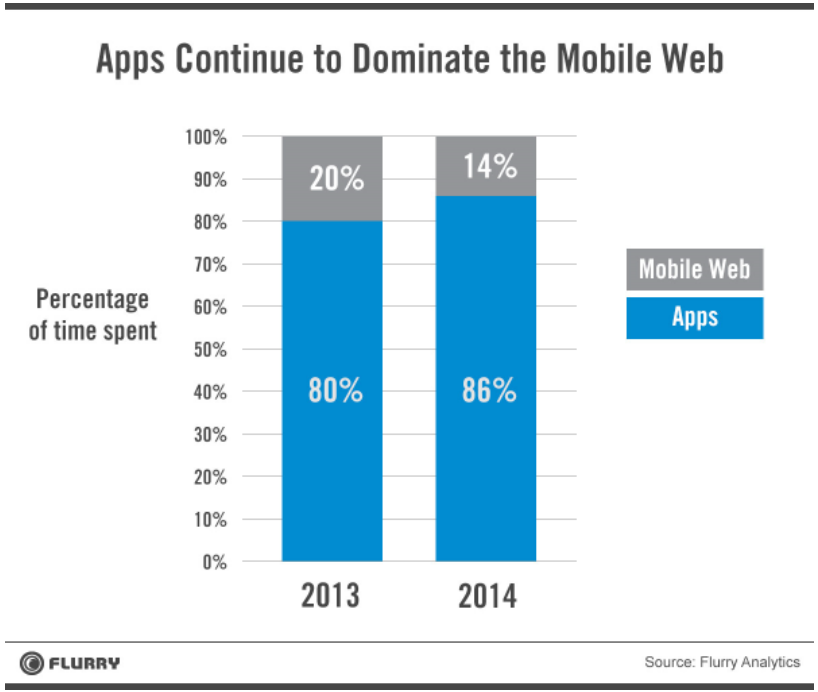
- Professor in Computer Science at UC Santa Barbara
  - many systems security papers in academic conferences
  - started malware research in about 2004
  - built and released practical systems (Anubis, Wepawet, …)
- Co-founder and Chief Scientist at Lastline, Inc.
  - Lastline offers protection against zero-day threats and advanced malware
  - effort to commercialize our research

What are we talking about?

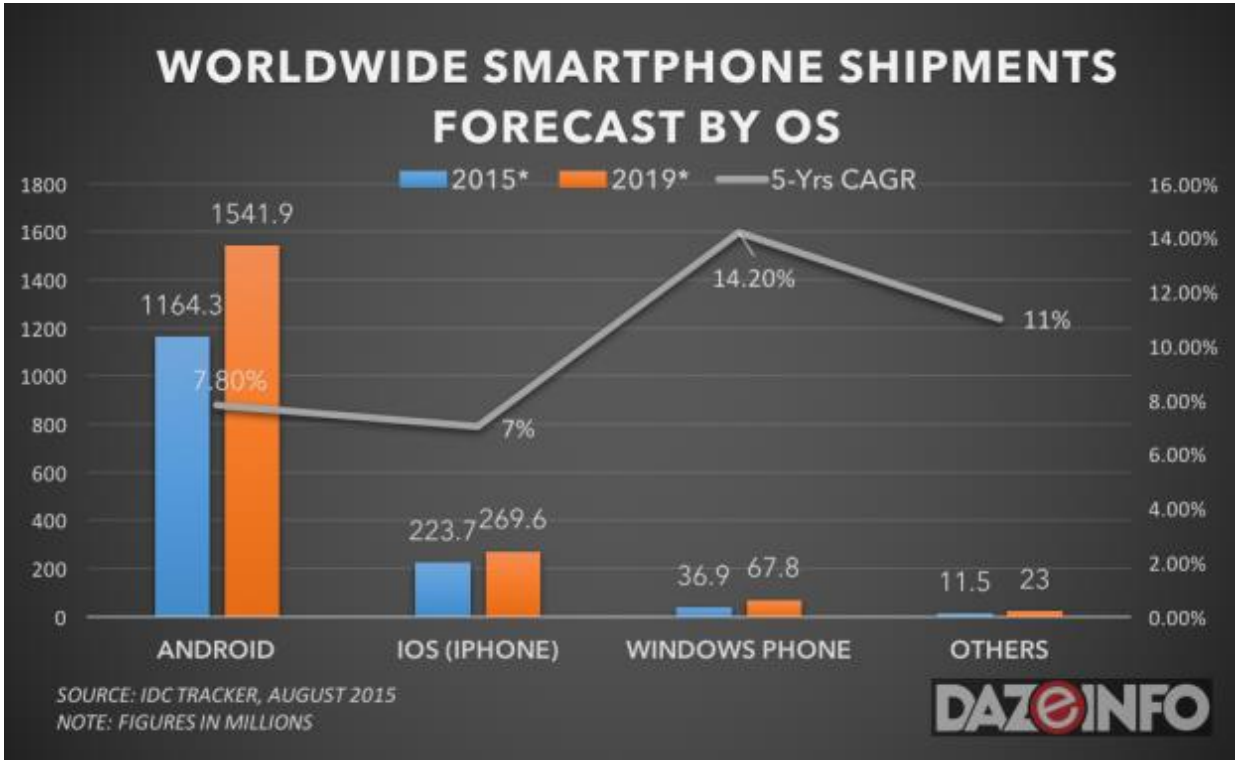
- Android applications (apps) and mobile malware
- Power of static code analysis in the context of Android apps
- Tricky malware that uses triggers to evade detection
- Triggers and triggered malware in the wild

Mobile devices (apps) dominate …





… and Android is leading the pack



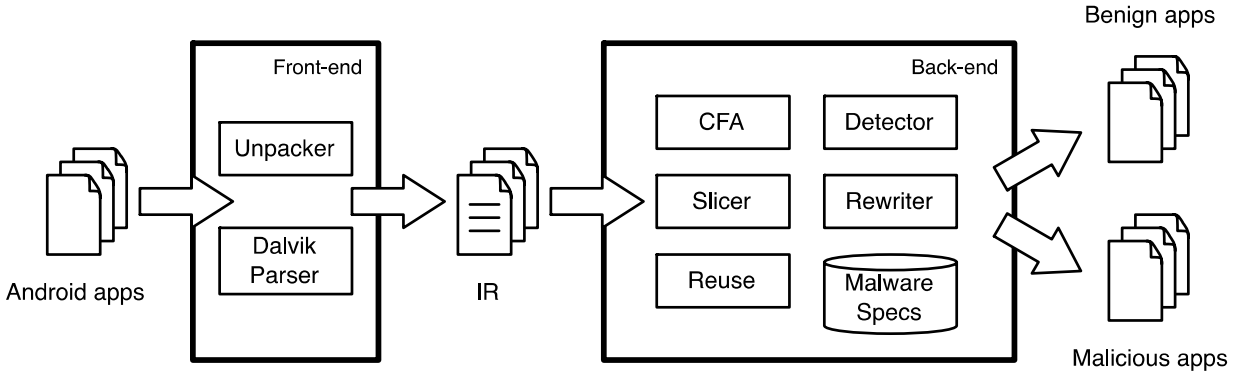
Android Malware on the rise!

- Where are the differences to desktop applications?
  - centralized control
    - vet applications before they enter store
    - can remotely remove installed applications
    - carriers might have more complete picture of users and traffic
  - apps are much easier to analyze statically
    - use of Dalvik bytecode instead of x86
  - interesting GUI issues

Static Analysis for Android Apps

- Precise, scalable static analysis of Android apps
  - Analysis over *bytecode*
  - Minimal *a priori* assumptions regarding app design
  - Cannot ignore difficult cases and adversarial behavior
  - Do not consider apps in isolation (intents, user interaction, …)
  - Scalability from the start

Dalvik Static Analysis



- Front-end
  - APK unpacker
  - DEX parser
    - Androguard, dexlib
  - produces custom IR
- Back-end
  - operates on IR
  - points-to, CFA, backwards slicing,…
  - trigger analysis

Static Analysis

- Tracking data flows is at the core of our analysis
- Analysis must handle not only data flows, but also values
  - apps request permissions, but they are very coarse-grained
  - string, value-range analysis

Static Analysis

- Analysis must handle user interaction, activities, and intents
  - otherwise, data flows can be “broken”
- Analysis must handle complex data structures
  - standard collection classes
  - key-value stores for intent parameters
- Whole system analysis

String Analysis

- Robust and precise string modeling is a fundamental capability
  - many interesting flows are parameterized by strings

    e.g., * ⇝ loadUrl
  - symbolic strings are interesting and problematic
- String constraint solver tailored to common string operations in Android applications
  - append, substring, reverse, charAt, delete, …
  - STP as the backend SMT solver

Implicit Flow Reconstruction

- Android applications written as collections of event-driven or asynchronous components
  - Activity, Service, BroadcastReceiver
  - AsyncTask, Thread, Runnable, Callable
  - requestLocationUpdates ⇝ onLocationChanged
  - takePicture ⇝ onPictureTaken
- Analyzer models implicit flows through framework
  - linking explicit handler registrations is straightforward
  - implicit flow recovery through dynamic analysis

Implicit Flow Recovery

- Identify previously unknown implicit flows
  - increase coverage while preserving data flow precision

- Enabled by combination of instrumentation and Clicker
  - tracing code injected, dynamic analysis driven by Clicker
  - records exit points from application code
  - records entry points into framework
  - trace events binned by thread

Implicit Flow Recovery

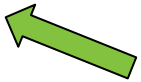
- Several classes of flows discovered
  - explicit callback registration
  - registration of callback groups
  - “implicit” registration

(Some) Interesting Applications

- Detection of triggered malware
- Finding GUI confusion attacks
- Finding dynamic code loading vulnerabilities
- Locate incorrect use of cryptography

(Some) Interesting Applications

- Detection of triggered malware
- Finding GUI confusion attacks



We focus on this one today!

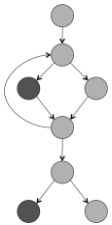
- Finding dynamic code loading vulnerabilities
- Locate incorrect use of cryptography

Evasion and Triggers

- Malware authors are not sleeping
  - they got the news that sandboxes are all the rage now
  - since the code is executed, malware authors have options ..
- Evasion
  - develop code that exhibits no malicious behavior in sandbox, but that infects the intended target

Evasion and Triggers

- One key evasive technique relies on checking for specific values in the environment or inputs – these checks are called triggers
- Other evasive techniques used against sandbox technology
  - exploit limited context
  - avoid analysis (sleep, stalling)
  - avoid analysis (move to kernel, for example, rootkits)



Trigger Analysis

- Data flow captures many important classes of malicious behaviors
  - however, some behaviors are difficult to reason about purely through data flow

1. Analysis to **find interesting checks** that exhibit characteristics of triggering behavior
   - adversaries often want to predicate their attack based on environmental conditions
   - analysis identifies usage of predicates based on environmental data (e.g., location, time, SMS)
2. Then **determine** whether these **checks guard** “interesting” **behaviors**

Find Interesting Checks (1/2)

- Find conditionals that depend on interesting input
  - location, date, time, SMS input …
- Use data flow analysis to determine where interesting input flows to, and what operations the program performs on these inputs
- Propagate constraints on input values along program paths
  - we call these path predicates

Find Interesting Checks (2/2)

- Check for characteristics of triggering behavior
  - checks might be very specific
    - do the predicates significantly constrain the value domain?
    - are there many checks applied to an input?
  - program performs unusual operations on certain inputs
    - do we see unusual operations / operands for certain inputs?

- These are heuristics, and we need to know something about the numbers and types of checks that benign programs perform on certain types of input

Find Interesting Checks

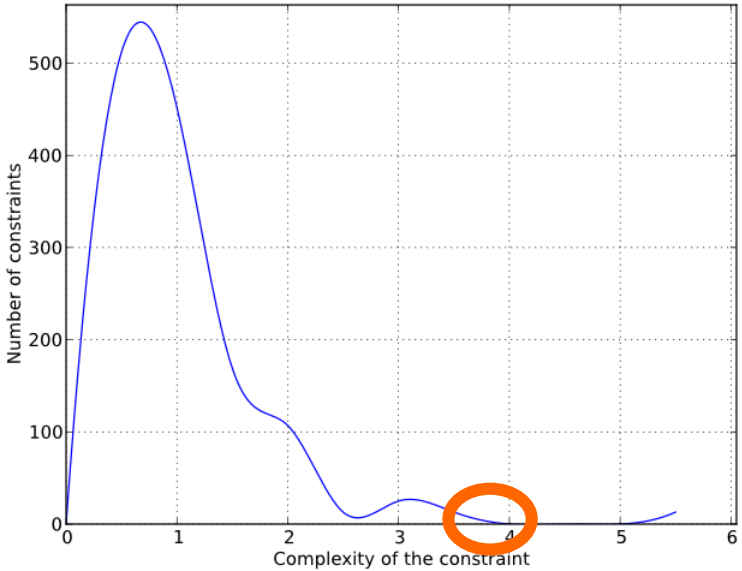
- Example – Malicious application

```
void onCreate(Bundle savedInstanceState) {
    currentTime = new Time(Time.getCurrentTimezone());
    currentTime.setToNow();
    newSiteAvailable =
    (currentTime.month > 4 && currentTime.month < 8) &&
     currentTime.year >= 2013;
```

- How unusual is this?

Find Interesting Checks

- Analysis of 3K apps yields



- Checks in malicious app yield a score of 3.7

Find Checks That Guard Behavior

- Analysis can point human to suspicious checks
  - support manual review
- Maybe we can do better
  - check whether trigger-like check guards sensitive operation
- Easier approach
  - check for sensitive operations directly on path guarded by check
- More complete approach
  - check whether check can influence sensitive operation anywhere in the program

Find Checks That Guard Behavior


```
boolean onOptionsItemSelected(MenuItem item) {
    String updateUrl = "";
    if(newSiteAvailable) {
        url = "http://www.evil.com";
        updateUrl = getUpdatedUrl();
        startIntent(url, updateUrl);
    else {
        startIntent(url);
    }
```

What did we find?

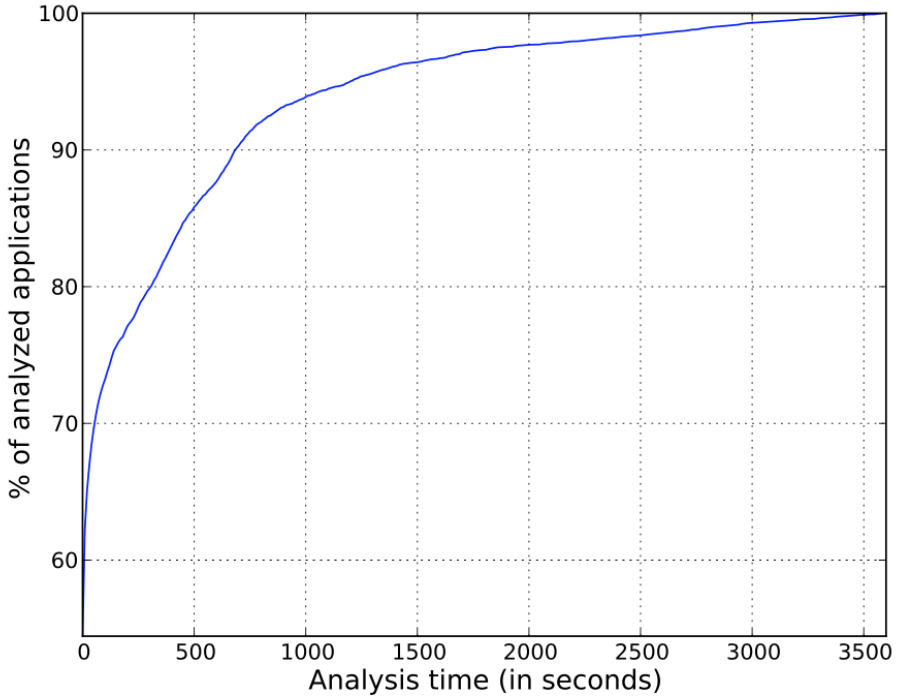
- Dataset of ~10K apps from the Google market
- Collection of triggered malware
  - DARPA apps developed by Red Team
  - Holy Colbert Trojan (backdoor in legitimate app)
  - Zitmo (Zeus-In-The-MOile)
  - RCSAndroid (Hacking Team)

Google Market

Dataset of ~10K apps from the Google market

Domain	# Apps	# Apps w/ Checks	# Suspicious Checks	# Guarded Behavior	# Post-processed
Time	4,950	1,026	302	30	10
Location	3,430	137	71	23	8
SMS	1,138	223	89	64	17

Processing Time



Time Triggers

- Benign triggers
  - to check for updates
  - to implement countdown functionality

Time Triggers

Application Package Name	Time-Triggered Behavior
bvz.commutesms	Automatically sends text messages, customizes their content given the hour of the day.
com.BjrM	Writes different files to disk depending on the day of the week.
com.bigtincan.android.adfree	Checks for expiration date.
com.blogspot.markalcalaramos.android	Automatically sends a SMS 30 seconds after a missed call.
com.ghostleopard.weathermaxlite	Customizes its GUI by selecting different icons depending on the hour of the day.
com.px3j.lso	Checks for expiration date.
com.sivartech.GoogleIO	Notifications for Google I/O events.
com.vesperaNovus.app.StrayPhoneFinderFree	Uses time as a source of randomness, by checking `(1L & System.currentTimeMillis() / 1000L) == 0L`.
com.zyxsolutions.schedulersms	Performs different SMS-related operations depending on the day of the week.
nz.co.mobiledevelopment.ProfileController	Checks for expiration date.

Location Triggers

- Benign triggers
  - based on user-defined areas
  - one interesting trigger for specific Japanese train station

Location Triggers

Application Package Name	Location-Triggered Behavior
com.mv.tdt	Checks the current location against a set of predefined locations.
com.harmanbecker.csi.client	Checks whether the user's location is within a specified area.
net.dotquasar.android.Imakokoroid	Compares the current location with a previously stored location.
com.mv.mobie	Checks the current location against a set of predefined locations.
jp.nekorl.rainnetwork	Checks whether the current latitude is between -90 and 90, and whether the current longitude is between -180 and 180. This is done as a sanity check, and is easy to filter similar cases out if deemed uninteresting.
com.googlecode.androidcells	Compares the current location with a previously stored location.
com.mv.tdtespana	Checks the current location against a set of predefined locations.
jp.co.sha.YamagataMap	Checks whether the device is in the vicinity of Yamagata Station, Japan. If that is the case, the application displays "Welcome to Yamagata Station." The check is implemented by comparing the latitude and longitude against hardcoded values.

SMS Triggers

- Mostly benign triggers
  - apps check for sender, sender's phone, or content to check if they need to process SMS for regular app functionality

- Two suspicious examples found
  - `tw.nicky.LockMyPhoneTrial (RemoteLock)`
  - `com.innovationdroid.myremotephone (MyRemotePhone)`

SMS Triggers

- RemoteLock (removed ~2013 from PlayStore)
  - checks whether the incoming SMS matches with a long, hardcoded string (“adf…yhytdfsw”). If that is the case, the application unlocks the phone!

- MyRemotePhone (still available on PlayStore)
  - checks whether the incoming SMS contains the following two strings: MPS: and gps (now mrp: gps). If that is the case, the application automatically sends an SMS to the original sender containing the current GPS coordinates!

Malware Samples

- DARPA Red Team apps
  - deliberately developed to bypass dynamic analysis
  - 5 time triggers (hardcoded dates and times)
  - 1 location trigger [based on `Location.distanceBetween()`]
  - 5 SMS triggers (contents of messages)
  - malicious activity were data leaks and integrity violations

Malware Samples

- Zitmo
  - checks for content in SMS to steal mTANs for banks
  - SMS used to implement command and control
- Holy Colbert
  - time trigger based on `SimpleDateFormat` API

Malware Samples

- RCSAndroid
  - remote control app written by HackingTeam
  - SMS-based trigger (comparison with values from a file)
  - behaviors range from data leakage (send conversations, device information) to capture of screenshots and voice calls

RCSAndroid

- SMS-based check (comparison with values from a file)

```
public static boolean isInteresting(Sms s, String number, String msg) {
        if (s.getAddress().toLowerCase().endsWith(number) == false) {
                return false;
        }

        // Case insensitive
        if (s.getBody().toLowerCase().startsWith(msg) == false) {
                return false;
        }

        return true;
}
```

RCSAndroid

- Based on the checks, an action is triggered …

```
private final boolean trigger(int actionId) {
        if (actionId != Action.ACTION_NULL) {
                if (Cfg.DEBUG) {
                        Check.log(TAG + " event: " + this + " triggering: " + actionId);
                }

                Status.self().triggerAction(actionId, this);
                return true;
```

(Some) Interesting Applications

- Detection of triggered malware
- Finding GUI confusion attacks
- Finding dynamic code loading vulnerabilities
- Locate incorrect use of cryptography

Dynamic Code Loading

- Apps can load code dynamically at runtime
  - download code from the Internet or local files
  - various ways (DexClassLoader, CreatePackageContext, …)

Dynamic Code Loading Vulnerabilities

- Insecure downloads
  - load code over HTTP
- Unprotected storage
  - downloaded code is stored in location accessible to other apps
- Improper use of package names
  - load code from other apps, specifying only package names

Dynamic Code Loading Vulnerabilities

- Top 50 free apps around end of 2013

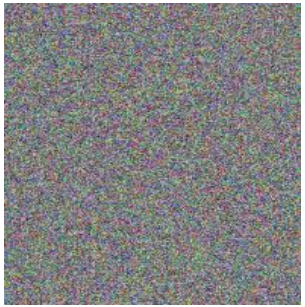
TABLE III. CODE-LOADING IN THE TOP 50 FREE APPLICATIONS AS OF AUGUST 2013. VULNERABILITIES MANUALLY CONFIRMED.

Category	Applications in the category (relative to the whole set)	Flagged vulnerable (relative to the whole set)
Class loaders	17 (34%)	2 (4%)
Package context	0	0
Native code	10 (20%)	0
APK installation	11 (22%)	7 (14%)
Runtime.exec	24 (48%)	n/a
Total	31 (62%)	8 (16%)

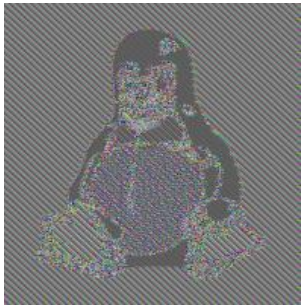
Finding Incorrect Use of Crypto

- Finding apps that use broken crypto
  - developers are not security experts

Plaintext



AES/CBC



AES/ECB

Finding Incorrect Use of Crypto

- Almost 12K apps that use crypto
  - 31% use known key
  - 65% use ECB
  - 16% use known IV for CBC

Lessons Learned

- Understand that mobile threats are real
  - Google Bouncer not enough
- Understand that fighting mobile malware is fundamentally different than fighting Windows malware
  - ask your vendor how they detect mobile malware
  - demand certain protection guarantees
- Vulnerabilities are real and widespread
  - how to manage app installation, patching, … ?

Conclusions

- Smartphones and apps increasingly popular and important
- Interesting differences between apps and traditional desktop programs allow for important security improvements
  - easier static analysis
- Using static analysis to identify triggers and environmental checks in malicious Android apps