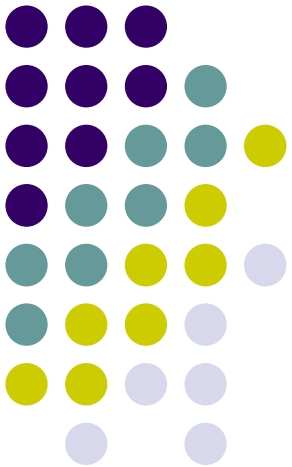
Ubiquitous and Mobile Computing CS 525M: RiskRanker: Scalable and Accurate Zero-day Android Malware Detection

Martti Peltola

Computer Science Dept.

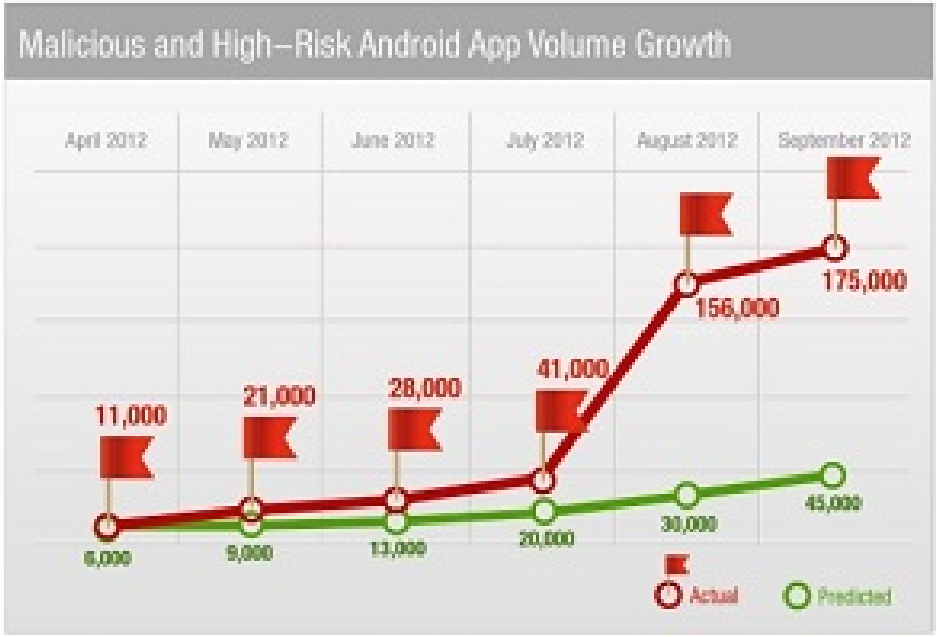
Worcester Polytechnic Institute (WPI)



Introduction/Motivation:

What was the main problem addressed?

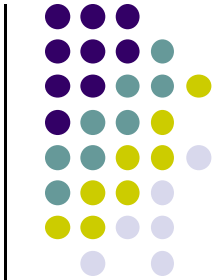
- Mobile devices have become popular targets for malware authors, threatening privacy, security, and finances.
- The growth in the number of malicious and high risk Android apps has far exceeded predictions.



* Note that the numbers in this chart are for all time and not just for this quarter.

Introduction/Motivation:

What was the main problem addressed?



From Wiki:

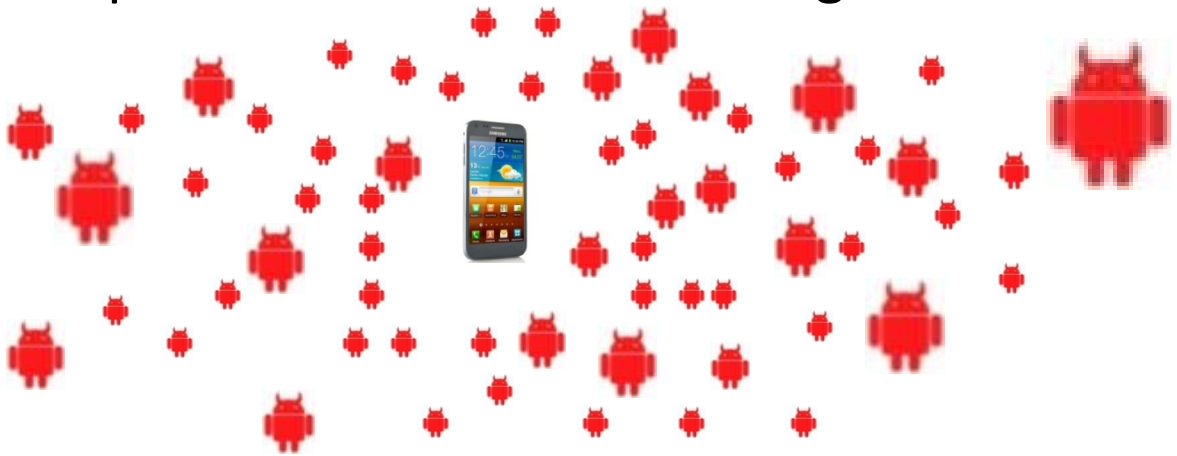
- Traditionally, antivirus software relies upon signatures to identify malware. This can be very effective, but cannot defend against malware unless samples have already been obtained, signatures generated and updates distributed to users. Because of this, signature-based approaches are not effective against zero-day viruses.

- A Zero day virus is a previously unknown computer virus or other malware for which specific antivirus software signatures are not yet available.

Introduction/Motivation:

What was the main problem addressed?

- Traditional signature based ***Reactive*** malware detection is too slow to respond to rate of new threat generation.

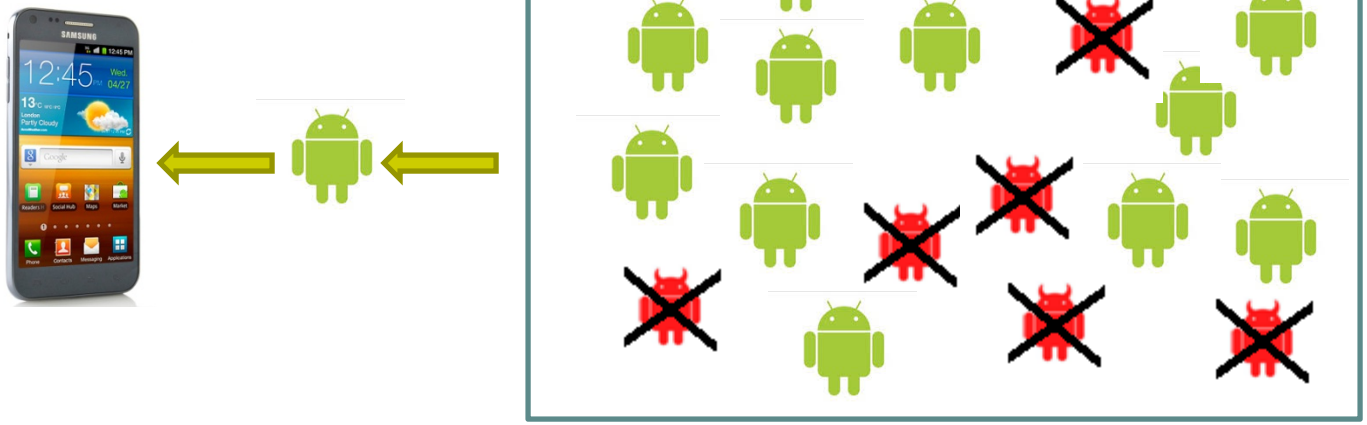


- Malw*a*re is only detected after it has been installed and operating.
- GooglePlay and alternative stores make distributing malware simple. (Apple's review process has prevented approving malware at its store. So far).

Introduction/Motivation:

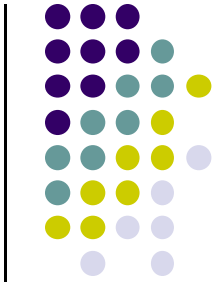
What was the main problem addressed?

- The authors proposed and implemented a ***proactive*** prototype zero-day malware detection scheme, called RiskRanker, which flags malware while it is still at the app store. Store curators can then remove the app from distribution.



Introduction/Motivation:

What will be learned?



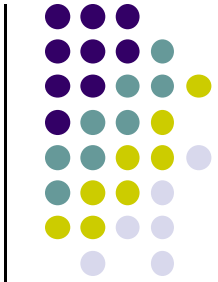
- The authors wish to demonstrate that a proactive approach can detect zero-day malware.

- Their system does not require prior recognition of a new malware threat and updating of remote devices

- Their system can detect the threat while still in the app store

Related Work:

What else has been done to solve this problem?

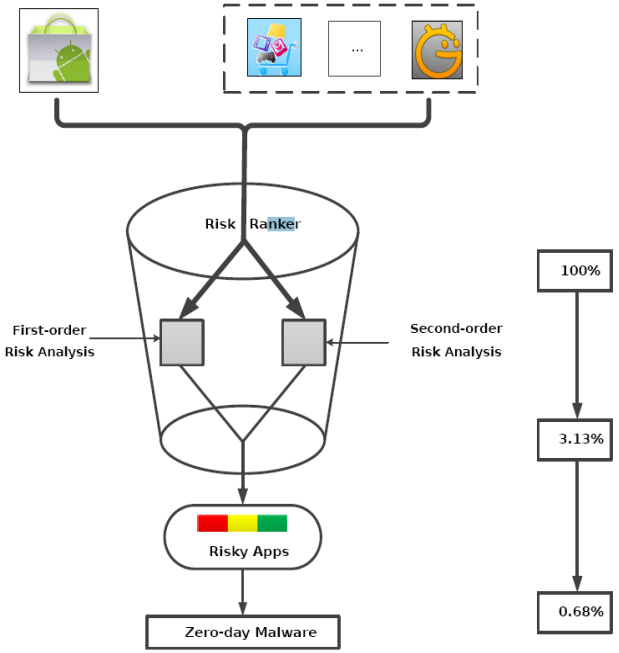


- Reactive malware detection systems
- ProfileDroid (week 10 paper). Detected various system calls, network access, and resource utilization. Could potentially detect malware in operation.
- Stowaway. Static analysis tool to detect over-privilege access in Droid apps.
- TaintDroid. Dynamic analysis to detect information leaks on phones.

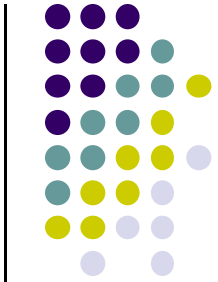
Methodology

Approach and Design

- The prototype system was designed to be scalable, paralizable, and accurately sift through a large number of apps from various Android markets
- The goal was to reduce and rank suspected apps by risk, from a large list to one that was short enough to verify manually.
- RiskRanker sends apps through 2 sets of analyzers:
  - A set of first order modules evaluate straightforward risks (malware not hiding malicious elements)
  - A set of second order modules looks for techniques that indicate malicious code is being purposely hidden from detection.



Methodology
Approach and Design



- First Order analysis
  - Detecting high-risk apps
    - Look for known exploits (see table 1) which leverage platform level vulnerabilities from use of native Android code
    - Can reduce search set by finding presence of native code.
    - Such exploits bypass built in security measures
    - Identify vulnerability specific signature of exploit (Exploid utilizing init daemon).

Table 1: An overview of existing platform-level exploits in Android

Exploit Name	Vulnerable Program	Malware with the Exploit
Asroot [9]	Linux Kernel	Asroot
Exploid [8]	init	DroidDream, zHash DroidKungFu
GingerBreak [29]	vold	GingerMaster
KillingInThe NameOf [2]	ashmem	-
RATC [12] Zimperlich [30]	adbd zygote	DroidDream BaseBridge DroidKungFu DroidDeluxe DroidCoupon
zergRush [20]	libsysutils	-

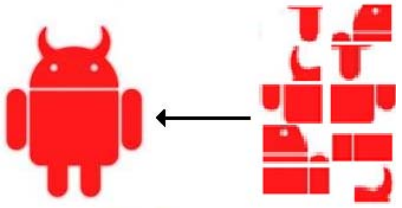
Methodology
Approach and Design

- First Order analysis
  - Detecting medium-risk apps
    - Look for behaviors that result in surreptitious charges
    - Sending SMS to premium phone numbers (and earn malware authors $$$)
    - Need to distinguish legitimate use of such functions
    - Assume legitimate actions initiated actions which invoke callbacks (button press callback)
    - Perform static data-flow and control-flow of Dalvik bytecode to see that money charging operations trace back to callbacks
    - Such analysis complicated by concurrent operations, code obfuscation, reflection, and actions initiated in external threads.
    - Minimize such issues by employing backward and forward slicing.
      - Slicing is technique of identifying reachable code or data

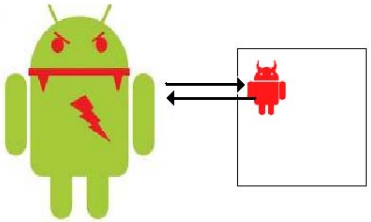
Methodology
Approach and Design

- Second Order analysis - Look for patterns common to malware, and rare in legitimate apps
  - Pre-processing
    - Look for secondary (child) apps within host app.
    - Look for apk or jar files saved in assets or res directories

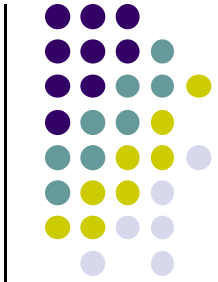
  - Encrypted native code execution
    - Hides signature of malevolent code
    - Look for calls to decryption routines



  - Unsafe Dalvik code loading
    - Load malevolent code at runtime
    - Bypasses discovery during static analysis
    - Look for use of DexClassLoader



Methodology :
Prototyping and Evaluation



- RiskRanker implemented as Linux application, using 3.6 K lines of Python, and 8.7 K of Java.

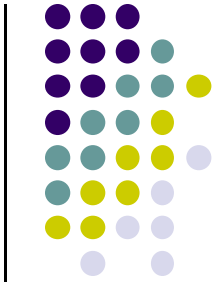
- Contains high-risk root exploit detection module aware of **7** kinds of known exploits (see Table 1, which only shows **6**?).

- Contains medium-risk detection module which scans for
  1. Sending background SMS
  2. Making background phone calls
  3. Uploading call logs
  4. Uploading received SMS messages

- Apps were preprocessed and data placed into MySQL to allow quick indexing and lookup.

Methodology :

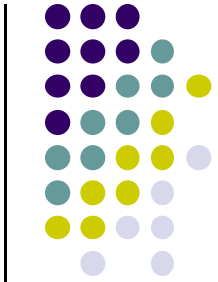
Prototyping and Evaluation



- 118,318 apps were collected over 2 months from GooglePlay (49.8%) and 14 alternate markets. Due to overlap in markets, this came to 104,874 distinct apps
- RiskRanker analyzed apps on a local 5 node cluster, with each node having 8 cores and 8GB of memory
- On this system, RiskRanker could process 3500 apps per hour
- Collected apps were processed in 30 hours.

Results

Verification experiments



- From this collection, RiskRanker identified:
  - 718 malicious apps from 29 malware families
  - Of these, 322 were zero-day from 11 new families
  - First order risk analysis revealed 220 malware samples from 25 families
  - Second order risk analysis found 499 samples from 6 families
  - Summary of results in Table 2

Results

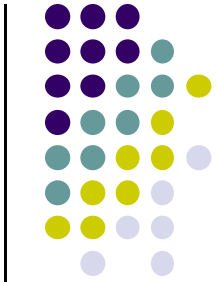
Verification experiments

Table 2: Overall results from RiskRanker

Family	# Samples	Zero-day?	First-Order Risk Analysis		Second-Order Risk Analysis	
			High-Risk	Medium-Risk	Encrypted Native Code Execution	Unsafe Dalvik Code Loading
Androidbox	13			√		
AnserverBot	185	√		√		√
BaseBridge	7		√			
BeanBot	6	√		√		
CoinPirate	1			√		
DogWars	1			√		
DroidCoupon	1	√	√			
DroidDream	2		√			
DroidDreamLight	30			√		
DroidFun	1	√		√		
DroidKungFu1	3				√	
DroidKungFu2	1		√		√	
DroidKungFu3	213				√	
DroidKungFu4	96	√			√	
DroidKungFuSapp	2	√			√	
DroidLive	10	√		√		
DroidStop	7	√		√		
FakePlayer	1			√		
Fjcon	9	√		√		
Geinimi	24		√	√		
GingerMaster	2		√			
GoldDream	21			√		
Kmin	48			√		
Pjapps	17			√		
Pushme	1			√		
RogueLemon	2	√		√		
RuPaidMarket	3			√		
TigerBot	3	√		√		
YZHC	8			√		
Total	718	11	6	20	5	1

Results

Verification experiments



- First Order Analysis Results
  - High-risk apps found (Table 3)

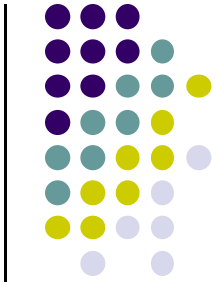
Table 3: Malware discovery in high-risk apps

	Apps with native code	High-risk apps	Actual malware
# of apps	9,877	24	14
Percentage	9.42%	0.02%	0.01%

- 24 High risk apps using known root exploits
- Only 3 Table 1 exploits in use (Exploid, RATC, GingerBreak)
- Found some exploits repackaged into popular legit apps
- System also detected legit (??) jail break tool

Results

Verification experiments



- First Order Analysis Results
  - Medium-risk apps found (Table 4)
    - 2437 apps were found to send background SMS
    - Of these, only 1223 distinct paths
    - A person manually analyzed these in 2 days. Of these, 94 paths were malicious
    - Overall of 2437 medium-risk apps, 206 were infected, including 8 zero-day malware families.

Table 4: Malware discovery in medium-risk apps

Family	# Samples	Zero-day?
Androidbox	13	
AnserverBot	1	√
BeanBot	6	√
CoinPirate	1	
DogWars	1	
DroidDreamLight	30	
DroidFun	1	√
DroidLive	10	√
DroidStop	7	√
FakePlayer	1	
Fjcon	9	√
Geinimi	23	
GoldDream	21	
Kmin	48	
Pjapps	17	
Pushme	1	
RogueLemon	2	√
RuPaidMarket	3	
TigerBot	3	√
YZHC	8	
Total	206	8

Results

Verification experiments

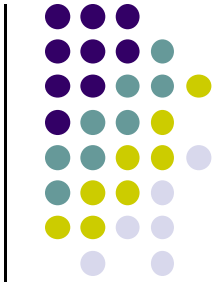


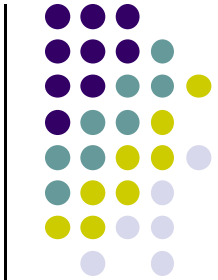
Table 5: Malware discovery in second-order risk analysis

Family	# Samples	Zero-day?
DroidKungFu1 [26]	3	
DroidKungFu2 [25]	1	
DroidKungFu3 [24]	213	
DroidKungFu4 [16]	96	√
DroidKungFuSapp [18]	2	√
AnserverBot [21]	184	√
Total	499	3

- Second Order Analysis Results
  - Medium-risk apps found (Table 5)
    - 328 apps encrypt native code
    - 4257 apps feature dynamic code loading, 1655 contain a child package, there are 492 apps in common.
    - Some are legit. These signatures were white listed to reduce the number of apps being considered.
    - They found one very nasty zero-day app (AnserverBot, removes mobile security software, and other things), and released a security alert.

Results

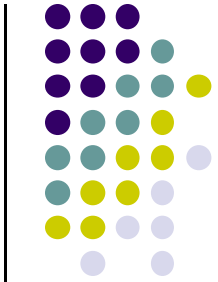
Verification experiments



- False Negative Measurements
  - The authors demonstrated the effectiveness of RiskRanker in finding new (zero-day) threats.
  - They then downloaded a set of known malware from a public contagion repository (exists for research?)
  - After removing duplicates, the set consisted of 133 apps from 31 families.
  - RiskRanker identified 121 of the 133 malevolent apps.
  - Analysis determined that the detection failures were out of scope of the detection modules of RiskRanker.

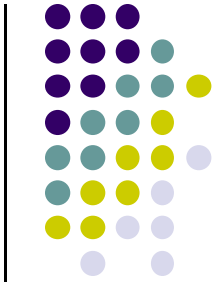
Results

Verification experiments



- Malware Distribution Breakdown:
  - Malware could be detected in all of the markets (GooglePlay included)
  - At one market, 220 apps, or 3% of their offerings were infected.
  - 4 alternative markets offered at least 90 malware apps.
  - Google only had 2 out of 52,208 apps infected. This may be due to adoption of GoogleBouncer (its approach to cleansing their market is unknown)

Discussions/Conclusions/Future Work

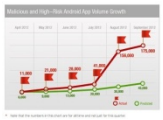


- The authors found that their RiskRanker system detected previously unknown malware from various stores.
- The system, even in a limited prototype stage, was very effective
- They discuss some weaknesses:
  - obfuscation complicates path analysis
  - a malware author could write their own crypto routine, so the known call used for detection is skipped.
  - Can hide malicious code as a large array of innocent looking bytes.

References

- Wiki (http://en.wikipedia.org/wiki/Zero-day_virus)

- http://www.pcmag.com/article2/0,2817,2411240,00.asp , **Trend Micro Warns of Increased Android Malware, Stephanie Mlot,** For malware growth figure and broken android art.

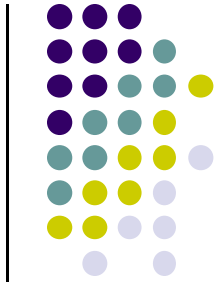


- **RiskRanker: Scalable and Accurate Zero-day Android Malware Detection ,**

  Michael Grace †, Yajin Zhou †, Qiang Zhang , Shihong Zou , Xuxian Jiang †
  †North Carolina State University NQ Mobile Security Research Center
  {mcgrace, yajin zhou, xjiang4}@ncsu.edu {zhangqiang, zoushihong}@nq.com

- **A Survey of Mobile Malware in the Wild,**

  Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steven Hanna, and David Wagner
  University of California, Berkeley
  {apf,finifter,emc,sch,daw}@cs.berkeley.edu



Thank You!

Questions?