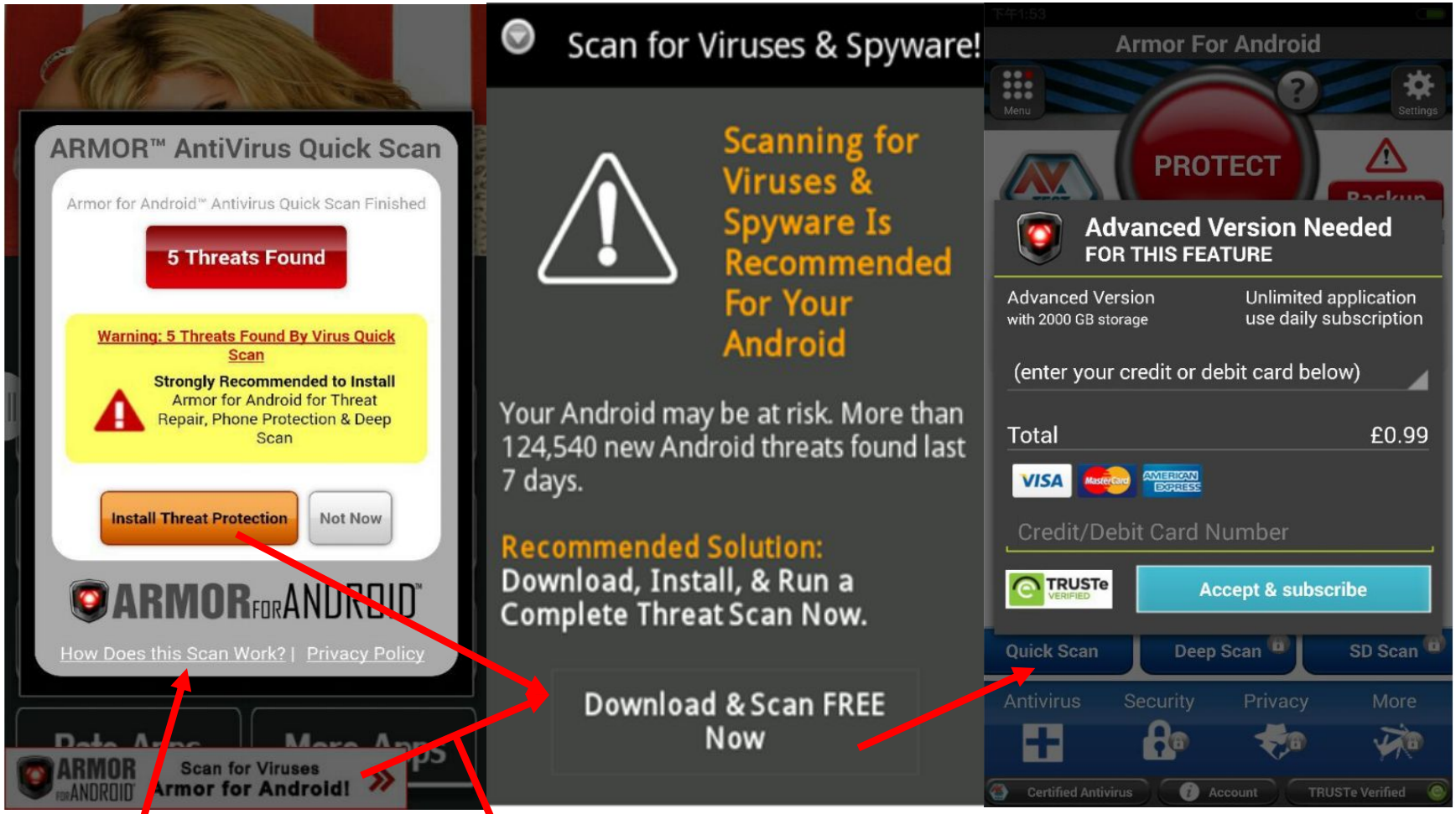
Detecting Hidden Attacks through the Mobile App-Web Interfaces

Yan Chen
Lab of Internet and Security Technology (LIST)
Northwestern University, USA

Motivation



Scan Automatically

Click on the buttons

Downloaded phishing app

Motivation

- Vast effort has been spent analyzing the malicious apps themselves
  - For both industry and academia

- An important, yet unexplored vector of malware propagation is benign, legitimate apps that lead users to websites hosting malicious apps

- We call this hidden attacks though the ***app-web interface***

Contributions

- Develop a framework for analyzing the app-web interfaces in Android applications
- Develop a novel technique to interact with UI widgets to trigger app-web interface
- Conduct a systematic study to associate ad networks with ad library packages
- Detect hidden attacks
  - Tested 600,000 apps in two months
  - Found several unknown attacks: a rogue antivirus scam, free iPad and iPhone scams, and ads propagating SMS trojans

Outline

- Background on mobile advertising
- System Design
- Detection Results
- Case study

Advertising Overview

ADCOLONY

admob

millennialmedia

友盟 UMENG

INMOBI

Advertisers | Ad networks | Apps / Developers | Users

Publishers and Advertisers

- Publishers – show ads to users

Apps / Developers

- Advertisers – the brand owners that wish to advertise

Advertisers

Ad networks

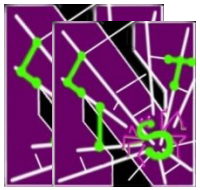
- Also called *aggregators*
- Link advertisers to publishers
- Buy ad space from publishers; sell to advertisers
- Sophisticated algorithms for
  - Targeting
  - Inventory management

Ad networks

- Ad networks may interface with each other
- Syndication
  - One ad network asks another to fill ad space
- Ad exchange
  - Real time auction of ad inventory
  - Bidding from many ad networks for many ad spaces

Mobile In-app Advertising

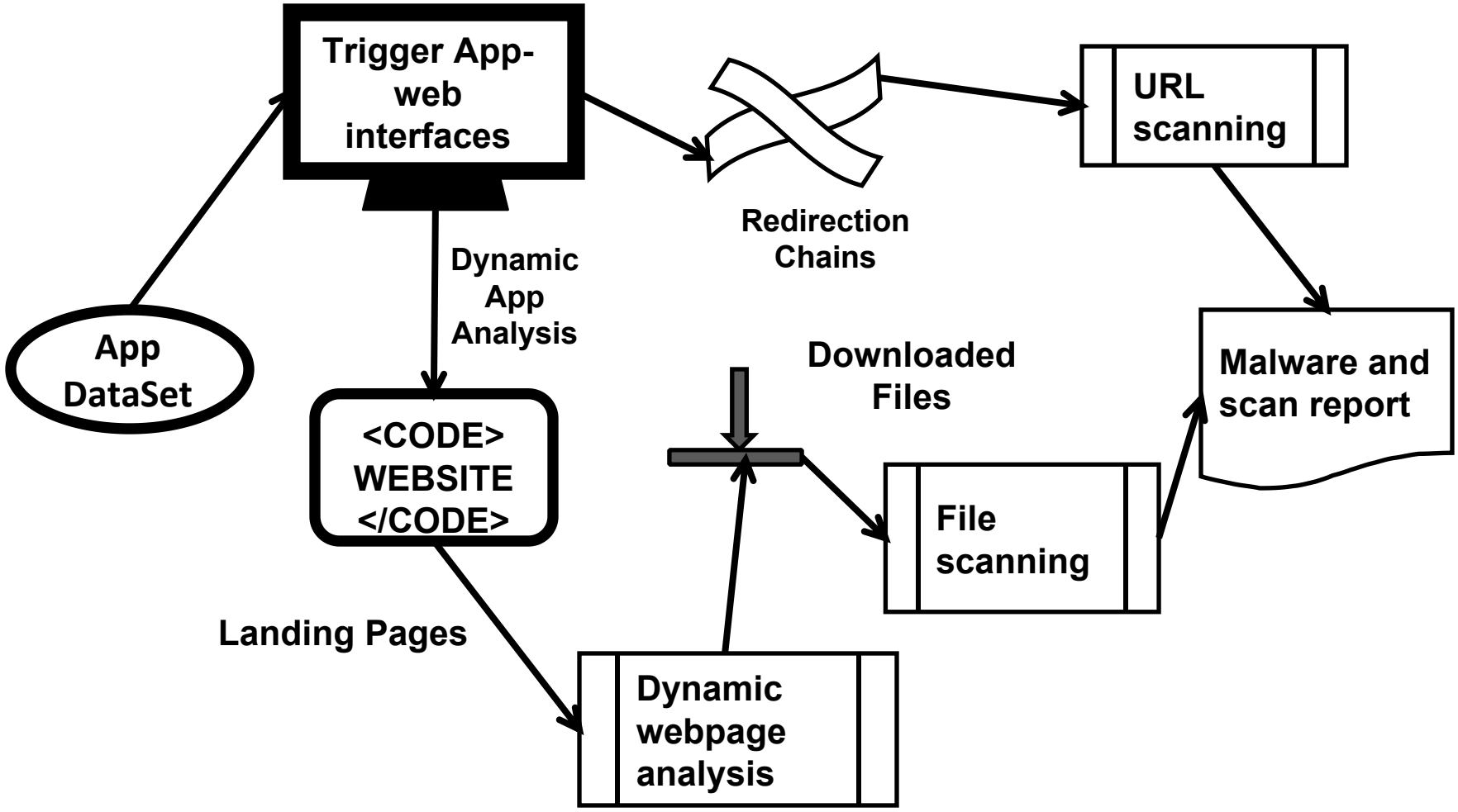
- Ad networks provide glue code that apps can embed and communicate with ad servers
  - Ad libraries, which identify ad networks
- Web links embedded directly in apps
- Malicious links are visited via the landing pages of ads coming from ad networks
  - Though the apps themselves are benign



Outline

- Background on mobile advertising
- System Design
- Detection Results
- Case study

Overview of Detection Methodology



Components

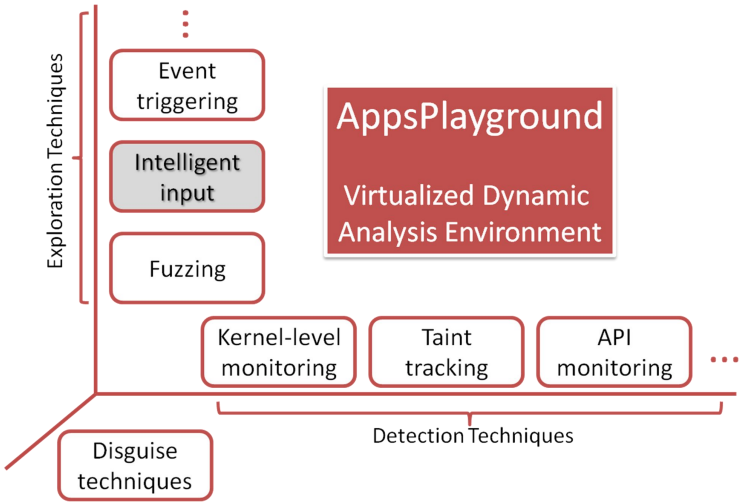
- Triggering
  - Interact with the app to launch web links
- Detection
  - Include the various processes to detect malicious and benig that may occur as a result of triggering
- Provenance
  - Understand the cause or origin of a detected malicious activity, and attribute events to a specific domain or an ad library

Triggering App-Web interfaces

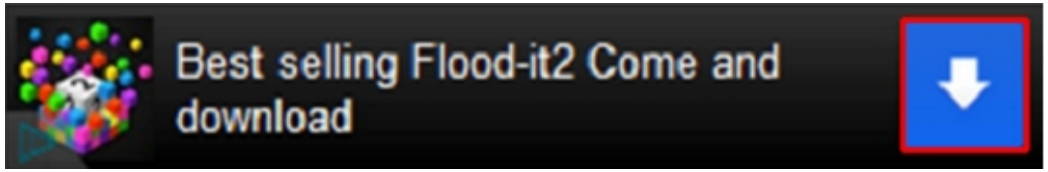
- Application UI Exploration
  - Use the heuristics and algorithms developed in AppsPlayground [Codaspy2013]
- Handling Webviews
  - Develop based on Selendroid to interact with Webviews
  - Apply computer vision techniques

UI Exploration of AppsPlayground

- Fuzzing is good but has limitations
- Another black-box GUI exploration technique
- Capable of filling meaningful text by inferring surrounding context
  - Automatically fill out zip codes, phone # and even login credentials
  - Sometimes increases coverage greatly



Examples of Handling Webviews



- Bounding boxes are depicted as red rectangles.
- The top two figures contain the whole screen while the bottom figure is just an ad.
- Note the detection of buttons.

Detection

- Redirection chains
- Landing pages
  - In a browser configured with a realistic user agent and window size
  - Download any files that can be downloaded
- File and URL scanning
  - VirusTotal URL blacklists
    - Google Safebrowsing, Websense, …
  - VirusTotal antivirus engines
    - Symantec, Dr. Web, Kaspersky, Eset, …

Provenance

- Understand the cause and origins of attacks
- Approach 1: through redirection chains
  - Identify the parties owning the URLs leading up to the landing URL
- Approach 2: attribute code-level elements to locate it: at app or ad libraries?

Discovering Ad Networks

- First systematic step towards understanding malvertising

- Finding ad libraries
  - Typically have their own Java packages, e.g., com.google.ads
  - Disassemble the app and get Java packages

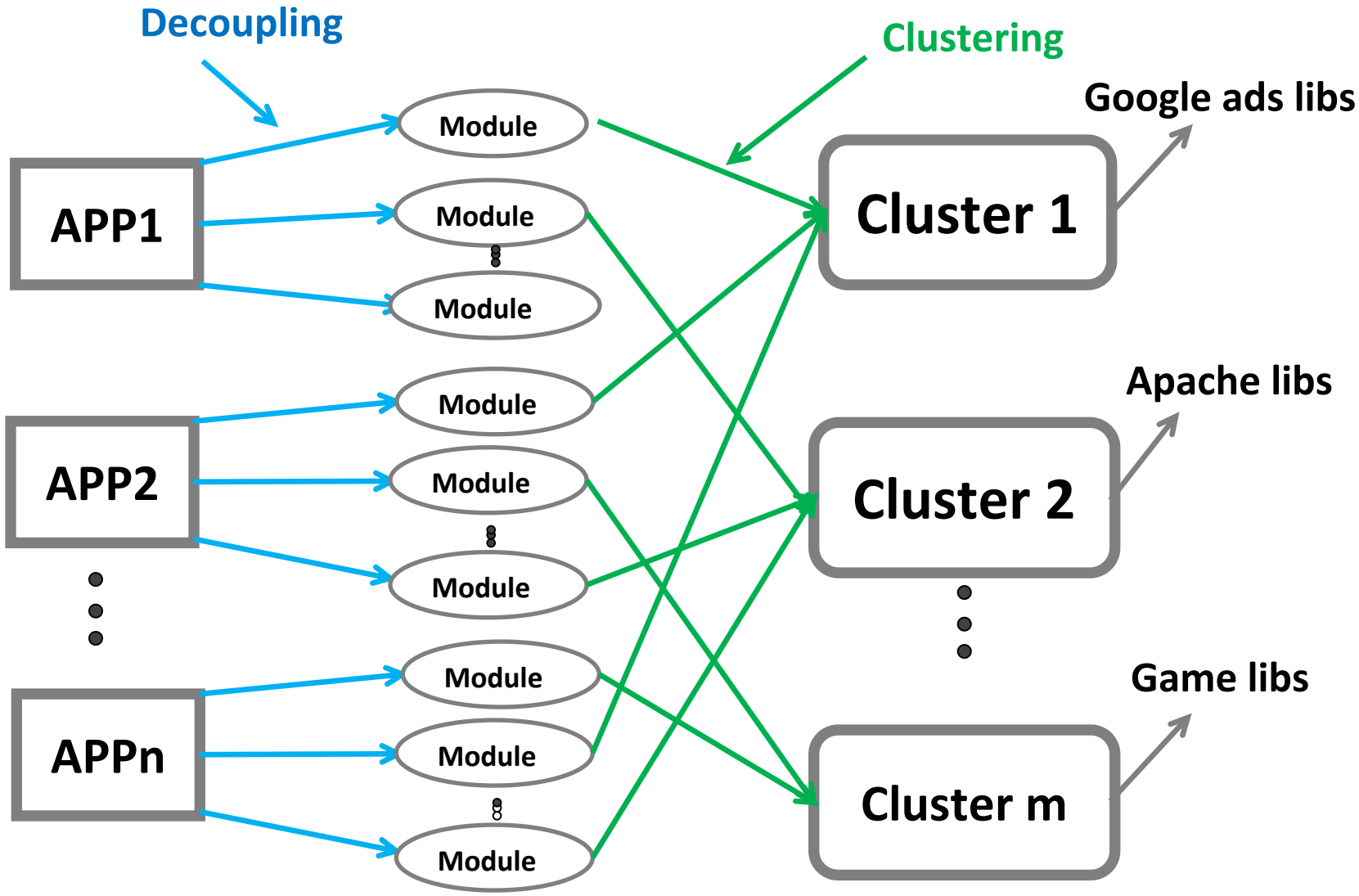
Approach 1

- Find frequent packages
- Ad networks included in many apps so their packages will be frequent
- So are some other packages, e.g., Apache libs, game development libs,…
- Have to manually filter them

Approach 2

- Observation: Ad functionality is different from the main app functionality
- Three steps
  - Get all android APIs
  - Decouple: Break the app into different modules based on code characteristics
    - Inheritance, function calls, field relationships
  - Cluster: cluster modules from multiple apps together based on their API call similarity
    - Frequent libs such as Apache, game libs
    - ad libraries

Approach 2


Decoupling
Clustering
APP1
APP2
APPn
Module
Module
Module
Module
Module
Module
Module
Module
Module
Cluster 1
Cluster 2
Cluster m
Google ads libs
Apache libs
Game libs

Discovering Ad Networks: Results

- Dataset
  - 492,534 apps from Google Play
  - 422,505 apps from four Chinese stores: 91, Anzhi (安智), AppChina(应用汇), Mumayi (木蚂蚁)
- Discovered a total of 201 ad networks
  - The most reported ad networks so far

Outline

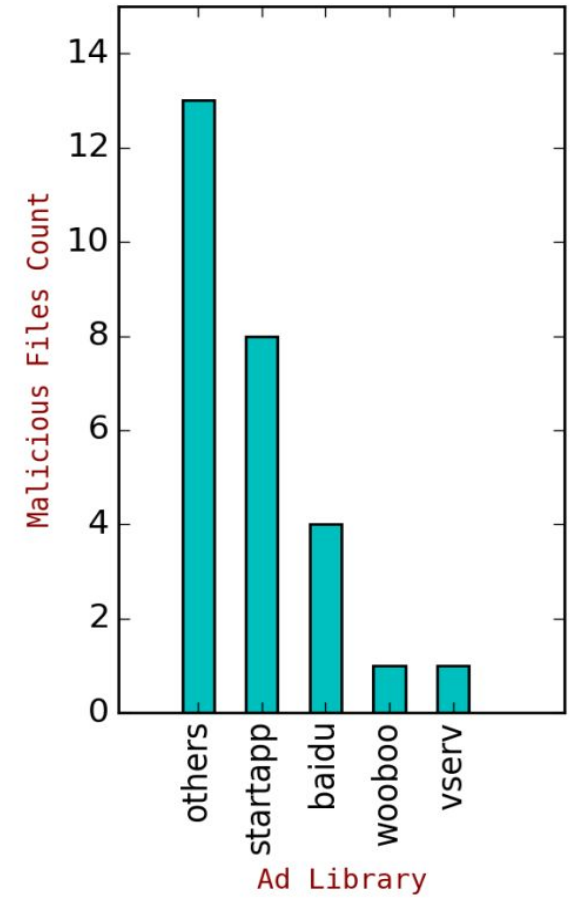
- Background on mobile advertising
- System Design
- Detection Results
- Case study

Overall Detection Findings

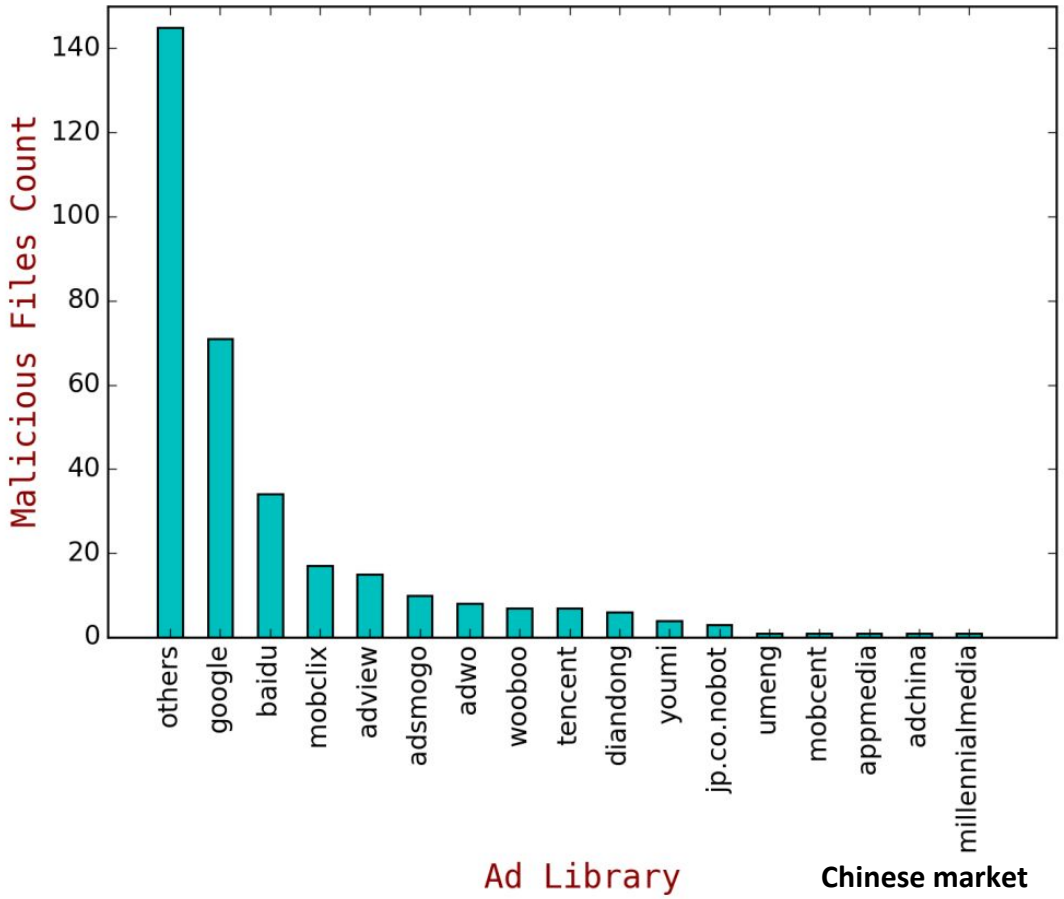
	Google Play	Chinese Markets
App-to-web links	1,000,000	415,000
Malicious URLs	948	1475
Downloaded Files	468	1097
Malicious Downloaded Files	271	435

Run 492,534 apps from Google Play and 200,000 apps from Chinese markets, having ad libraries

Which Ad Libraries Have Attacks



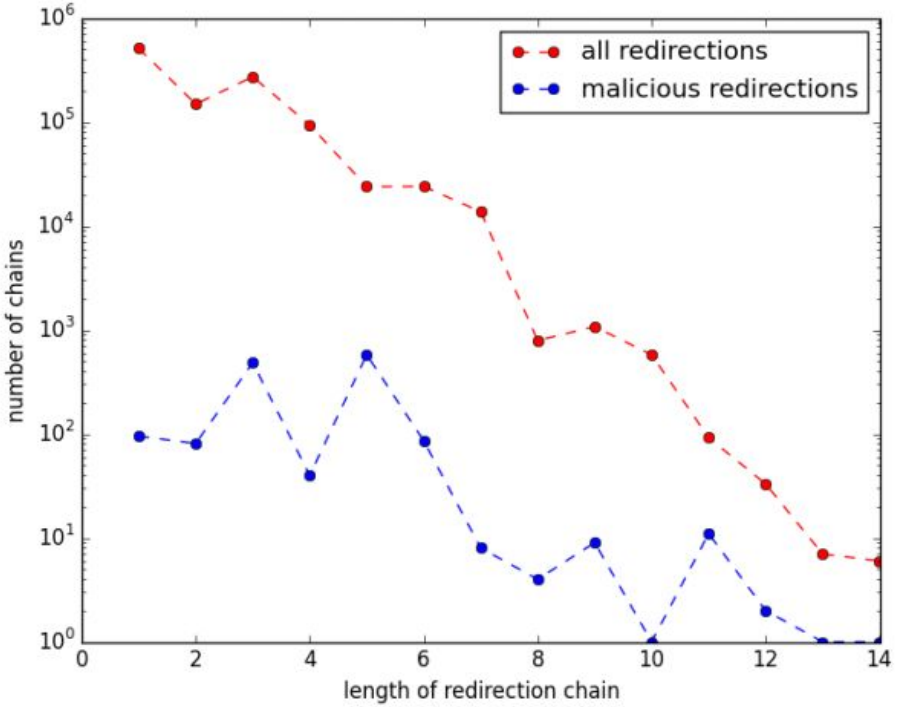
Google Play



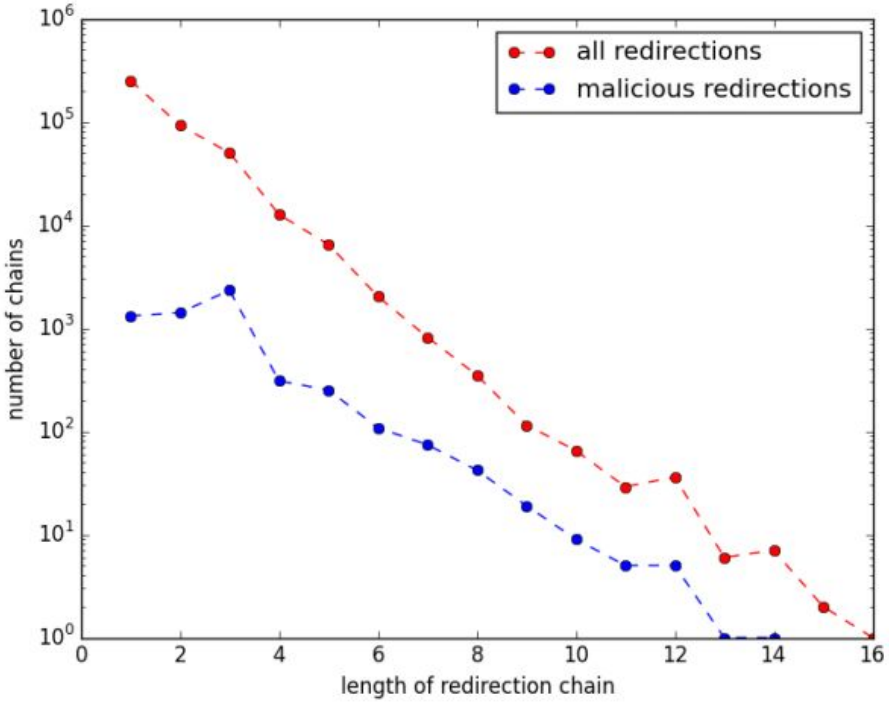
Chinese market

- Malicious files downloaded through ad libraries and other links.
- Tapcontext malware has the most malicious file download, but we exclude them here for better viewing

Comparison on Redirection Chains



Google Play



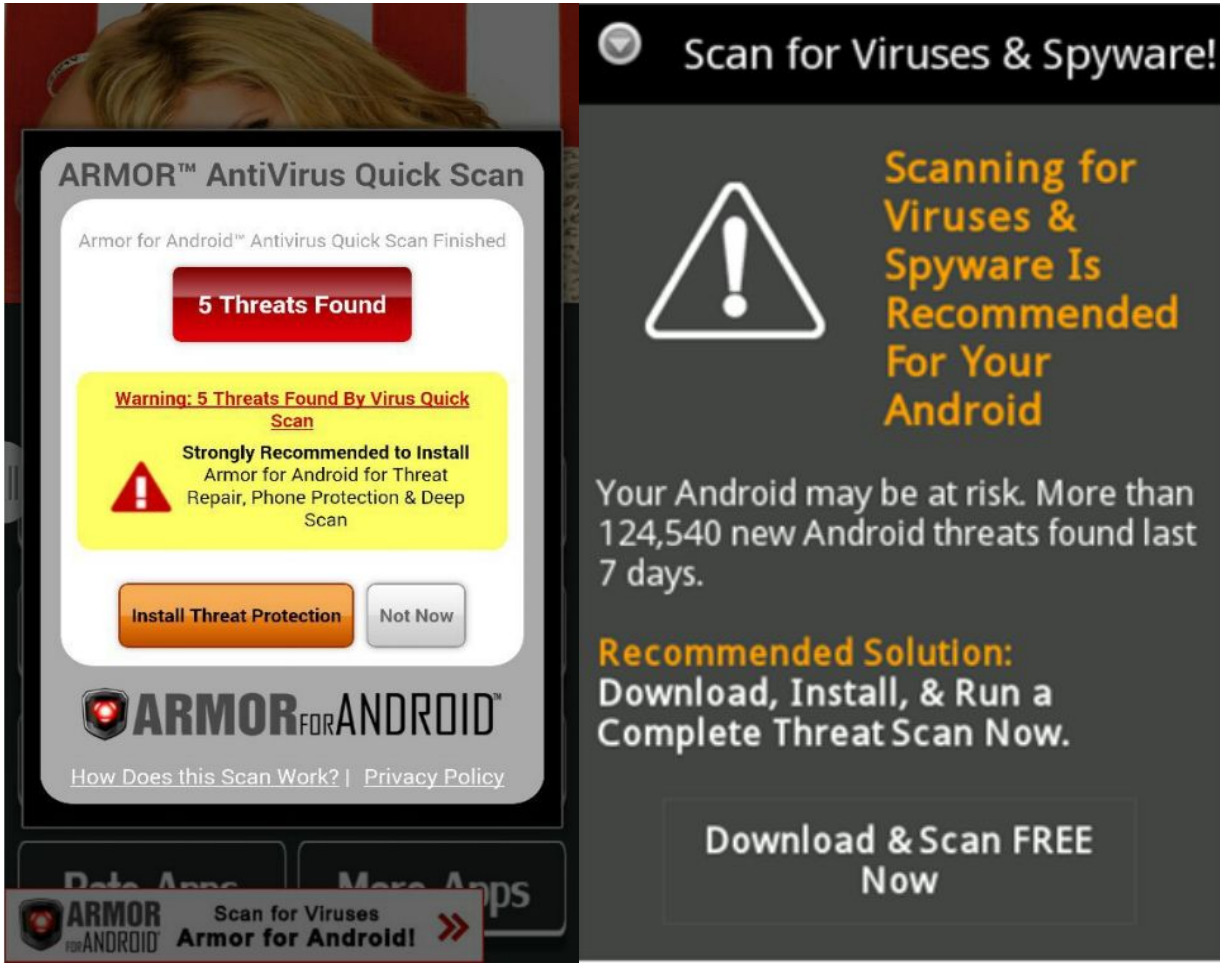
Chinese market

- As the length of the chains increase, the two curves come closer
- We have a greater fraction of malicious chains when they are longer

Outline

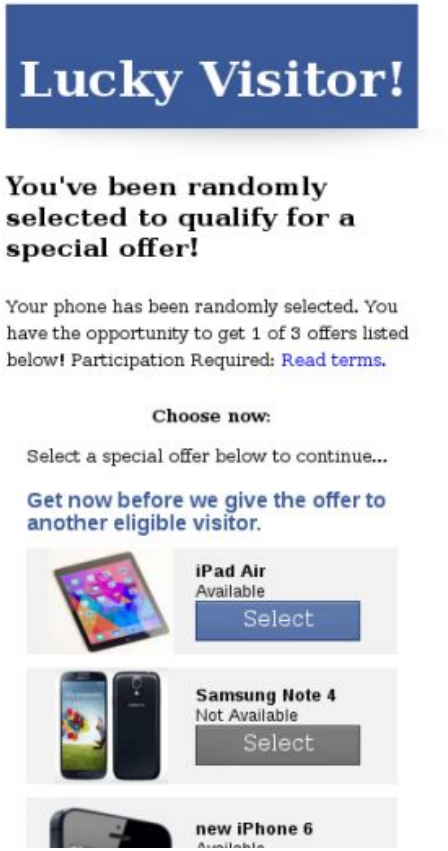
- Background on mobile advertising
- System Design
- Detection Results
- Case study

Case Study:Fake AV scam



- Campaign found in multiple apps, one network: Tapcontext (244 instances in America and 102 in China)
- Website design mimics Android dialog box
- We detected this campaign 20 days before the site was flagged as phishing by Google and others

Case Study:Free iPad scam

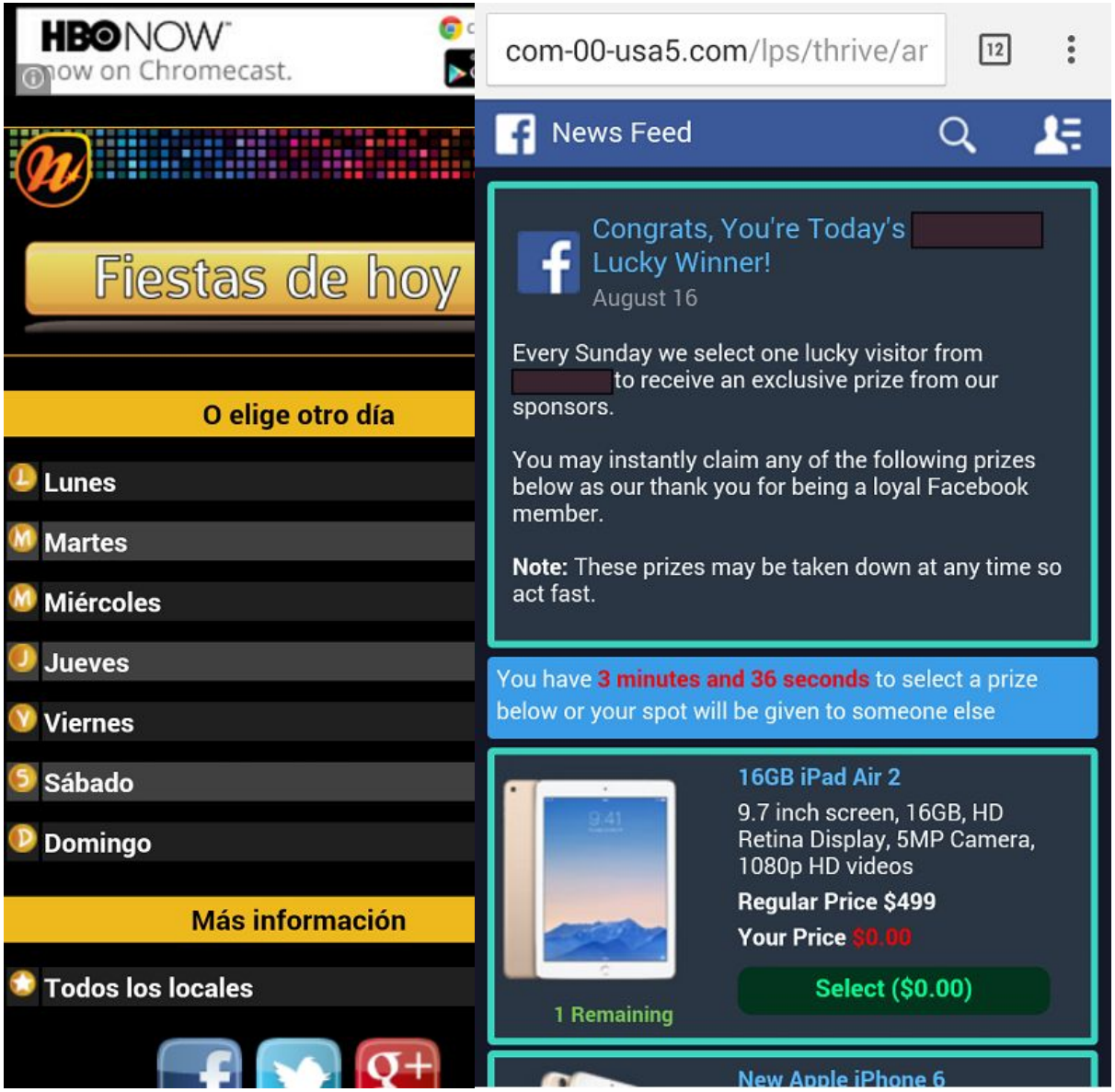


Landing Page

Click on the button

- Phishing: asked to give some very personal information without getting anything in return
- After that, receiving spam on our email address registered with this ad

Case Study:Free iPad scam



- The scam originates not through an ad in the app, but through a link statically embedded in the app.

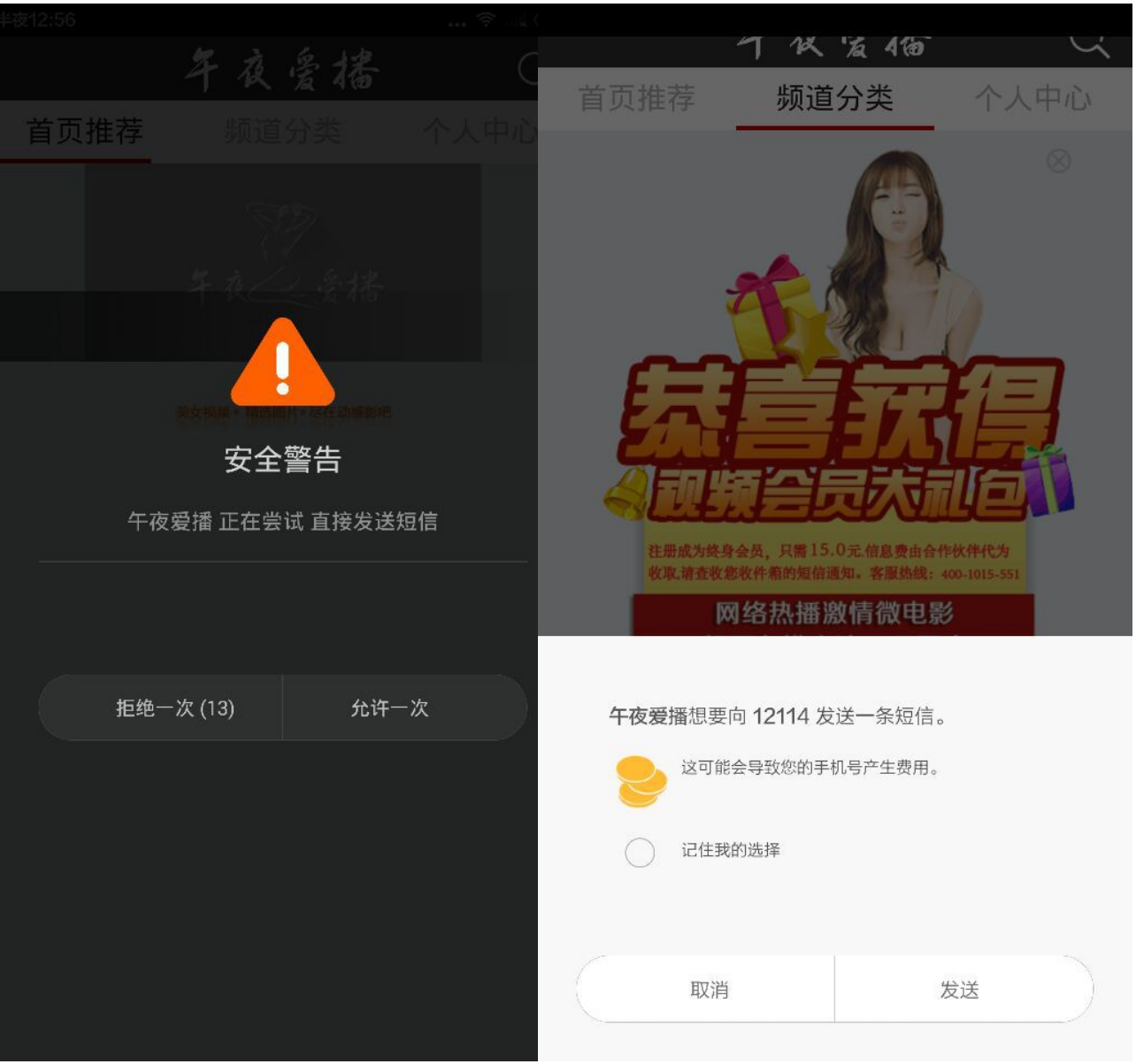
Case Study: Downloaded Player

Click on the ad

Download player

- Ad library name: jp.co.nobot
- It leads to download a video player
- The purported video player is actually an SMS trojan
- Automatically send out paid SMS?

Case Study: SMS Trojan



- When opening the app, it will try to send a message directly
- Once click on the "send" button, it will send message to 12114 (a charged SMS service)
- Receive warning on a Xiaomi phone

Case Study: Porn Phishing

- It also asks you to pay 15 RMB to register a member to see porn content

Case Study: 禁播视频

- Get your location
- Send SMS to 12114
- Download several malicious apps directly
- Some get installed directly without prompt

Conclusions

- Explored the app-web interface, wherein a user may go from an app to a Web destination via ad or web links embedded in the app
- Tested 600,000 applications in two months
- Identified several malware and scam campaigns propagating through both ads and web links in apps.
- We are working with CNCERT to protect Android users
  - by screening out offending apps that embed links leading to malicious content
  - by making ad networks more accountable for their ad content

Thank you !

http://list.zju.edu.cn/
http://list.cs.northwestern.edu/

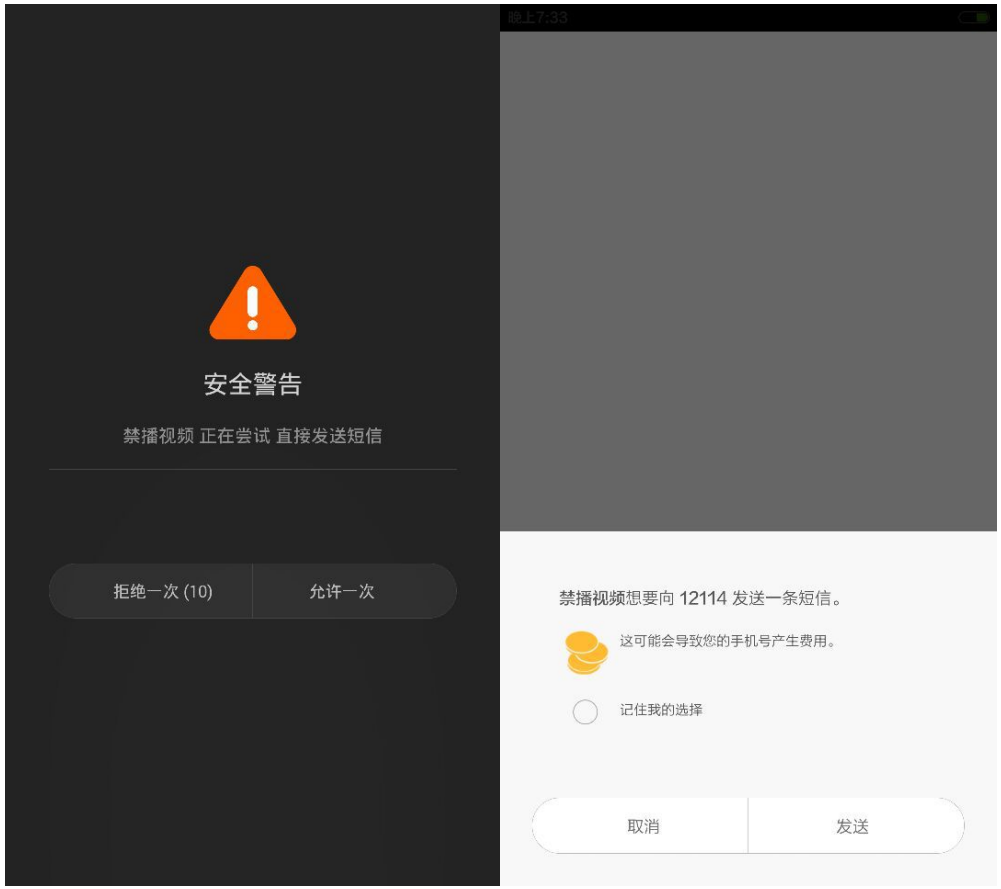
Questions ?

Android Ecosystem

Button Detection Algorithm

a. Perform edge detection on the view's image
b. Find contours in the image
c. Ignore the non-convex contours or those with very small area
d. Compute the bounding boxes of all remaining contours

Case Study:Downloaded App



- Malicious apps downloaded from Baidu ads
- Sent message directly to 12114