DARK HAZARD: *LEARNING-BASED, LARGE-SCALE* DISCOVERY OF HIDDEN SENSITIVE OPERATIONS IN ANDROID APPS

Xiaorui Pan, Xueqiang Wang, Yue Duan[*],
XiaoFeng Wang, Heng Yin[*]
Indiana University Bloomington, [*]University of California Riverside

- **Automated Runtime Analysis**

Symantec.

paloalto NETWORKS®

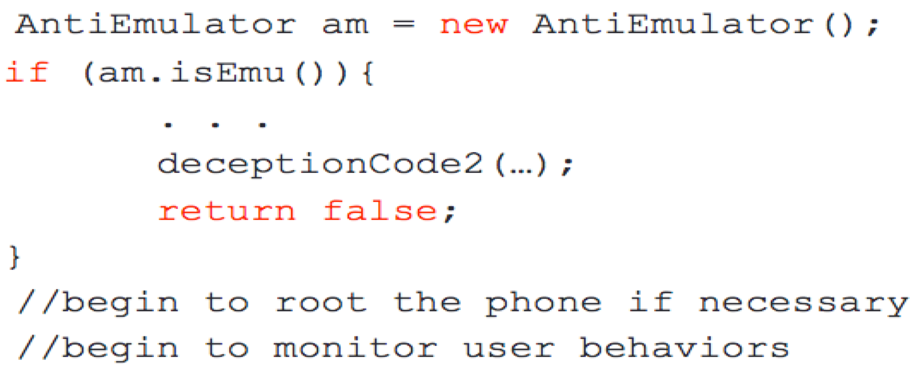
Google

intel® Security

FireEye®

- # The problem?

  - **Hidden Sensitive Operations (HSO):** Malware (or benign) apps conducted ***sensitive operations*** only on certain conditions (trigger) to ***hide*** from automated runtime analysis



```
 AntiEmulator am = new AntiEmulator();
if (am.isEmu()){
          . . .
         deceptionCode2(…);
         return false;
}
 //begin to root the phone if necessary
 //begin to monitor user behaviors
```

Hacking Team Remote Control System

Hidden Sensitive Operations (HSO)

- Anti-emulator
  - QEMU property
  - Performance difference
- Anti-sandbox
  - FireEye Sandbox Profiled
- Logic bomb
  - time, location
- Anti-runtime analysis
  - Determine the absense of a human user

Traditional Approaches

- **Academia solutions**
- Morpheus ACSAC 14
  - High false positive as a detection tool

- TriggerScope S&P 16
  - Precise but *heavyweight*: symbolic execution
  - Need to know the types of trigger in advance
    - Currently limited to time, location, SMS

- **Industry solutions**
  - Signature based
  - *manual* analysis

Our approach

- *Lightweight* program analysis
  - Features based on unique observations
  - Scalability
  - >330K applications
- Semi-supervised learning
- First step towards a more general approach
  - Not limited to certain types of triggers or sensitive operations

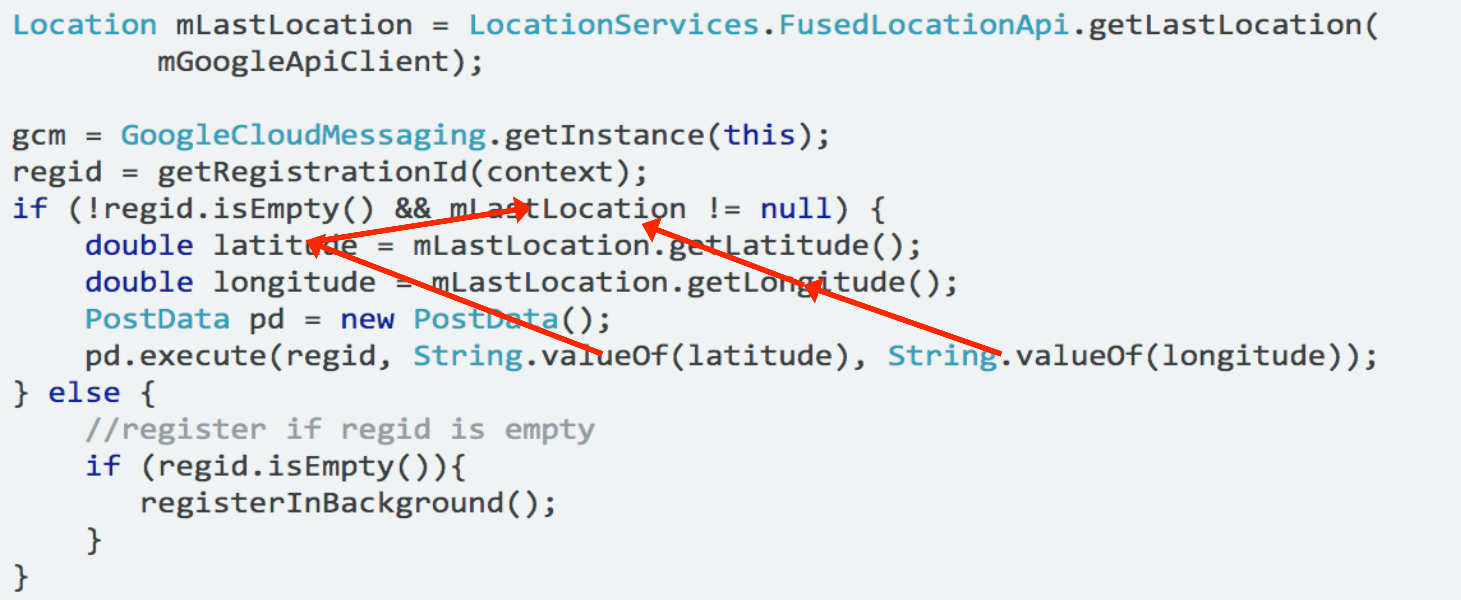
Observations

- *Data and semantic dependency* between conditions and paths in HSO are *weak,*
- Conditions only serve as *guard* of malicious behaviors

```
 AntiEmulator am = new AntiEmulator();
 if (am.isEmu()){
      ...
     deceptionCode2(…);
     return false;
 }

...//begin to root the phone if necessary
...//begin to monitor user behaviors
```

Observations

- Normal case



```
Location mLastLocation = LocationServices.FusedLocationApi.getLastLocation(
        mGoogleApiClient);

gcm = GoogleCloudMessaging.getInstance(this);
regid = getRegistrationId(context);
if (!regid.isEmpty() && mLastLocation != null) {
    double latitude = mLastLocation.getLatitude();
    double longitude = mLastLocation.getLongitude();
    PostData pd = new PostData();
    pd.execute(regid, String.valueOf(latitude), String.valueOf(longitude));
} else {
    //register if regid is empty
    if (regid.isEmpty()){
        registerInBackground();
    }
}
```

Observations (2)

- Behavior difference between two paths
-

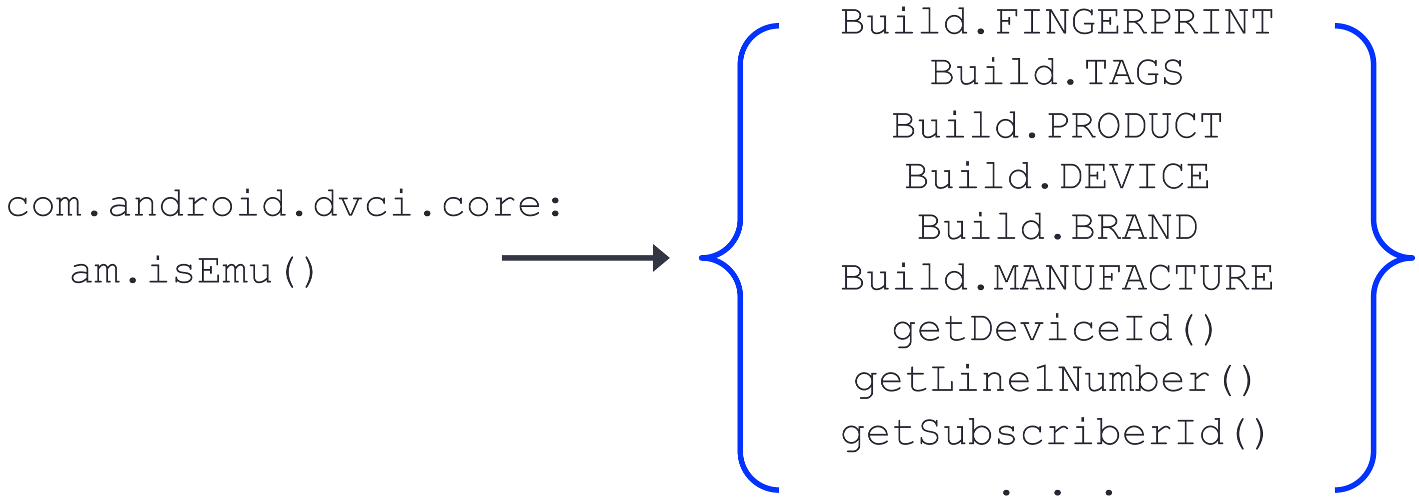
```
AntiEmulator am = new AntiEmulator();
if (am.isEmu()){
        ...
        deceptionCode2(…);
        return false;
}

 ...//begin to root the phone if necessary
```

no sensitive behaviors

root exploit & monitor

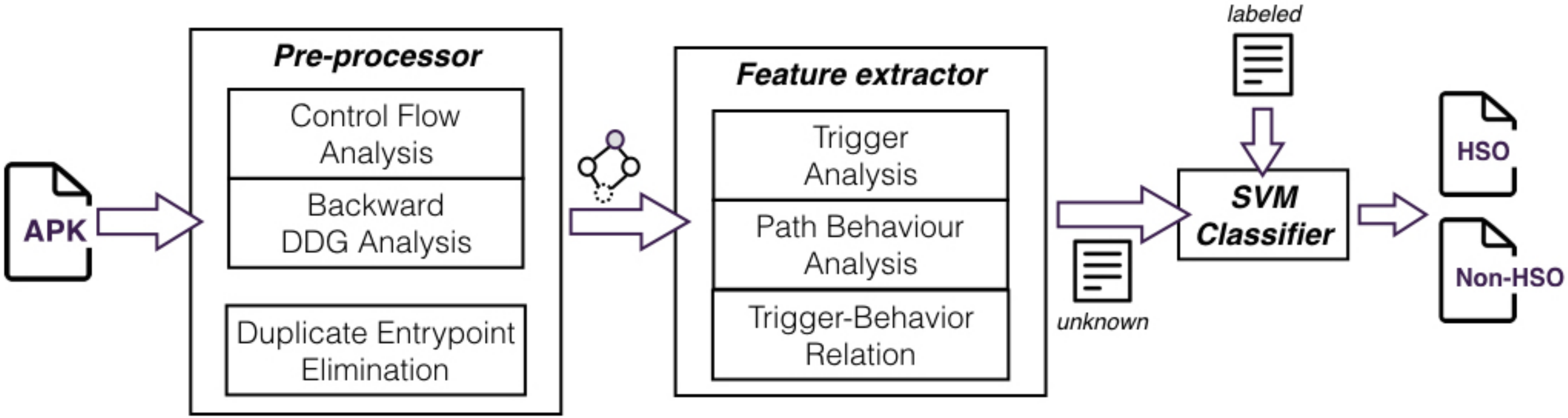
▪ Observations (3)

➢ Source of trigger conditions



```
com.android.dvci.core:
   am.isEmu()          →   {  Build.FINGERPRINT
                              Build.TAGS
                              Build.PRODUCT
                              Build.DEVICE
                              Build.BRAND
                              Build.MANUFACTURE
                              getDeviceId()
                              getLine1Number()
                              getSubscriberId()
                              . . .             }
```

- **Architecture**



Features

- Data and semantic dependency between Condition and Paths
  - Data Dependency (DF_1 DF_2) : k/n
  - Semantic relevance: Implicit Relation (IR_1 IR_2)
    - Based on semantic relevance
    - And Frequency Analysis

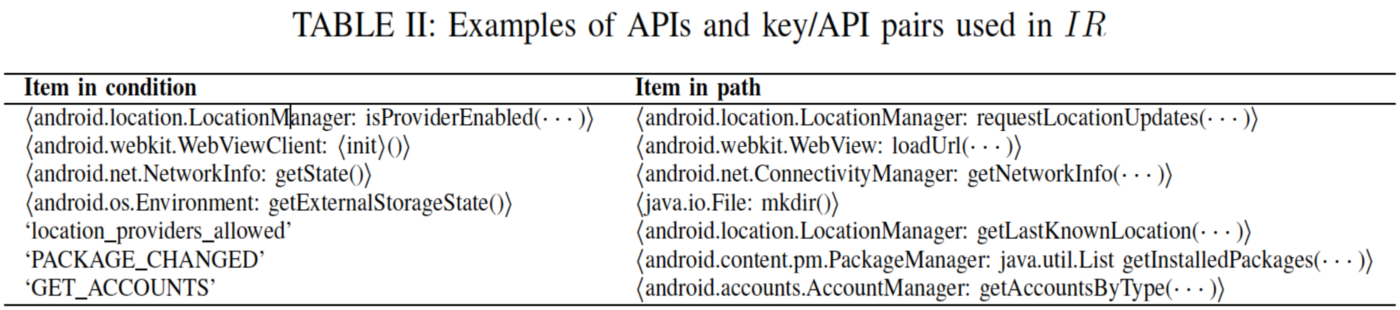
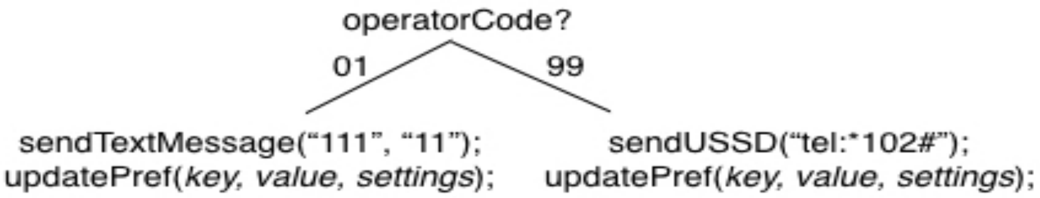


TABLE II: Examples of APIs and key/API pairs used in IR

Item in condition	Item in path
⟨android.location.LocationManager: isProviderEnabled(· · ·)⟩	⟨android.location.LocationManager: requestLocationUpdates(· · ·)⟩
⟨android.webkit.WebViewClient: ⟨init⟩()⟩	⟨android.webkit.WebView: loadUrl(· · ·)⟩
⟨android.net.NetworkInfo: getState()⟩	⟨android.net.ConnectivityManager: getNetworkInfo(· · ·)⟩
⟨android.os.Environment: getExternalStorageState()⟩	⟨java.io.File: mkdir()⟩
'location_providers_allowed'	⟨android.location.LocationManager: getLastKnownLocation(· · ·)⟩
'PACKAGE_CHANGED'	⟨android.content.pm.PackageManager: java.util.List getInstalledPackages(· · ·)⟩
'GET_ACCOUNTS'	⟨android.accounts.AccountManager: getAccountsByType(· · ·)⟩

Features

- Behavior Differences
  - Data distance (DD)



```
sendTextMessage("111", "11");
updatePref(key, value, settings);
```

```
sendUSSD("tel:*102#");
updatePref(key, value, settings);
```

- We also want to know data relations between two paths

$$DD = 1 - \frac{1}{2}\left(\frac{V_l \cap V_r}{V_l \cup V_r} + \frac{F_l \cap F_r}{F_l \cup F_r}\right)$$

Features

- Behavior differences
  - Activity distance (AD)
  - Group APIs or system keys based upon similarity of their functionalities
    - Android official documentation
    - Pscout
    - DroidSIFT
    - other system properties & settings.
  - Jaccard distance

Features

- Source of trigger conditions
  - SI (System input)
  - System properties (OS or hardware traces of a mobile device) or environment parameters (time, locations, user inputs, etc.)
  - SUSI

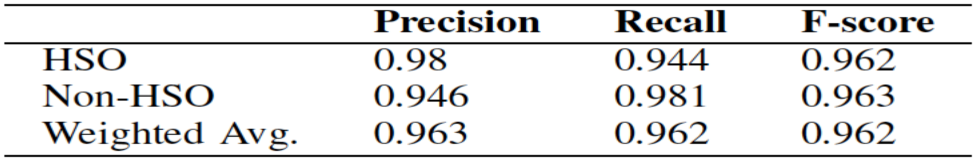
Dataset

- **Ground Truth:**
  - One HSO branch in each of 213 malwares
    - Found by known HSO trigger signatures
  - Non-HSO branches in 213 benign apps
    - Manual confirm and VirusTotal scan

- **Unknown Apps from the wild**
  - 124,207 Google Play Apps
  - 214,147 VirusTotal Apps

Evaluation

- Ground Truth
  - Cross-validation



	Precision	Recall	F-score
HSO	0.98	0.944	0.962
Non-HSO	0.946	0.981	0.963
Weighted Avg.	0.963	0.962	0.962

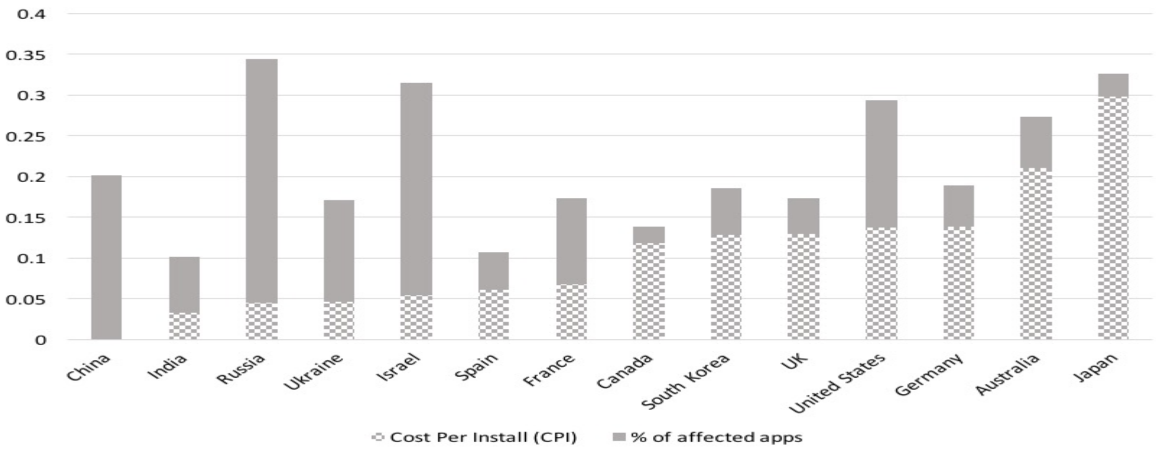
- Apps in the wild
  - Random Sampling
  - Precision: 98.4%
  - Recall not available

Performance

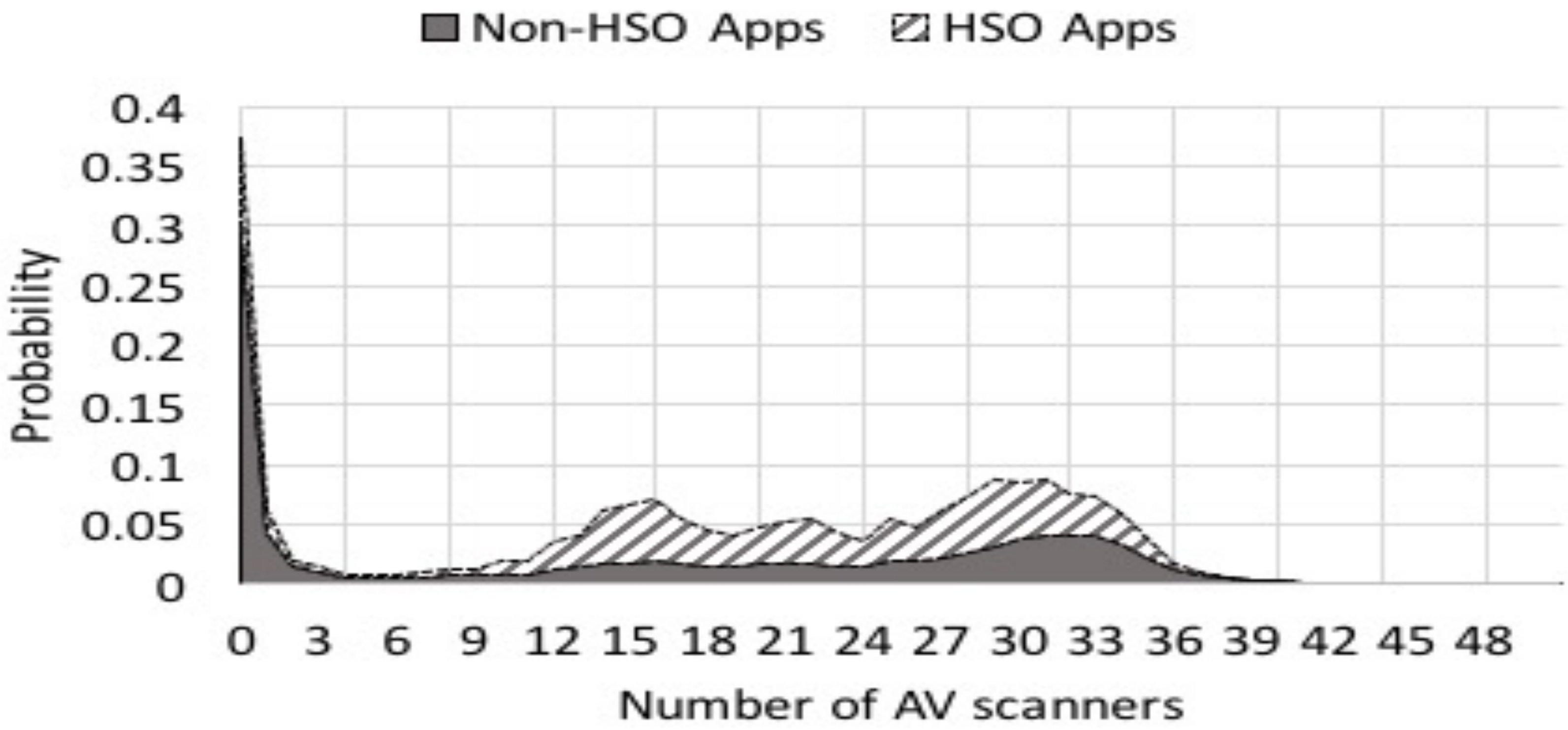
- Random 3000 apps from Google-play average size of 8.43MB
  - 765.3 s per app
  - Dell desktop with 3.3GHz i5 processor and 16GB RAM
  - Timeout: 60 mins
  - 8.4% timeout
- Compared with TriggerScope
  - 5.2 times faster, on **their** dataset
  - 35 apps which is publicly available
  - 42.0 s *VS* 219.2 s

Understanding HSO

- Landscape
  - Overall, 63,372 (18.7%) of 338,354 contain HSO
  - 3,491 unique HSO instances

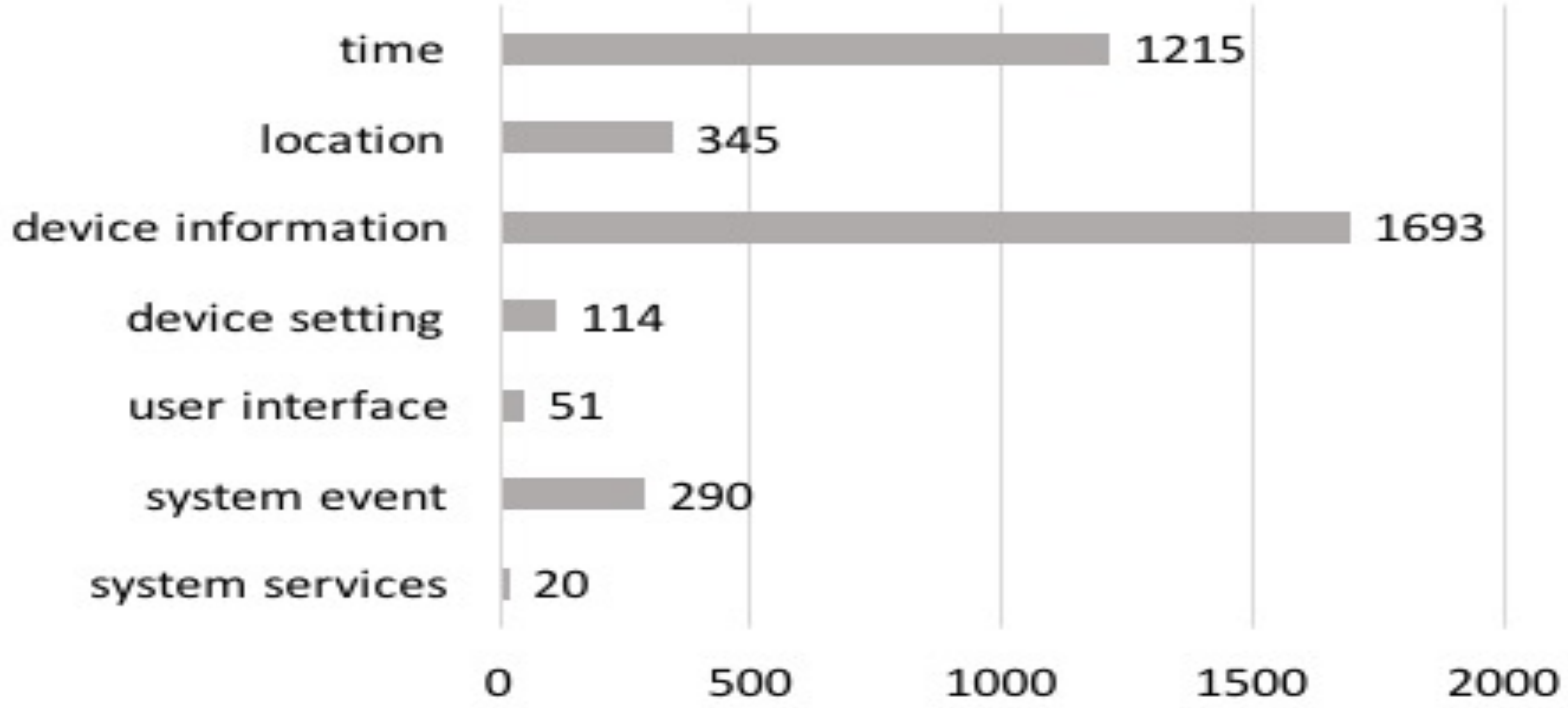


- **HSO and PHA**

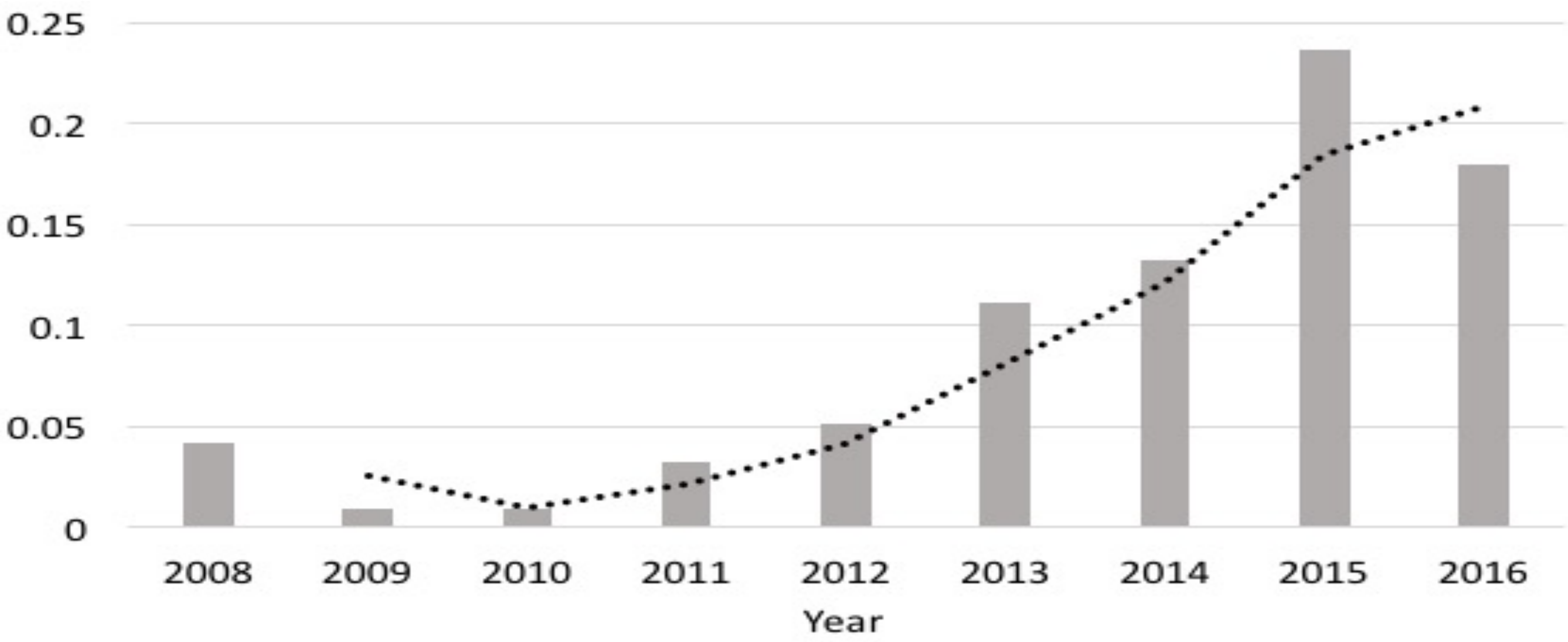


■ Non-HSO Apps
▨ HSO Apps
Probability
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48
Number of AV scanners

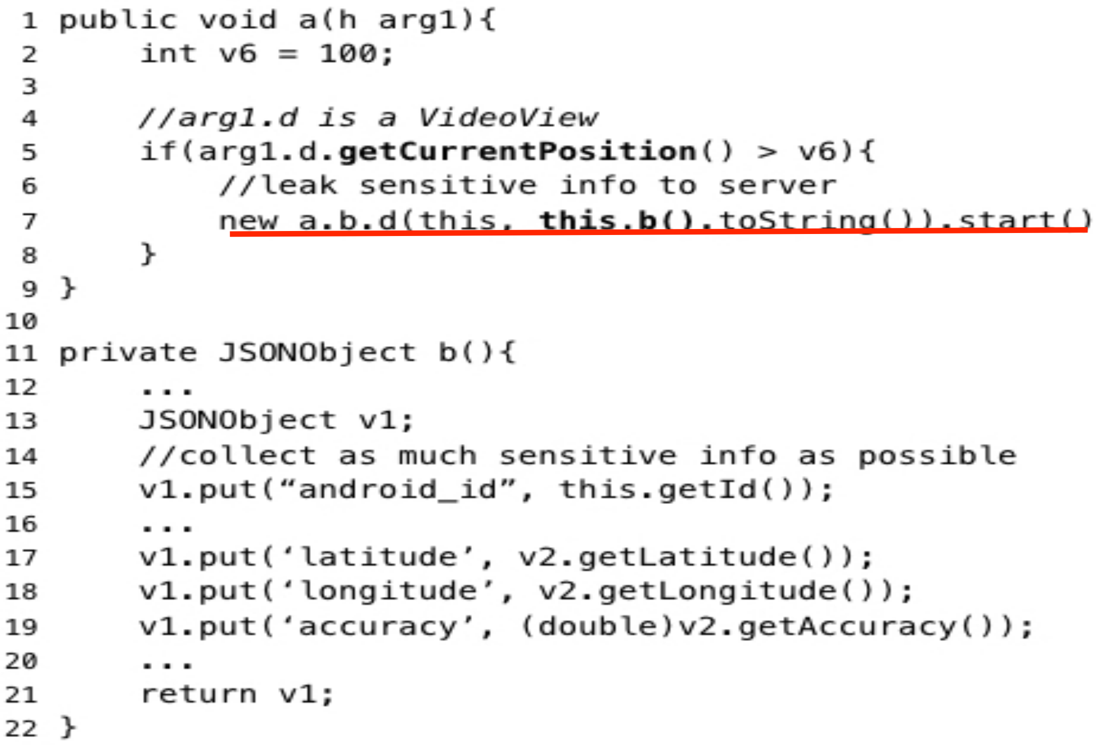
- **Triggers**


time
1215
location
345
device information
1693
device setting
114
user interface
51
system event
290
system services
20
0
500
1000
1500
2000

- Evolution

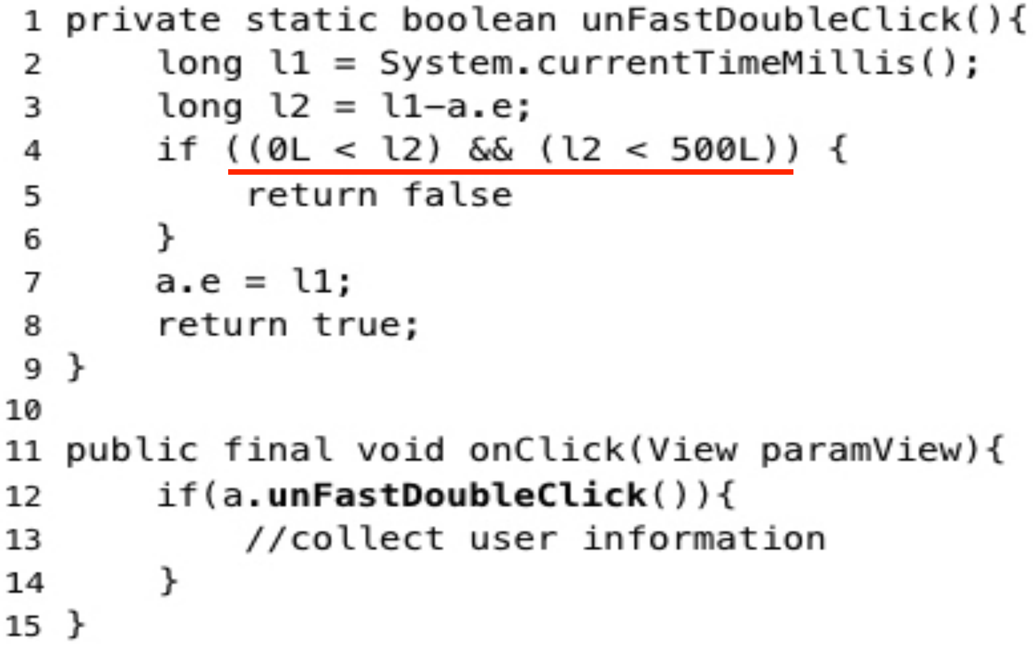

0.25
0.2
0.15
0.1
0.05
0
2008
2009
2010
2011
2012
2013
2014
2015
2016
Year

Video trigger



```
public void a(h arg1){
    int v6 = 100;

    //arg1.d is a VideoView
    if(arg1.d.getCurrentPosition() > v6){
        //leak sensitive info to server
        new a.b.d(this, this.b().toString()).start()
    }
}

private JSONObject b(){
    ...
    JSONObject v1;
    //collect as much sensitive info as possible
    v1.put(“android_id”, this.getId());
    ...
    v1.put(‘latitude’, v2.getLatitude());
    v1.put(‘longitude’, v2.getLongitude());
    v1.put(‘accuracy’, (double)v2.getAccuracy());
    ...
    return v1;
}
```

▪ Click interval



```
private static boolean unFastDoubleClick(){
     long l1 = System.currentTimeMillis();
     long l2 = l1-a.e;
     if ((0L < l2) && (l2 < 500L)) {
          return false
     }
     a.e = l1;
     return true;
}

public final void onClick(View paramView){
     if(a.unFastDoubleClick()){
          //collect user information
     }
}
```

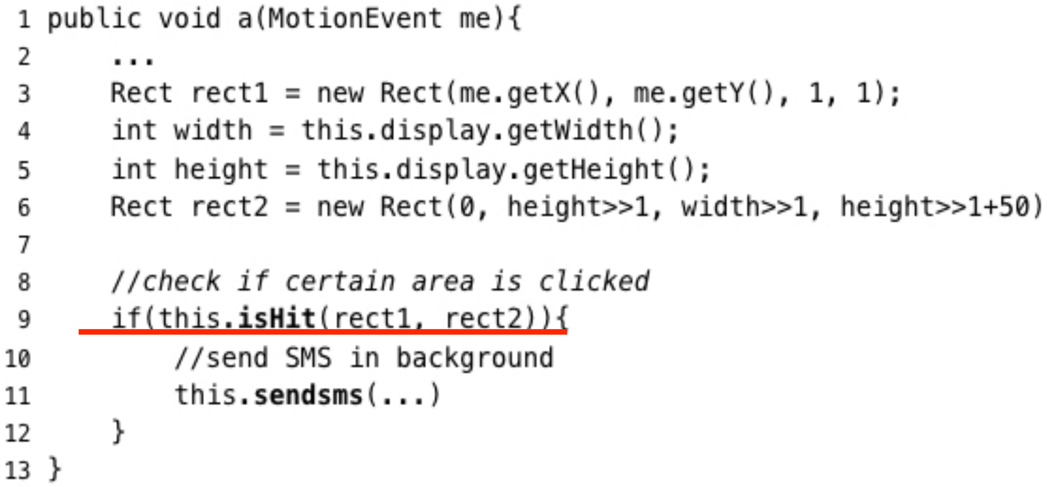
Takeaways

- Promising to combine machine learning and lightweight program analysis
  - Towards scalability
- First step towards generic evasion detection techniques
  - Verify the feasibility
- By >330k apps, prevalence of HSO in the wild
  - Urgency of countermeasures

Thank you!

Questions ?

Trapdoor on view



```java
public void a(MotionEvent me){
    ...
    Rect rect1 = new Rect(me.getX(), me.getY(), 1, 1);
    int width = this.display.getWidth();
    int height = this.display.getHeight();
    Rect rect2 = new Rect(0, height>>1, width>>1, height>>1+50)

    //check if certain area is clicked
    if(this.isHit(rect1, rect2)){
        //send SMS in background
        this.sendsms(...)
    }
}
```

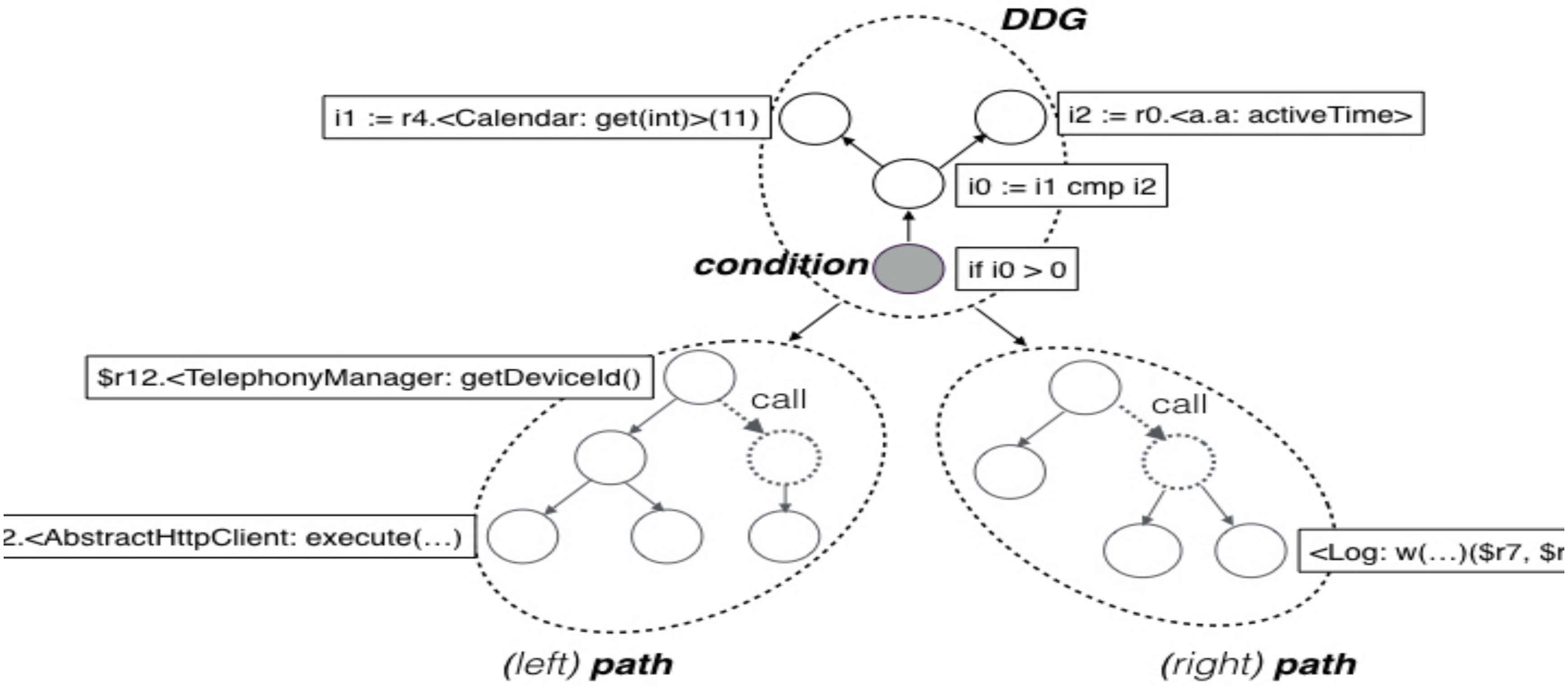
Limitations

- Further Evasion
- Intrinsic limitation of static analysis
- Coverage
  - Native code
  - Server side

Future work

- UI Context
- User perception, app description context

- # Condition Path Graph(CPG)


DDG
i1 := r4.<Calendar: get(int)>(11)
i2 := r0.<a.a: activeTime>
i0 := i1 cmp i2
condition
if i0 > 0
$r12.<TelephonyManager: getDeviceId()
call
2.<AbstractHttpClient: execute(...)
call
<Log: w(...)($r7, $r
(left) path
(right) path