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
The problem?

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

```java
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 absence 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

```java
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

```java
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
  
  ```java
  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

```java
com.android.dvci.core:
am.isEmu()

{ Build.FINGERPRINT
  Build.TAGS
  Build.PRODUCT
  Build.DEVICE
  Build.BRAND
  Build.MANUFACTURE
  getDeviceId()
  getLine1Number()
  getSubscriberId()
  ...
}
• **Architecture**

![Diagram of architecture](image-url)
- **Features**

- **Data and semantic dependency between Condition and Paths**
  - Data Dependency (DF1 DF2) : k/n
  - Semantic relevance: Implicit Relation (IR1 IR2)
    - Based on semantic relevance
    - And Frequency Analysis

<table>
<thead>
<tr>
<th>Item in condition</th>
<th>Item in path</th>
</tr>
</thead>
<tbody>
<tr>
<td>(android.location.LocationManager: isProviderEnabled(· · ·))</td>
<td>(android.location.LocationManager: requestLocationUpdates(· · ·))</td>
</tr>
<tr>
<td>(android.webkit.WebViewClient: {init}())</td>
<td>(android.webkit.WebView: loadUrl(· · ·))</td>
</tr>
<tr>
<td>(android.net.NetworkInfo: getState())</td>
<td>(android.net.ConnectivityManager: getNetworkInfo(· · ·))</td>
</tr>
<tr>
<td>(android.os.Environment: getExternalStorageState())</td>
<td>(java.io.File: mkdir())</td>
</tr>
<tr>
<td>‘location_providers_allowed’</td>
<td>(android.location.LocationManager: getLastKnownLocation(· · ·))</td>
</tr>
<tr>
<td>‘PACKAGE_CHANGED’</td>
<td>(android.content.pm.PackageManager: java.util.List getInstalledPackages(· · ·))</td>
</tr>
<tr>
<td>‘GET_ACCOUNTS’</td>
<td>(android.accounts.AccountManager: getAccountsByType(· · ·))</td>
</tr>
</tbody>
</table>
Features

- Behavior Differences
  - Data distance (DD)

We also want to know data relations between two paths

\[
DD = 1 - \frac{1}{2} \left( \frac{V_i \cap V_r}{V_i \cup V_r} + \frac{F_i \cap F_r}{F_i \cup F_r} \right)
\]
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

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSO</td>
<td>0.98</td>
<td>0.944</td>
<td>0.962</td>
</tr>
<tr>
<td>Non-HSO</td>
<td>0.946</td>
<td>0.981</td>
<td>0.963</td>
</tr>
<tr>
<td>Weighted Avg.</td>
<td>0.963</td>
<td>0.962</td>
<td>0.962</td>
</tr>
</tbody>
</table>

- **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
- Triggers

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>1215</td>
</tr>
<tr>
<td>location</td>
<td>345</td>
</tr>
<tr>
<td>device information</td>
<td>1693</td>
</tr>
<tr>
<td>device setting</td>
<td>114</td>
</tr>
<tr>
<td>user interface</td>
<td>51</td>
</tr>
<tr>
<td>system event</td>
<td>290</td>
</tr>
<tr>
<td>system services</td>
<td>20</td>
</tr>
</tbody>
</table>
Evolution
Video trigger

```java
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

```java
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)

\[ i_1 := r_4.<\text{Calendar: get(int)}> (11) \]

\[ i_2 := r_0.<a.a: \text{activeTime}> \]

\[ i_0 := i_1 \text{ cmp } i_2 \]

\[ \text{if } i_0 > 0 \]

\[ \text{condition} \]

\[ \begin{align*}
\text{Sr12.<TelephonyManager: getDeviceId()} \\
2.<\text{AbstractHttpClient: execute(...)}
\end{align*} \]

(left) \text{path} 

(right) \text{path} 

\[ \text{call} \]

\[ \text{call} \]

\[ \text{<Log: w(...)(r7, r)} \]