Dynamic Detection of Inter-Application Communication Vulnerabilities in Android

Daniel Barton

Authors/Paper Metadata

- *Roee Hay*
  - *IBM Security*
- *Omer Tripp*
  - *IBM T.J. Watson Research Center*
- *Marco Pistoia*
  - *IBM T.J. Watson Research Center*
- *International Symposium on Software Testing and Analysis (ISSTA) 2015*
  - *27.7% Acceptance Rate*

Paper Overview

- *Testing for IAC integrity vulnerabilities.*
  - *Low overhead, high coverage.*
- *IntentDroid*
  - *Monitor select set of APIs and use it to guide testing.*
  - *Security related, IAC data.*
  - *Prune redundant tests, recover custom IAC fields, vary inputs to increase path coverage.*

IAC Attack Model

- *Attacks that exploit public components.*
  - *Exported, requires neither signed nor system permissions, receives unsanitized data from another public component.*
- *IAC Attack Vectors*
  - *Cross-Application Scripting*
  - *Fragment Injection*
  - *Client-side SQL Injection*
  - *File Manipulation*
  - *Native Memory Corruptions*
  - *Unhanded Exceptions (DOS)*

IAC Data Retrieval Example

```
1  class WriteActivity extends Activity {
2    static { System.loadLibrary("native"); }
3
4    protected void onCreate(Bundle savedInstanceState) {
5     super.onCreate(savedInstanceState);
6     setContentView(R.layout. activity_write );
7     Intent intent = this.getIntent();
8     String data = intent.getDataString();
9     File target = new File(this.getFilesDir(), "log");
10    int fd = NativeIO.open(target.getAbsolutePath());
11    int size = NativeIO.write(fd, data);
12   Bundle bundle = intent.getBundle();
13   String fooExtra = intent.getStringExtra("foo");
14   String barExtra = bundle.getString("bar");
15   if (fooExtra == null) return;
16   SQLiteDatabase db = ...;
17   db.execSQL(String.format("UPDATE ...
18       WHERE name='%s' ... id= '%s'", fooExtra, barExtra));
19   boolean b1 = intent.getBooleanExtra("b1", false);
20   if (true == b1) {
21    boolean b2 = intent.getBooleanExtra("b2", false);
22    boolean b3 = intent.getBooleanExtra("b3", false);
23    if (true == b2)
24     db.execSQL(String.format("UPDATE ...
25         WHERE name='%s'", fooExtra));
26    if (true == b3)
27     db.execSQL(String.format("UPDATE ...
28         WHERE name='%s'", fooExtra)); } } }
```

Challenges/Solutions in IntentDroid

- *Naive Fuzzing - Injecting all available test payloads into intents' data field.*
  - *Limited coverage at a high cost.*
  - *Solution: Prune irrelevant test via probing.*
- *Optimized performance (accuracy) via probing.*
  - *Payload is in a custom (extra) parameter.*
  - *Solution: Monitors getExtra(...) calls.*
- *Potentially unexecuted execution paths.*
  - *Solution: Manipulate boolean parameters.*

Pruning Tests via. Probing

- *Goal: Decide which test should be applied to an input.*
- *Solution: For different security rules, track which relevant APIs are invoked while processing the input and which data arguments reach the input.*
  - *Security rules define necessary conditions for a vulnerability to manifest.*

Retrieving Custom Parameters

- *Goal: Recover extra fields in intents.*
- *Solution: Instrument platform APIs used to read custom fields (getStringExtra(...), etc.).*
  - *Monitors Intent.getBundle() for additional extras influenced by IntentDroid message by placing monitoring code within the Bundle copy constructor.*

Thorough Path Exploration

- *Enumerate all possible combinations of boolean extras (naive).*
- *"..IntentDroid enforces a certain simplifying assumptions.." (Hay et al. 5)*
  - *Extras dominate if one dictates access to the other.*
  - *Independent if neither dominate one another.*
- *Toggles all independent and dominant extras.*

IntentDroid Algorithm

- *Deploys target app in debug mode.*
- *Obtain manifest file.*
- *Parse manifest file for public (vulnerable) activities.*
- *Create benign IAC inputs for the vulnerable activities.*
- *Begin testing loop.*

Testing Loop

- *For each input activity:*
  - *Identify which attack types apply.*
  - *Create payloads for each applicable attack type.*
  - *Apply payload to input. Yields:*
    - *Additional input points.*
    - *Records app behaviors/outputs.*
  - *Record vulnerability is output confirms a successful attack.*

Testing Loop Modes

- *Monitoring*
  - *Tracks which security relevant APIs are invoked and which custom fields are accessed.*
- *Testing*
  - *When new inputs are detected, probes are sent to detect potential attacks.*
- *Exploration*
  - *Toggles boolean extras for a probe.*

Formalized IntentDroid Algorithm

Input: (D, M, V, I) `// testing capabilities`
Input: A `// subject mobile app`
Output: O `// detected vulnerabilities`
begin
  $A' \longleftarrow I(A)$ `// instrument` A
  $E \longleftarrow$ declared interface points of A
  $O \longleftarrow \emptyset$
  while $E \neq \emptyset$ **do**
    $e \longleftarrow$ choose from E
    $E \longleftarrow E \setminus \{e\}$
    foreach $m \in M(e)$ **do**
      $b \longleftarrow D(m, e)$
      if b **then**
        $p[m, e] \longleftarrow$ create payload for e with m
        $(r, E') \longleftarrow$ fire $p[m, e]$
        $E \longleftarrow E \cup E'$
        if $V(r)$ **then**
          $O \longleftarrow O \cup \{r\}$
        end
      end
    end
  end
end

Algorithm 1: **Outline of the Core IntentDroid Algorithm, where D, M, V and I Denote Detection, Mutation, Validation and Instrumentation, Respectively**

Experimental Evaluation Setup

- *80 Android app suite:*
  - *4 Enterprise apps, 3 native apps for Android 4.4, 73 top-popular Google Play apps.*
- *LG Nexus 5 Phone with Android 4.4.*
- *Professional ethical hacker audited the apps using a brute-force fuzzing tool.*
  - *Revealed 163 IAC vulnerabilities.*

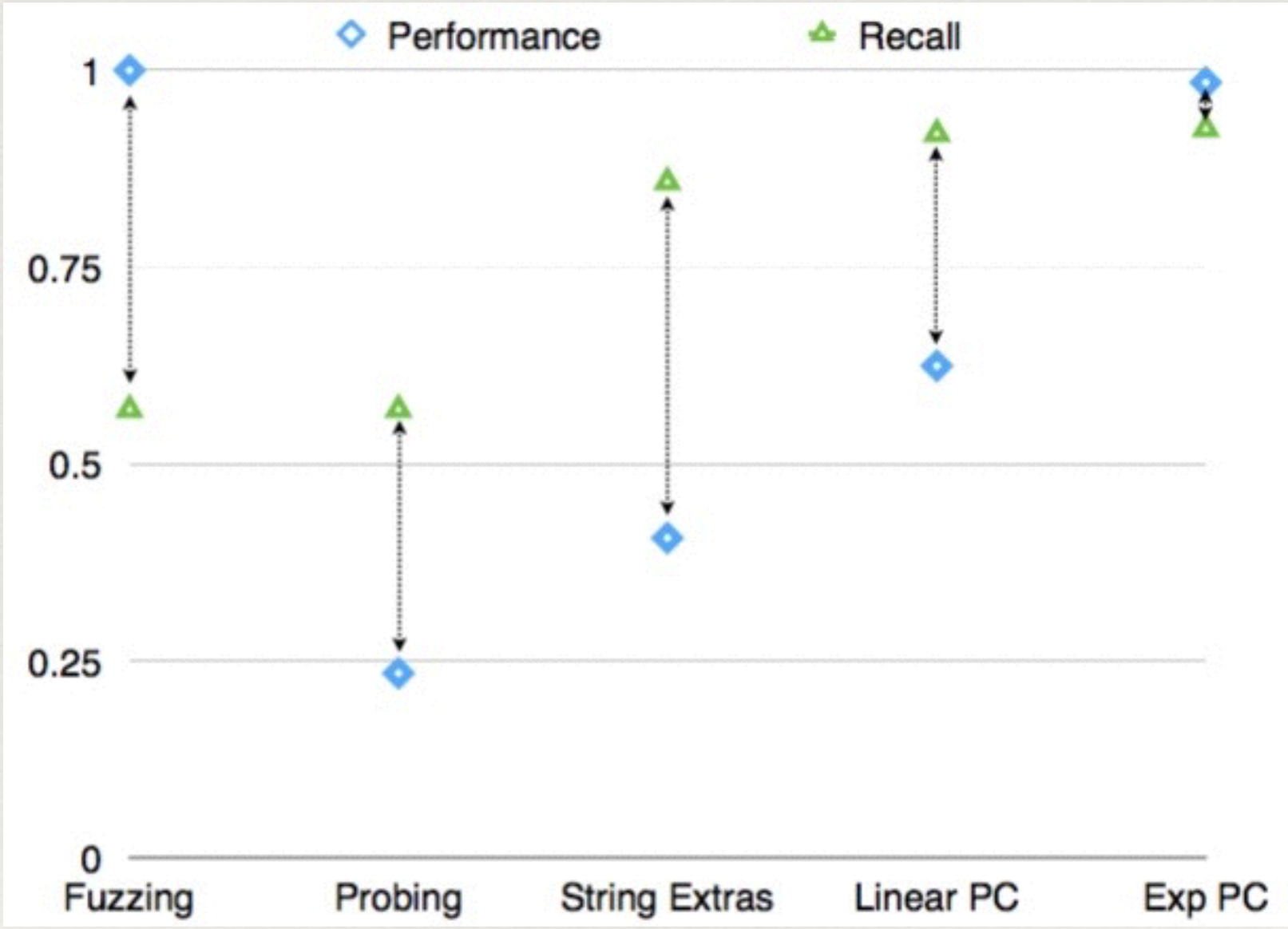
Hypotheses to be Tested

- *H1: Probing boosts performance.*
  - *Averages 64 tests and 24 mins without probing, < 15 tests and < 7 mins with.*
- *H2: String extras are often vulnerable.*
  - *94/163 (0.57) without strings as attack targets, 140/163 (0.85) with.*
  - *Increases time (12. as opposed to 7 min.) and tests (26 tests as opposed to 15).*

Hypotheses to be Tested

- *H3: Boolean extras manifest in path conditions.*
  - *151/163 (0.92) recall.*
  - *Increases time (12 min. to 25 min.) and tests (26 to 63).*
- *H4: Linear-time path exploration is effective.*
  - *Tests wether domination/independence allows for individual toggling.*
  - *Time decreases (19 min. as opposed to 25) as well as tests (40 as opposed to 63).*

Evaluation Results



Preference Activities and Fragment Loading

- *"Any app containing and exported Activity that extends the PreferenceActivity can be subverted to load an arbitrary class (available to the class loader of the target application) by exploiting the unsafe dynamic Fragment loading process." (Hay et al. 9)*
- *Able to exploit Gmail, Google Translate, and Dropbox.*

XAS Weakness in Apache Cordova

- *"...a malicious caller could launch the Activity with an Intent whose respective Bundle maps 'url' to an unintended value. The provided URL will consequently be loaded by Cordova and rendered within the WebView." (Hay et al. 9)*

- *Enables theft of private data, such as login credentials, in apps running on Cordova.*

File Manipulation in the Firefox Browser

- *"...an adversarial agent can manipulate the source path of the moved file as well as the deduced extra file." (Hay et al. 10)*
- *Allows the attacker to have control over the server that the crash dump is reported to, as well as theft of sensitive information.*

Conclusions

- *IntentDroid: Comprehensive testing algorithm for inbound IAC integrity threats.*
  - *Commercial cloud service.*
- *Most detected threats in the evaluation were low severity.*
  - *Only impact app stability or assume complex payload hard to create in practice.*
- *Found 3 severe vulnerabilities.*