INTELLIDROID
A Targeted Input Generator for the Dynamic Analysis of Android Malware

Michelle Y. Wong and David Lie
University of Toronto
Department of Electrical and Computer Engineering
NDSS Symposium 2016
Static vs. Dynamic Analysis

• Static analysis: analyze source code or byte code
  ◦ Imprecise
  ◦ No run-time data

• Dynamic analysis: analyze during execution
  ◦ Run-time values $\rightarrow$ precise
Dynamic Code Coverage

• To detect malicious activity, first have to execute it

• Example:

```python
message = <receive confirmation SMS>
if message.number == '1234':
    <malicious action>
```
Concolic Testing

- Run all execution paths in application
- Symbolic execution, solve constraints for inputs

![Diagram showing constraint relationships]

- constraint 1
- constraint 2
- !(constraint 2)
- !(constraint 1)
- constraint 3
- !(constraint 3)
Specific Malicious Paths

- Malicious activity only executed in certain parts of the code
IntelliDroid

• Targets specific parts of the application
  ◦ Input generator for existing dynamic detector
  ◦ Hybrid static and dynamic design

• Implemented for Android

• Improve malware analysis and detection
Target Malicious Paths

• Malicious activity present only in certain parts of the code
Target Malicious Paths

- Use static analysis to look for call paths to malicious activity
Target Over-Approximation

- Target over-approximation of malicious behaviors
Target Over-Approximation

• Target over-approximation of malicious behaviors
Targeted Methods

- Use method invocations as over-approximation
  - Depends on attached dynamic malware detector

- Existing dynamic detectors for Android:
  - ✓ Method invocations
  - ✓ System call traces
  - ✗ Anomaly detection

<table>
<thead>
<tr>
<th>Dynamic Tool</th>
<th>Goal</th>
<th>Features for Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>AASandbox [10]</td>
<td>Monitor behavior via tracking of system calls</td>
<td>System calls</td>
</tr>
<tr>
<td>Andromaly [36]</td>
<td>Malware detection via system resource usage</td>
<td>Low-level device features (e.g., battery usage, CPU load)</td>
</tr>
<tr>
<td>CopperDroid [39]</td>
<td>Monitor behavior via system call tracking</td>
<td>System calls</td>
</tr>
<tr>
<td>Crowdroid [12]</td>
<td>Monitor behavior via tracking of system calls</td>
<td>System calls</td>
</tr>
<tr>
<td>DroidBox [18]</td>
<td>Sandbox to monitor external accesses</td>
<td>Sink API methods</td>
</tr>
<tr>
<td>DroidRanger [50]</td>
<td>Detect malware using pre-specified behavioral footprints and heuristics</td>
<td>Sequence of API method invocations and parameters</td>
</tr>
<tr>
<td>DroidScope [39]</td>
<td>Plugins for API tracking, instruction tracing, and taint tracking</td>
<td>API methods; source/sink API methods</td>
</tr>
<tr>
<td>RiskRanker [39]</td>
<td>Detect malware using known vulnerability signatures</td>
<td>Sequence of API method invocations</td>
</tr>
<tr>
<td>TaintDroid [19]</td>
<td>Detect privacy leakage</td>
<td>Source/sink API methods</td>
</tr>
<tr>
<td>VetDroid [47]</td>
<td>Malware detection via permission use behavior</td>
<td>Permission requests (can be mapped to API methods)</td>
</tr>
</tbody>
</table>
Static Constraint Extraction

- Extract constraints on inputs that can trigger targeted paths
Targeted Input Injection

- Inject constrained inputs to execute paths at run-time

Static

- Path Constraints
- ...
Challenges

• Finding targeted paths using static analysis
  ◦ Imprecision?

• Executing path to suspicious code
  ◦ Dependencies between paths?

• Run-time input injection
  ◦ Where to inject?
Static Imprecision

- Static analysis cannot determine run-time values

- Example:

```java
message = <receive confirmation SMS>
if message.number == <file A>.text:
    <malicious action>
```

**Constraint**

```java
<SMS message>.number == <file A>
```
Using Run-time Data

- Solve constraints at run-time (with run-time data)

Static

Static Constraints

Path 1 Constraints

Path N Constraints

Dynamic

Run-time

constraint solver

file A “1234”

location San Diego

<SMS message>.number == “1234”
Path Dependencies

• Data- and control-flow dependencies between call paths
Path Dependencies

• Data- and control-flow dependencies between call paths
Run-Time Injection

Diagram:
- Application
  - Event Handler
  - SMS Handler
  - Event Handler
- Framework
  - System Service
  - SMS Service
  - System Service
- Hardware/Device
  - Cellular Radio
  - Sensor
Application Injection

Application
- Event Handler
- SMS Handler
- Event Handler

Framework
- System Service
- SMS Service
- System Service

Hardware/Device
- Cellular Radio
- Sensor

info on SMS?
what SMS?
Device-Framework Injection

```
Application
  Event Handler
  SMS Handler
  Event Handler

Framework
  System Service
  SMS Service
  System Service

Hardware/Device
  Cellular Radio
  Sensor
```

info on SMS?

OK!
Contributions

• Static imprecision
  ◦ Dynamic constraint solving with run-time values

• Path dependencies
  ◦ Event chains

• Consistent input injection
  ◦ Device-framework injection
Static Component

IntelliDroid
Static Component

Targeted Behaviors

App APK

Application

BootReceiver
onReceive(Intent i):
if i == BOOT_COMPLETED:
    a = 1234

SMSReceiver
onReceive(Intent i):
if i == SMS_RECEIVED:
    handleSMS(…)

handleSMS(addr, msg):
if a == addr:
    sendTextMessage(…)

Targeted Behaviors
App APK
Static Component

Targeted Behaviors
- Extract event handlers
- Find call paths
- Extract path constraints
- If dependency: find dependent path

App APK

Application

BootReceiver
onReceive(Intent i):
if i == BOOT_COMPLETED:
  a = 1234

SMSReceiver
onReceive(Intent i):
if i == SMS_RECEIVED:
  handleSMS(...)
handleSMS(addr, msg):
if a == addr:
  sendTextMessage(...)

Add to event chain

IMPLEMENTATION
Static Component

- Extract event handlers
- Find call paths
- Extract path constraints
- Add to event chain

If dependency: find dependent path

Application

**BootReceiver**

```java
onReceive(Intent i):
if i == BOOT_COMPLETED:
a = 1234
```

**SMSReceiver**

```java
onReceive(Intent i):
if i == SMS_RECEIVED:
handleSMS(…)
```

handleSMS(addr, msg):
```java
…”
```
Implementation

• Static analysis (Android-specific): WALA

• Dynamic component:
  ◦ Client program (Python)
    - Constraint solver: Z3
  ◦ Custom Android OS
    - IntelliDroidService: system service to receive input information and inject events

Evaluation

• Can IntelliDroid be integrated with existing dynamic malware detectors?

• Can it execute targeted behaviours at run-time?

• Is the analysis time reasonable?
Integration with TaintDroid

- Attached to TaintDroid (dynamic taint tracking tool)
- Input generator to execute taint sources and sinks

**Diagram:**
- **IntelliDroid (Static)**
- **IntelliDroid (Dynamic)**
- **TaintDroid Dynamic Detector**
- **leakage paths**
- **taint source**
  - e.g. getDeviceId()
- **taint sink**
  - sendTextMessage()
IntelliDroid-Driven TaintDroid

• Tested on 26 privacy leaks in 17 malicious apps \(^1,2\)

• IntelliDroid: Triggered and detected all leaks
  ◦ Monkey: Missed 21 leaks

• Executed < 5% of application code

---


Targeted Input Injection

- Target malicious behaviours in Android Malware Genome and Contagio
- Triggered 70 out of 75 behaviours
- Missed behaviors:
  - Encoding
  - File dependencies (currently not supported)
Performance

- Scales for large-scale analysis of applications
- Static analysis:
  - 138.4s per application
- Dynamic constraint solving:
  - 4.22ms per targeted call path

---

Conclusion

- Targeted input generation for effective dynamic malware detection

- IntelliDroid
  - Static constraint extraction with run-time data
  - Event chains and framework injection

- Integrated with existing dynamic tools (TaintDroid)

- Improve effectiveness, reduce amount of code to be executed (< 5%)