RV-Android: Efficient Parametric Android Runtime Verification, a Brief Tutorial

Philip Daian, Yliès Falcone, Patrick Meredith, Traian Florin Serbanuta, Shin'ichi Shiriashi, Akihito Iwai, Grigore Rosu

RV-Android: Efficient Parametric Android Runtime Verification, a Brief Tutorial

Philip Daian[1], Yliès Falcone[4], Patrick Meredith[1], Traian Florin Şerbănuţă[1], Shin'ichi Shiriashi[3], Akihito Iwai[2], and Grigore Rosu[14]

[1] Runtime Verification Inc. support@runtimeverification.com
[2] Denso International America Inc. southfieldpr@denso-diam.com
[3] Toyota InfoTechnology Center U.S.A. info@us.toyota-itc.com
[4] University of Illinois at Urbana-Champaign {ylies, grosu}@illinois.edu

Abstract. RV-Android is a new freely available open source runtime library for monitoring formal safety properties on Android. RV-Android uses the commercial RV-Monitor technology as its core monitoring library generation technology, allowing for the verification of safety properties during execution and operating entirely in userspace with no kernel or operating system modifications required. RV-Android improves on previous Android monitoring work by replacing the JavaMOP framework with RV-Monitor, a more advanced monitoring library generation tool with core algorithmic improvements that greatly improve resource consumption, efficiency, and battery life considerations. We demonstrate the developer usage of RV-Android with the standard Android build process, using instrumentation mechanisms effective on both Android binaries and source code. Our method allows for both property development and advanced application testing through runtime verification. We showcase the user frontend of RV-Monitor, which is available for public demo use and requires no knowledge of RV concepts. We explore the extra expressiveness the MOP paradigm provides over simply writing properties as aspects through two sample security properties, and show an example of a real security violation mitigated by RV-Android on-device. Lastly, we propose RV as an extension to the next-generation Android permissions system debuting in Android M.

1 Introduction

With the rise in popularity of Android [1], a Linux-based consumer smartphone operating system, the need for effective techniques to improve the security and reliability of third-party applications running on end user devices is well established [2]. One solution explored by previous work in the field is the use of runtime verification and runtime enforcement to detect and recover from violations of formal safety properties during the execution of Android applications [2] [3]. Some previous work in this space has relied on using kernel modifications to the Linux base of Android to generate the runtime traces required to verify safety properties [3] [4]. This solution is inflexible for several reasons: it requires root access to the device to install, requires reinstallation on each operating system

upgrade, and provides few additional guarantees from a compromised kernel over a userspace monitoring solution.

Other RV-based Android work has used JavaMOP, an experimental monitor oriented programming framework, to generate Android monitoring libraries which are weaved into third-party userspace applications, providing monitoring functionality and guarantees without requiring kernel-level modifications [2].

This approach has been demonstrated using AspectJ [5] for weaving, and allows us to weave binary bytecode useful towards instrumenting off-the-shelf third party packaged applications (apk's) [2] [4]. Our work focuses on improving these approaches with efficient and versatile runtime verification tools that benefit from the previous research endeavors related to JavaMOP, resulting in a unified and open framework for runtime verification and analysis of Android applications.

1.1 Contributions

RV-Monitor is a proprietary library generation technology, allowing for the runtime monitoring, verification, and enforcement of safety properties through the generation of generic monitoring libraries. RV-Monitor is provided free for non-commerical use, and represents the evolution of the prototype JavaMOP tools with improvements in the codebase and core algorithms. Like JavaMOP, RV-Monitor supports logic plugins, allowing for the specifications of properties in multiple formalisms including regular expressions, context-free grammars, automata, and past-time linear temporal logic.

We make the case for the future use of RV-Monitor and its related Android runtime library for use in runtime verification and runtime enforcement of Android applications. We compare monitor-oriented programming techniques to popular aspect-oriented techniques, acknowledging the usefulness of both and supporting both as property inputs to our tools. We analyze a real security violation on the Android platform with the potential to be stopped by monitor-oriented programming, including the relevant property with our tool's distribution. Lastly, we discuss the future of RV on the Android platform and lay out a roadmap for future industry-lead work in the space.

2 RV-Android Overview and Build Process

RV-Android consists of two components, a monitoring library generation tool and a runtime environment used in the generation of these libraries for dynamic on-device property monitoring and violation recovery. For the first of these RV-Monitor is used off-the-shelf, allowing for the specification of both formal properties over events and the instrumentation points for these events in a single monitor-oriented file. This monitor-oriented programming is achieved through a new version of JavaMOP, a lightweight compiler that generates RV-Monitor (monitoring library) and AspectJ (program instrumentation) output. Unlike versions of JavaMOP used in previous work, which were fully responsible for

monitoring applications and did not leverage RV-Monitor, this next-generation JavaMOP stands to benefit from the significant core algorithmic improvements that form the basis of the RV-Monitor IP [6] while completely separating the generation of efficient monitoring libraries from application instrumentation.

In our work, we will separate the discussion on instrumentation from the discussion on event and property definitions. We do this to leave open the possibility of future instrumentation models. While we focus on instrumentation methods using AspectJ in this work, other instrumentation mechanisms for packaged Android binaries have already been effectively demonstrated and proved [2,4,7,8]. RV-Monitor is compatible with any Java instrumentation method, and the JavaMOP project can be extended to generate the required input for other instrumentation tools if necessary.

Our website, `http://runtimeverification.com/android`, provides downloads and full instructions for the use of RV-Android, as well as the examples we discuss in the remainder of the paper and a video demonstrating a few currently available capabilities of the tool.

2.1 Build Process

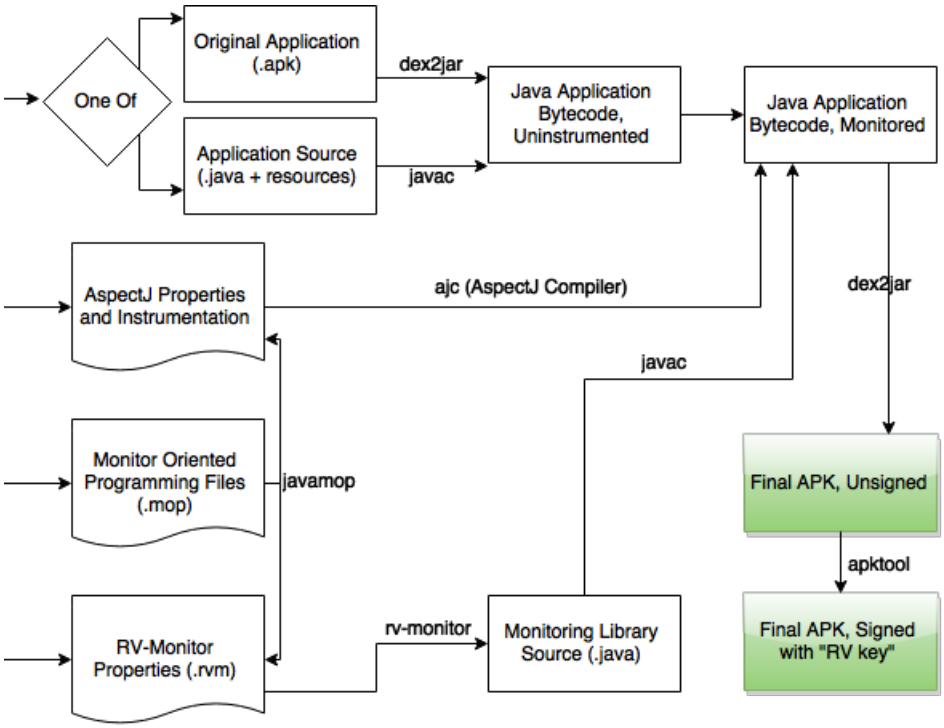

Fig. 1: RV-Android Build Process

The RV-Android process (shown above in Figure 1) forms the basis of all tools based on RV-Android. Taking either a binary packaged Android application or set of source files as input, RV-Android additionally optionally takes any (or all) of aspect files, RV-Monitor properties, and monitor-oriented programming (MOP) files compatible with the JavaMOP tool. This flexibility in input allows RV-Monitor to be used with a wide variety of property formats, including aspect-oriented AspectJ properties that do not require formalisms or the additional features provided by RV-Monitor or JavaMOP.

By being able to mix and bundle these diverse formats together into a single set of properties as input to a single tool, property developers and application developers have an easy way to develop, apply, and share dynamic properties of their choice without the need to constrain themselves to a single tool. Futhermore, property developers have the ultimate control over the instrumentation of their properties in the original application: they can choose to use AspectJ directly, use MOP, or use other techniques of manual instrumentation. In doing so, the goal is to create a platform in which all developers of runtime properties for the Android platform feel comfortable using their preferred technologies, encouraging third-party property development.

RV-Monitor also features a flexible plugin architecture that allows for the development of custom or third-party formalisms supported by the tool, allowing virtually unlimited expressiveness in the properties it defines. By combining this with instrumentation in a single monitor-oriented programming files, advanced developers can write complex properties succinctly while defining simple properties not requiring dedicated formalisms in native Java directly through AspectJ.

RV-Android is also an open source project built around the closed source RV-Monitor project, but can be used without RV-Monitor as an aspect oriented programming platform, as well as extended to work with any number of tools supporting Java transformation and analysis. Because it is open source, RV-Android can also be extended by other tool developers who provide runtime analysis and verification of applications to integrate any technique into its workflow. We are already developing prototype extensions for taint analysis and other popular dynamic analysis techniques, extending the capabilities of RV-Android beyond the capabilities of any single tool.

Figure 1 shows the output of RV-Android as a signed, packaged Android application to be distributed to users. By signing all verified and monitored applications with an "RV" key from a source trusted by the user, the user's smartphone can then verify that the application has been correctly monitored and transformed to the specifications provided as input to the tool.

Excluded from the figure is the inclusion of the AspectJ and RV-Monitor runtimes in the monitored and instrumented application. These runtimes are needed for the instrumentation and monitoring of the application on-device, and are simply the binary runtimes as provided by each project.

2.2 RV-Android for Developers

In order to provide a full framework for runtime verification and enforcement on the Android platform, we must allow developers to complete their two primary tasks of interest with regards to runtime verification: developing new properties to check on their applications and third-party applications, and checking their own applications against existing property sets. To do this, we provide two versions of RV-Android for developers, targeted at two possible use cases. The first allows developers to integrate RV-Android into projects where they are using the standard Android build process and have access to the source code of the application, instrumenting source code directly. The second version is able to monitor properties of binary Android applications, or apks, and can be used on any application that runs on the Android platform.

Monitoring Source Code To monitor source code, we provide a version of RV-Android called the "developer source" edition. Found in the "developer_src" subdirectory of the RV-Android distribution, this edition requires the developer to create two directories, "aspects" and "properties". By placing AspectJ files and RV-Monitor or MOP properties in the relevant directories and following the remainder of the setup instructions for the source edition, developers can integrate monitors and runtime verification in their build and testing process with no modification required to the source of the application itself.

This provides a convenient way for developers interested in obtaining the maximum assurance from their Android applications to leverage runtime verification. Additionally, developers using the default Android build process who wish to ship monitors integrated with their final application to end users (for assurance, security, or enforcement purposes) can use this version to monitor their source directly, requiring no binary transformations.

Monitoring Binary APKs The second and more flexible distribution of RV-Android for developers focuses on monitoring arbitrary binary Android applications, and is referred to as the "command line" distribution. This distribution can also be used by advanced users who are comfortable using command-line tools. Similarly to the previous application, the tool takes both properties and aspects, and has a simple command-line interface with parameters as follows:

```
./rv_android [apk] [keystore] [keystore password]
[signing key alias] [monitors_directory] [aspects_directory]
```

Because of its simplicity and extensibility, this version of RV-Android can be integrated into any environment or build process. We recommend this version to all new developers interested in RV on Android.

2.3 RV-Android for End Users - GUI Frontend

For end-users who are not familiar with command-line applications and property development, but still wish to gain some additional assurance in or control over

Fig. 2: RV-Android On-Device Architecture

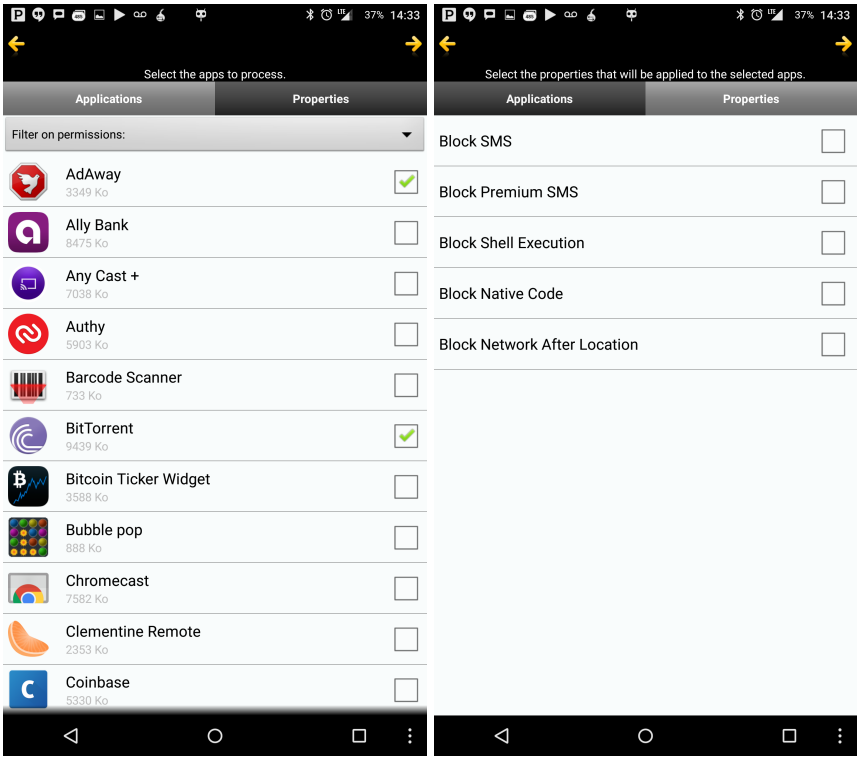

Fig. 3: RV-Android Screenshots, Application and Property Selection

their applications, we provided a GUI version of RV-Android. The architecture of this system is shown in Figure 2. RV-Android runs entirely on a remote server in the cloud, which runs the same CLI version of RV-Android with the workflow in Figure 1. The program on the user's device is simply a shell allowing them to manage and select properties, and select an application on their device to instrument. This is similar to previous approaches applied to runtime verification on Android [9].

Through this graphical user interface, we allow users that may be completely unfamiliar with RV or its concepts to apply generic "properties" to any of the applications currently on their device. We also allow them to download third-party sets of properties and read their descriptions.

Such an application aims to bring RV to the mainstream by providing simple, human-readable descriptions of what each property does or checks for and allowing any property to be used with any userspace application on any Android smartphone, without requiring root access. In the user frontend, there is no distinction made between properties written as aspects, monitors, or other formats, all of which are simply referred to as "properties".

We believe such a platform is an ideal introduction point to runtime verification for users and developers alike. With the increased importance of security on mobile devices, which often carry personal information and other sensitive data [10] [11], and the ability to instrument any arbitrary binary Android application by virtue of the Java bytecode format used, we are able to show off the powerful and generic nature of runtime verification technology in an environment becoming increasingly important to end users.

Figure 3 shows two screenshots of the RV-Android frontend for end-users, currently in beta. Users select an application followed by a set of properties to apply to the application, with the remainder of the process being automated. These properties have extended descriptions that can be viewed by the user, and can be extended to include custom or user-defined properties. Applications can also be filtered by those requesting a set of permissions considered by the user to be particularly sensitive.

3 Towards Practicality - RV-Android Case Studies

Having presented the overall architecture and currently available distributions of the RV-Android tool, the question of whether such techniques are practical and cost effective for use on real applications and devices arises naturally. The practicality of the tools presented hinges on the ability to develop a useful set of properties supported by the tool, which can then be shared, packaged, and distributed to end users for monitoring of arbitrary applications on-device or used by developers to test and evaluate their applications against the given set of API rules, coding best practices, and security properties.

To address the matter of usable properties, we will first consider the ideal language for expressing properties. Because RV-Monitor supports a variety of techniques for writing properties, including raw aspect-oriented properties and

monitor-oriented properties that define both formal mathematical properties and their instrumentation, it is important to note the drawbacks and advantages of both techniques.

3.1 Two Properties in MOP and AspectJ

Blocking Device Location and Network Accesses The first property is a security property intended to block malicious or questionably-sourced applications from accessing a user's location through the Android location API. The second property is not security oriented as the first one is. Instead, it merely logs and notifies the user when an application attempts to use an Android API call related to networking and connections to the Internet.

The Properties in AspectJ We present how to implement monitors for the two properties in AspectJ. Note, the monitors make use of the following imports: import android.app.Activity; import android.content.Context; import android.content.ContextWrapper; import android.widget.Toast;

The monitor for blocking the device location is shown in full in aspect form in Listing 1.1. It prints a message to the user and log noting the attempted location access, and denies the access by returning null to the aspect around() join point rather than proceeding with the original call. It is clear from examining the property that there is a lot of code that can be automatically generated, including the code required to track the current activity in order to show the notification to the user.

The monitor for keeping track of network accesses is shown in full in aspect form in Listing 1.2. The monitor does not attempt to preserve any security properties (such as an unknown application cannot access user location). Instead, it merely logs and notifies the user when an application attempts to use an Android API call related to networking and connections to the Internet.

The monitors for these properties share a large amount of code, which is expected as both are fundamentally performing the same task (notifying the user of some event occurring on the device). This also suggests that there is a significant potential for automatically generating the monitor, and that both monitors could be expressed more concisely.

The Properties in MOP While aspect-oriented programming is one approach to monitoring and analyzing applications on the Android platform [4], a newer paradigm familiar to the runtime verification community involves using the monitoring-oriented programming paradigm. Listing 1.3 shows the same property being enforced as in Listing 1.1 with a monitor defined with significantly less code. Using several keywords provided by RV-Monitor and RV-Android, particularly __TOAST and __SKIP, which in addition to __ACTIVITY form the basic keywords for user interaction currently supported by RV-Monitor, we automatically generate much of the code performing similar functions to the previous monitors in AspectJ.

```
aspect BlockGetLocation extends Activity  {

  private ContextWrapper contextWrapper;
  private Activity activity;
  private Object object;

  Object around(): call(* android.location..*(..))
    && !within(BlockGetLocation)
   {
    String method = thisJoinPoint.getSignature().getName();
    String classe = thisJoinPoint.getSignature().getDeclaringType().
        getName();
    /* Log information about method and class */;
    Object object = thisJoinPoint.getThis();
    Toast.makeText(
          ((android.content.ContextWrapper) object).
              getApplicationContext(),
          "Application accessing location information", Toast.
              LENGTH_LONG).show();
    return null;
  }

   // Advice to get the context application
  after(): execution(void Activity.onCreate(..))
    && !within(BlockGetLocation) {
  try {
    activity = new Activity();
    object = thisJoinPoint.getThis();
    System.out.println((object instanceof Activity));
    contextWrapper = new ContextWrapper(
            ((android.content.ContextWrapper) object)
            .getApplicationContext());
  } catch (Exception e) {
      System.err.println(e.toString());
    }
  }
}
```

Listing 1.1: Aspect-Oriented Property Blocking Location Accesses

Similarly, Listing 1.4 shows the same monitor as in Listing 1.2 using MOP format. Some clear advantages include shorter property that is easier to write, debug, and validate for developers and even advanced end-users.

Neither of these properties, however, leverage the primary feature of RV-Monitor, which is the ability to define logical properties over the event trace of an application in a variety of formalisms.

One example of using such formalisms is shown in Listing 1.5, which combines the previous two properties by denying all network-related API calls after the user's location is accessed. The access is otherwise permitted by the system.

The property is formally defined by a finite state machine in which the first state is safe, as the application has not accessed any location information. The application accessing any location information brings it to an unsafe state, after which any network access lead to a "denied" state. On entering the denied state, the monitor skips the function call to the network, returning null rather than executing that call. The distinction between the unsafe and deny states is that

```
aspect WebAspect extends Activity {
  // Android Internet methods
  pointcut webCall() : call(* android.net..*(..)) || call(* android.
      webkit.WebView..*(..)) || call(* java.net.HttpURLConnection..*(..)
      ) && !within(WebAspect);
  pointcut onCreate(): execution(* onCreate(..)) && !within(WebAspect);

  private ContextWrapper contextWrapper;
  private Activity activity;
  int count;
  after(): webCall() {
  try{
    if(count == 0){
    if(contextWrapper != null && activity != null && count==0){
      activity.runOnUiThread(new Runnable() {
          public void run() {
            count++;
             // Toast message to inform
            Toast.makeText(contextWrapper.getApplicationContext(),
          "Application accessing to Internet", Toast.LENGTH_LONG).show()
              ;
          }
        });
    }
    else { /* Log error about the missing context application */ }
    }
  } catch(Exception e){
     /* Log exception using thisJoinPoint.getTarget().toString() */
    }
  }
  // Advice to get the context application
  after(): onCreate() {
    try {
      count=0;
      activity = new Activity();
      Object object = thisJoinPoint.getThis();
      contextWrapper = new ContextWrapper(((android.content.
          ContextWrapper) object).getApplicationContext());
       } catch (Exception e) { /* Log error message */ }
  }
}
```

Listing 1.2: Aspect-Oriented Property Monitoring Network Accesses

the unsafe state does not skip the call, allowing for further location accesses after the first access, but not for further

Such a property is useful to users concerned about the leakage of their location data, though it is not in its current form comprehensive for all possible data exfiltration channels on Android devices.

In addition to allowing for the addition of logic to Android programming, RV-Monitor can extremely efficiently monitor parametric properties, or properties of a specific Java object, class, or memory location and allowing for millions of monitors to be created with low runtime overhead [6]. This expressiveness and low overhead makes RV-Monitor ideal for a wide range of properties, from simple global properties like the above to complex parametric properties.

```
import java.lang.*;

Android_DenyLocation() {

    // Deny all location package calls
    event location Object around(): call(* android.location..*(..)) {
        __TOAST("Application accessing location information.");
        __SKIP;
    }
}
```

Listing 1.3: Monitor-Oriented Property Blocking Location Accesses

```
import java.lang.*;

Android_MonitorNetwork() {

    // Display application toast on all network API calls
    event web_call after(): call(* android.net..*(..))
      || call(* android.webkit.WebView..*(..))
      || call(* java.net.HttpURLConnection..*(..)) {
        __TOAST("Application accessing the Internet,");
    }

}
```

Listing 1.4: Monitor-Oriented Property Monitoring Network Accesses

3.2 Preventing Security Violations - A Real Attack

While the above properties may be useful for users wishing to gain fine grained control over the privacy of their data when faced with potentially malicious applications, they are generally ineffective in protecting against real attacks. Listing 1.6 shows a MOP monitor in blocking a wide range of attacks. Similarly to the monitors in the previous subsection, this is a simple monitor designed to skip all calls to any exec method in the Runtime class. This method in the Android API allows developers to execute shell code directly through the currently running process, allowing for arbitrary commands interpreted by the Linux kernel underpinning Android [12].

One potential practical application of this property is the disabling of currently available Tor libraries, which are becoming widely used on Android in a malware context [13]. The Tor network allows malware, including spyware, adware, and ransomeware, to obfuscate its network connections to command and control servers, the location of which can be hidden from law enforcement and the public [14]. A 2014 survey of Android security by Google specifically mentions ransomware as an up-and-coming problem needing to be addressed by the Android security team, further suggesting the application of dynamic and static analysis techniques against spyware and other malicious applications which often use Tor to communicate with their operators [15,13].

The above property disbales Tor due to the code in Listing 1.7, which needs to install executable binary blobs of native code. Because there is no native file

```
import java.lang.*;

Android_MonitorLocationNetwork() {

    event web_call Object around(): call(* android.net..*(..))
      || call(* android.webkit.WebView..*(..))
      || call(* java.net.HttpURLConnection..*(..)) {
        __TOAST("Application accessing the Internet,");
    }

    event location Object around(): call(* android.location..*(..)) {
        __TOAST("Application accessing location information.");
    }

   fsm :
     start [
        location -> unsafe
        web_call -> start
     ]
     unsafe [
        web_call -> deny
        location -> unsafe
     ]
     deny [
        web_call -> deny
        location -> unsafe
     ]

    @deny {
        __SKIP;
    }

}
```

Listing 1.5: Monitor-Oriented FSM Property Preventing Network Accesses after Location

permission API on Android, the above aspect blocks all attempts to change file permissions and thus install native code from any third-party applications. In doing so, it blocks a wide range of potential attacks and privilege escalation exploits which rely on shell access, rendering it a powerful practical defense against violations of user privacy by malware that leverages the Tor network.

To test this property, it is sufficient to instrument the "Orbot" application in the Play store and attempt to run it on-device for the first time. We include detailed instructions and a sample of such monitored applications on `https://runtimeverification.com/android`, one of which is a sample of real spyware disabled through the monitoring and enforcement of the above property.

4 Into the Future - Beyond Android Permissions

The future of the Android platform brings a substantial amount of change, with an ever increasing level of the platform's popularity and a significant number of potentially harmful applications still being deployed to user devices regularly [15].

```
import java.lang.*;

Android_BlockShellExec() {

    // Skip all shell execution calls
    event shell_call Object around(): call(* Runtime+.exec(..)) {
    --SKIP;
    }
}
```

Listing 1.6: Monitor-Oriented Property Blocking Shell Calls

```
public static void copyRawFile(Context ctx, int resid, File file, String
     mode, boolean isZipd) throws IOException, InterruptedException
    {
        ... // Copy file
        // Change the permissions
        Runtime.getRuntime().exec("chmod "+mode+" "+abspath).waitFor();
}
```

Listing 1.7: Potentially Malicious Code Installing the Tor Network from Orbot

4.1 Android M - A New Permissions Model

One of the most recently announced changes in the fundamental security model of the Android platform is the change in the permissions system being released with Android M. Rather than simply granting an application blanket permission to perform all operations in a given category, application permissions can be revoked at any time by the user [16]. One example of this is the revocation of location permissions from applications which may be leaking user data to third parties. This replaces many of the simpler security properties and aspects previously written targetting Android permissions, including the location property presented in this paper.

Despite this, as shown in Listing 1.5, RV-Android can provide much finer grained control over how permissions are used on device, revoking or granting permissions dynamically based on logical properties of the application state or current trace. In this way, context-sensitive permissions become possible for cases where a high level of control is desirable. One perfect example of this is the revocation of the Internet permission we discussed, which is now granted by default and without user confirmation to all applications being installed in Android M [17]. This change potentially removes some control from the user related to which applications have network access. However, because almost all applications rely on network access, context-sensitive permissions in which allowed hosts are whitelisted and certain services (like Tor or known ad networks) are blacklisted, and in which the level of action reacts to other system events may be more appropriate for informing the end user and providing them with ultimate control.

Using runtime verification and monitoring, one can control every category of permission and API access, specifically defining the access patterns and data

allowed to the application. By defining such specific and tightly controlled executions, users can notice and prevent malicious or unexpected application behavior. Such control may be useful for employers with employees dealing with sensitive data on Android devices, and in all security-sensitive Android applications. The ability to run third-party applications on such devices introduces security vulnerabilities, a wide range of which can be detected and prevented with runtime verification.

4.2 Integrating with Other Approaches

In addition to preventing security violations as they occur, RV-Android can be used as a testing tool on a large number of applications to detect the presence of such violations in third party applications. In doing so, RV-Android can leverage a number of automated unit testing tools already available for Android designed to mimic and mock user behavior on the platform [18] [19], checking a large number of properties automatically.

In addition to testing, RV-Android could potentially integrate with other dynamic analysis tools, including popular tools designed to ensure data security through taint analysis. Taint analysis is a popular technique that tracks sensitive information as it is handled by applications, preventing it from leaving through any unprivileged API calls or other "sinks" [20] [21]. By integrating many such techniques, we aim to create the foremost security tool and platform for Android, with an ability to implement any future dynamic analysis techniques desirable to users and developers.

Lastly, runtime verification can integrate with static analysis to improve its efficiency and inform the automatic generation of properties checkable at runtime [22]. While there are no concrete plans to do so in RV-Android currently, this remains a future potential research direction.

4.3 Towards a Public Property Database

Undoubtedly the most important work in developing a framework for practical Android runtime verification is the development of properties that thoroughly encompass both known misuses of the Android and Java API's and violations of secure states by previously observed malware. To this end, we are developing a property database which provides annotated copies of both the Android and Java API's with RV-Monitor properties and AspectJ instrumentation built in. The current property database can be viewed and downloaded for free at `https://github.com/runtimeverification/property-db/`, and can be used by RV-Android with minimal modification. The property database currently contains around 180 safety properties of the Java API, which can be practically simultaneously monitored by our technology. These properties differ substantially from the security properties presented in this paper in that they monitor correct use of the API and thus functionality of the application rather than attempting to enforce the security of the device itself. This flexibility of RV-Android in capturing a wide range of potential properties on Android, together

with this robust database of existing properties provides a good starting point for the development of a comprehensive dynamic analysis and runtime verification tool.

We further plan on providing additional privacy profiles focused on stronger security guarantees for the security conscious user, such as profiles useful in avoiding data exfiltration when using specific API's. While we will still allow users to define their own properties, the utility of future iterations of RV-Android will stem partially from the robustness of the default properties in our property database.

Acknowledgements

We would like to thank Patrick Meredith for developing the initial prototype of RV-Monitor applied to Android applications and continued feedback, as well as our partners at ITC and Denso for their continued support in the investigation of Android-related work for the automotive domain. We would also like to thank the miSecurity application team, for providing a basis for the graphical user frontend we describe.

References

1. Google Inc.: Android Developers (2014) `http://developers.android.com`.
2. Falcone, Y., Currea, S., Jaber, M.: Runtime verification and enforcement for Android applications with RV-Droid. In Qadeer, S., Tasiran, S., eds.: Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers. Volume 7687 of Lecture Notes in Computer Science., Springer (2012) 88–95
3. Bauer, A., Küster, J., Vegliach, G.: Runtime verification meets Android security. In Goodloe, A., Person, S., eds.: NASA Formal Methods - 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings. Volume 7226 of Lecture Notes in Computer Science., Springer (2012) 174–180
4. Falcone, Y., Currea, S.: Weave Droid: aspect-oriented programming on Android devices: fully embedded or in the cloud. [23] 350–353
5. Eclipse: The AspectJ project (2014) `http://eclipse.org/aspectj`.
6. Luo, Q., Zhang, Y., Lee, C., Jin, D., Meredith, P.O., Serbanuta, T., Rosu, G.: RV-Monitor: Efficient parametric runtime verification with simultaneous properties. In Bonakdarpour, B., Smolka, S.A., eds.: Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Volume 8734 of Lecture Notes in Computer Science., Springer (2014) 285–300
7. Mulliner, C.: Dynamic binary instrumentation on Android. (2012)
8. Bodden, E.: Instrumenting Android apps with Soot (2014) `http://www.bodden.de/2013/01/08/soot-android-instrumentation/`.
9. Binns, P., Englehart, M., Jackson, M., Vestal, S.: Domain specific software architectures for guidance, navigation and control. Journal of Software Engineering and Knowledge Engineering **6**(2) (1996) 201–227
10. Enck, W., Ongtang, M., McDaniel, P.: Understanding Android security. IEEE security & privacy (1) (2009) 50–57

11. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., Glezer, C.: Google Android: A comprehensive security assessment. IEEE Security & Privacy (2) (2010) 35–44
12. Google Inc.: Runtime — Android Developers (2015) http://developer.android.com/reference/java/lang/Runtime.html.
13. TrendMicro Security Intelligence Blog: Android ransomware uses tor (2014) http://blog.trendmicro.com/trendlabs-security-intelligence/android-ransomware-uses-tor/.
14. PCWorld: Cybercriminals are using the Tor network to control their botnets (2013) http://www.pcworld.com/article/2045183/.
15. Google Inc.: Google report Android security 2014 year in review (2014) https://static.googleusercontent.com/media/source.android.com/en/us/devices/tech/security/reports/Google_Android_Security_2014_Report_Final.pdf.
16. BGR: This will be the most important (and possibly most overlooked) new android m feature (2015) http://bgr.com/2015/05/28/android-m-granular-permissions-controls/.
17. Android Police: Android M will never ask users for permission to use the internet, and that's probably okay (2015) Published on the 2015/06/06 at www.androidpolice.com.
18. Amalfitano, D., Fasolino, A.R., Tramontana, P., Carmine, S.D., Memon, A.M.: Using GUI ripping for automated testing of Android applications. [23] 258–261 http://wpage.unina.it/ptramont/GUIRipperWiki.htm.
19. on Github, W.C.: Swifthand (2015) https://github.com/wtchoi/swifthand.
20. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: ACM SIGPLAN Notices. Volume 49., ACM (2014) 259–269
21. Fritz, C., Arzt, S., Rasthofer, S., Bodden, E., Bartel, A., Klein, J., le Traon, Y., Octeau, D., McDaniel, P.: Highly precise taint analysis for Android applications. EC SPRIDE, TU Darmstadt, Tech. Rep (2013)
22. Bodden, E., Hendren, L.J., Lam, P., Lhoták, O., Naeem, N.A.: Collaborative runtime verification with Tracematches. In: Runtime Verification, 7th International Workshop, RV 2007, Vancouver, Canada, March 13, 2007, Revised Selected Papers. (2007) 22–37
23. Goedicke, M., Menzies, T., Saeki, M., eds. In Goedicke, M., Menzies, T., Saeki, M., eds.: IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012, ACM (2012)