Scalable Testing of Mobile Antivirus Applications

Andrea Valdi, dubizzle.com
Eros Lever, Secure Network S.r.l.
Simone Benefico, Moviri SpA
Davide Quarta, Stefano Zanero, and Federico Maggi, Politecnico di Milano

AndroTotal, a scalable antivirus evaluation system for mobile devices, creates reproducible, self-contained testing environments for each antivirus application and malware pair and stores them in a repository, benefiting both the research community and Android device users.

For most people, mobile devices have become a digital wallet that they can trust to securely hold everything from contacts and appointments to banking and retail transactions. The growth of smartphones, tablets, phablets, and wearables is evidence that mobile devices are becoming essential to daily life, with mobile applications limited only by developers' imaginations. Android devices and applications, in particular, are in wider use; as of September 2015, Android held 82.8 percent of the mobile device market,[1] with Google Play Store reporting more than 150 billion downloads as of April 2015 and offering 1.6 million individual apps as of July 2015 (www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store). This number excludes applications offered exclusively on alternative markets such as Amazon, Samsung Apps, AndroLibs, and AppBrain.

The heavy reliance on mobile devices for trusted information combined with the pervasiveness of mobile devices and applications is attractive to cybercriminals, who develop and distribute malware to steal sensitive data and compromise banking and other electronic services. Although Android has some security countermeasures and many antivirus (AV) applications exist, the security community lacks a convenient and reliable way to test them.

The main challenge of scientifically evaluating AV applications is how to reproduce the exact conditions an AV encounters when running on a user's device, such as network connectivity, OS version, and system load. Existing AV evaluation approaches tend to rely on time-consuming tasks that require a high degree of user intervention, or they ignore the need to ensure that testing conditions are reproducible. Ideally, testing must fully emulate the entire application stack, which is not practical. Thus, existing methods rely on human-assisted tests and reviews, which require heavy manual intervention and can manage only 25 to 30 application scans per person-day of testing.

To provide Android device users with deeper assurance that their applications are secure, we developed AndroTotal, which is based on the model that VirusTotal successfully implemented in the desktop world. Like VirusTotal desktop users, any Android device user can submit an application to a website to check how it is classified by commercial mobile AV products. Unlike existing methods, AndroTotal uses a completely automatic approach to scan hundreds of suspicious applications per day against all major AV application versions. Unlike VirusTotal, it creates reproducible, self-contained testing environments for each AV–malware pair, while ensuring a high throughput because of its inherent scalability.

With AndroTotal, users can be assured that the desired application is clean according to the best

TABLE 1. Top 20 antivirus applications in Google Play Store as of October 2015.

No. of installations (millions)	Product
100 to 500	AVG Mobile Antivirus Free
	AVAST Software Antivirus & Security
	CM Security Antivirus AppLock
	Security & Antivirus | Lookout
	360 Security, Antivirus Free
50 to 100	Psafe Antivirus Booster & Cleaner
	Antivirus Dr.Web Light
10 to 50	McAfee Antivirus & Security
	Kaspersky Internet Security
	Avira Antivirus Security
	Norton Security and Antivirus
	Trustlook Antivirus & Mobile Security
	TrustGo Antivirus & Mobile Security
	NQ Security Lab Antivirus Free
5 to 10	ESET Mobile Security & Antivirus
1 to 5	Malwarebytes Anti-Malware
	Bitdefender Antivirus Free
	Panda Free Antivirus and Security
	Itus Antivirus for Android
	Bitdefender Mobile Security & Antivirus

available information to date, while AV application developers benefit from AndroTotal's generic, scalable approach for mobile AV application testing, which is openly accessible as a Web service that conducts thorough, precise, and repeatable tests against existing or novel malware. Finally, researchers and AV vendors benefit from AndroTotal's ability to create an indexed database of mobile malware samples, which is useful in updating products and conducting more in-depth research on a particular threat.

The research community has reacted positively to AndroTotal's release, with several AV vendors automating submission samples to our system and retrieving samples for their own analysis. We are also cooperating with other well-known Android malware research projects, including CopperDroid (http://copperdroid.isg.rhul.ac.uk) and Andrubis (https://play.google.com/store/apps/details?id=org.iseclab.andrubis&hl=en).

After nearly two years of operation, we have been able to collect user feedback and observe how researchers employ AndroTotal. We have also taken quantitative measurements, such as detection rates or malware classification and misclassification rates.

AVAILABLE ANDROID COUNTERMEASURES

The last few years have seen an alarming spike in Android malware diversity[2] and a steady increase in malware complexity.[3] Current Android security measures to combat this malware include code signing and signature verification when the user installs an app. Android's "Verify Apps" setting sends the hash of each installed application to Google servers: if the hash corresponds to known malware, a warning is displayed. Moreover, Google Play Store enforces developer verification and uses Bouncer, a system that screens submitted applications. However, studies have shown that Bouncer can be circumvented.[4] Other than these countermeasures, Android device users have no protection from malicious intrusions without downloading an AV application.

Users seem to understand the need for these applications, according to our review of Google Play Store statistics from 2013 to 2015. As Table 1 shows, AV application installations are increasing to the 500 million mark. However, there is no guarantee that an AV application will work as promised. For example, in early 2014, 70 of the 100 AV applications in Google Play Store were Android specific, meaning that no (more mature) desktop equivalent existed and that the applications' developers were likely to be untested market players. The lack of a known AV company behind these applications raises obvious questions about their trustworthiness and efficiency.

AV APPLICATION TESTING CHALLENGES

The problem of testing an AV application under consistent conditions becomes much thornier if tests must scale. In a desktop environment, testing

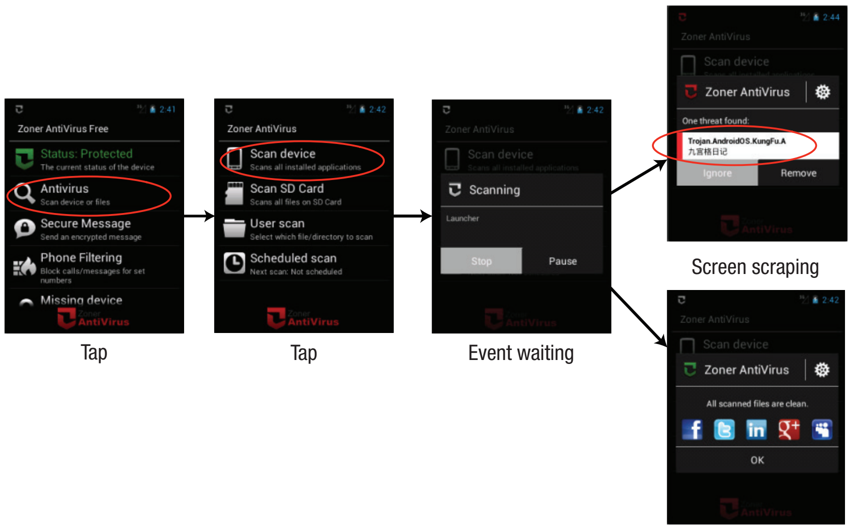

FIGURE 1. User interaction needed to perform an on-demand device scan with Zoner AntiVirus Free v. 1.7.0. Although the actions seem basic, they are difficult to automate.

consists of creating scripts that automate command-line versions of the AV program. Unfortunately, these versions are poorly maintained and inherently limited because detection relies solely on signature matching. In other words, testing ignores behavioral heuristics, or techniques triggered by network communication or interprocess communication (IPC) anomalies. Testing also fails to consider the AV application's working environment, and does not exercise the entire application. It is no surprise that script-based testing has been heavily criticized for failing to reflect the actual AV capabilities in real-world conditions.[5] VirusTotal implements the script approach, and its authors are careful to clearly point out these limitations (www.virustotal.com/en/about/best-practices).

In the mobile world, the same consistency and scalability challenges apply. Several products, such as SRTAppScan, use behavioral heuristics; others, such as Kaspersky, analyze network-traffic anomalies. Some AV applications, such as TrendMicro and SourceFire, use in-cloud scanning, verifying samples through interrogations with a remote service, rather than against a local database. In our view, these applications are a solid argument for developing a completely different approach to mobile AV application testing.

Automating a device scan

Mobile AV applications generally have highly interactive GUIs, making them technically difficult to automate. Figure 1 shows the gestures needed to perform a device scan with Zoner Antivirus Free. Automating even these basic steps requires creating a program that can emulate tapping and then wait for the displayed results—not a trivial undertaking. Add customized view components or other tailored graphical elements not provided by the Android software development kit (SDK), and automation becomes even more difficult.

Another scanning challenge is that mobile AV applications work in multiple detection modes, all of which must be supported by a testing system. The two most popular detection modes are *on demand*, in which the user requests a device scan, and *on install*, in which the AV application waits for an application programming kit (APK) to be installed—registering a broadcast receiver for the `ACTION_PACKAGE_ADDED` or `ACTION_PACKAGE_INSTALL` intents—and analyzes it on the spot.

Automatically capturing detection results is also problematic. The Android GUI is asynchronous, which makes this operation technically complex because many AV applications use the notification bar, and other applications are not allowed to capture those notifications by design.

Modifying the Android framework or AV application might alleviate some of these automation obstacles, but doing so would violate the main goal of preserving real-world working conditions as much as possible.

Existing approaches

Researchers, practitioners, and vendors have proposed a variety of mobile security methodologies, many of which are listed on the AV Comparatives website (www.av-comparatives.org/mobile-security). However, they all require time-consuming manual tasks, including

- preparing a clean testing image (about 2 to 3 minutes per image);
- installing the AV application and the suspicious application (about 2 to 5 minutes); and
- restoring a clean system state (about 10 to 15 minutes).

For this reason, testing becomes largely manual, with a throughput of 25 to 30 suspicious applications scanned per AV application per person-day invested in testing.

Although some mechanisms can automate the creation of a clean testing image and the installation of suspicious applications, even state-of-the-art tools[6,7] still require an operator to check the outcome of each scan, which significantly limits scalability.

Other researchers have concentrated on building resiliency to malware mutations by applying multiple transformations (repackaging or obfuscation) to the samples before testing them with AV applications.[8] Although this work is technically interesting, it does not focus on ensuring that the tests are run under reproducible conditions, which we believe is essential to the scientific evaluation of AV applications.

ANDROTOTAL DESIGN

A 2013 survey of mobile malware variety and complexity and an analysis of existing AV application testing methods[9] laid the foundation for developing AndroTotal. After establishing a set of design requirements, we reviewed and discarded any testing approaches that did not meet our requirements.

Our main challenge was increasing scalability. A possible approach was to simultaneously install a set of AV applications and samples in a single emulator instance or device. However, we rejected that strategy because it can lead to unwanted interactions between the AV applications and between the samples, which could result in biased outcomes.

Instead, we chose to abstract the environment preparation and testing procedure so the system could create tests and schedule them automatically, allowing us to scale the system by simply adding computational nodes. Because AndroTotal does not require software modification or special hardware and works in emulated environments, we could easily deploy it on cloud infrastructures for additional scaling.

After reviewing scalability demands, we selected six critical requirements for AndroTotal:

1. Input stimulation on the device's GUI by reproducing the typical gestures a user would perform with an AV application.
2. Obtain feedback about the device's displayed views and activities so that it can synchronize testing procedures with the running AV application's state. In addition, scrape information from the display to retrieve data such as the name of an identified threat.
3. Support complex testing procedures that involve multiple applications (an AV application and a browser, for example), as well as execute basic operations such as notification management, which still require accessing different Android application contexts.
4. Conduct testing without AV application modification. Any modification to the AV package to enable testing—including injecting code, changing its signature, or repackaging—might alter its true behavior, which could bias test results.
5. Support any Android version.
6. Natively support Android notification operations such as waiting for a notification to appear, handling an open notification, and checking for notifications, because notifications are the only feedback for some AV applications.

ANDROTOTAL IMPLEMENTATION

AndroTotal supports both on-demand and on-install detection modes, exposing a Python API to automate test procedures and gather results through GUI scraping. AndroTotal's key difference from existing systems that scan multiple AV applications is that it runs the actual AV application on a real or emulated Android device. During each test, AndroTotal captures screenshots from the application, the network dump, and the log file, although it does not rely on the log file to derive the detection label because not all mobile AV applications log that information. An approach based on GUI and screen scraping is more general, flexible, and less constrained than a log parser.

AndroTotal gives the user access to the analyses (if any) generated by VirusTotal, CopperDroid, ForeSafe, and SandDroid as additional information sources. We have set up data-sharing agreements with the VirusTotal and CopperDroid services.

Test automation

After analyzing the six most promising, publicly available libraries to support testing automation,[9] we did not find any suitable candidates. Robotium excelled in white-box testing, but requires application resigning and has limited application sandboxing, which did not meet requirements 4 and 5. Android's Monkey and Monkeyrunner met all requirements except requirement 2 because they do not support data retrieval from a running device or emulator. Android's UI Automator supports only Android SDK API 16 or higher, which did not satisfy requirement 5.

AndroidViewClient and Apk-viewtracer were good tradeoffs, as both effectively simulate typical user inputs and can retrieve information about displayed activities. Under the hood, they rely on Monkey and Monkeyrunner but also implement the missing functionality to satisfy requirement 2. They support all Android versions and do not need any package modifications to interact with an application. However, notification support and most of the other implemented functions were unstable and slow.

Because of these shortcomings, we decided to build and implement our own test automation library. AndroPilot is the first Android AV-application-automation library written solely in Python that supports all the functionalities needed to conduct AV application tests.

To build AndroPilot, which became the foundation of AndroTotal's scalable architecture, we extended Apk-view-tracer by reimplementing part of the existing code to improve stability, fixing several bugs and correcting suboptimal design choices. We also enhanced the library speed.

To correct the design choices, we leveraged Android's ViewServer component and introduced new procedures to properly manage application synchronization during testing stages, including functions that wait for an arbitrary view, text, or notification to appear on the screen. We also improved view management to correctly report when a view is shown on the running Android instance and implemented a new function to retrieve the screenshot from an attached device or emulator.

Creating an AndroPilot adapter module for an AV application was straightforward and took little coding effort. For the 10 adapters currently implemented, our libraries enabled adapter implementation with an average of 36 lines of Python code per adaptor (as measured with count lines of code; http://cloc.sourceforge.net).

Figure 2 shows the code for two of these adapter implementations.

Architecture

AndroTotal's workflow begins when a user submits an Android application (APK package) to AndroTotal's Web interface. If AndroTotal has not yet analyzed the application, it pushes the application to the analysis queue as a series of tasks (one for each AV application or submitted application pair). Worker servers execute the tasks, each of which is treated as an execution unit, using concurrent multiprocessing. When a worker server receives a task, it starts an emulator with a clean image, installs the application sample, performs the required tests, and stores the results in a database.

A test is essentially a Python script written on top of AndroPilot, which runs a given AV application either in on-demand or on-install mode and retrieves the results through GUI scraping. AndroTotal then stores the results in its database and returns them to the user. It also exposes a Representational State Transfer (REST) API, which ensures interoperability with external services.

Ensuring scalability. We ensure AndroTotal's scalability by making each testing procedure self-contained so that a single worker server can perform each test job (task) independently. By leveraging the Android emulator's snapshot function, AndroTotal can run a test in an average of 1 to 3 minutes. To store the Android image and run the emulator, each test requires 50 to 250 Mbytes of temporary disk space and 1 to 2 Gbytes of RAM. Once the test terminates, the worker server will clear the temporary files and ensure that the emulator has correctly terminated and freed the used resources. The time to scan an application or malware sample against a set of AV applications grows linearly with the number of AV applications, ensuring that parallelization can provide scalability.

Supporting multiple application versions. Unlike similar services such as VirusTotal, AndroTotal maintains multiple versions of the same AV application over time. In this way, it allows the testing of new samples against older versions as well as the computing of evolution statistics. By accessing scan results such as logcat, network dumps, and screenshots, users can visualize and download the data associated with each AV application test. By aggregating data from various reports, AndroTotal also provides insight into why a sample might be malicious.

AndroTotal adapters contain an automated function that checks for AV signature updates and automatically performs them, modifying the image as needed. These tasks are asynchronous and do not affect AndroTotal's throughput.

New application versions are handled through a semiautomated procedure. AndroTotal monitors Google Play Store each day for new AV application releases and notifies the AndroTotal maintainers when one is found. The maintainers use an automated script to initialize a clean image of the new AV release, manually test the current AV adapter and adjust it to deal with any changes in the application's user interface, and plug the new image and adapter into the AndroTotal system.

EVALUATION

As of early October 2015, 2,491 users have requested access to and are actively using AndroTotal. This has enabled us to collect 85,677 distinct samples of malicious and benign

```
class TestSuiteProductX(base.BaseTestSuite):
    def detection_on_demand(self, sample_path):
        “”” Test the AV’s capability of detecting malware by scanning
        the whole device.
        “””
        # Connect to a running device
        p = self.pilot
        # Install application sample
        p.install_package(sample_path)

    def detection_on_install(self, sample_path):
        “””Test the AV’s capability of detecting malware upon installation.
        “””
        p = self.pilot
        p.install_package(sample_path)
```

```
class TestSuiteProductY(base.BaseTestSuite):
    “”” Test suite for ProductY
    “””
    def detection_on_install(self, sample_path):
        “””Test the AV’s capability of detecting malware upon installation.
        “””
        # Connect to a running device
        p = self.pilot
        if sample_path: # Install sample on the running device
            p.install_package(sample_path)

        time.sleep(2)  # Sleep
        self.__check_popup()  # Check if detected

    def detection_on_copy(self, sample_path):
        “”” Test the AV’s capability of detecting a malware when the
        sample is copied on the device.
        “””
        p = self.pilot
        if sample_path:
            p.push_file(sample_path)

        time.sleep(2)
        self.__check_popup()

    def __check_popup(self):
        “”” Check if the alert popup is displayed
        “””
        p = self.pilot
        # Wait for the antivirus’s screen to appear
        if p.wait_for_activity(
                “com.kms.free.antivirus.gui.AppCheckerAlert”, 10):
            # Tap on button to start scan
            p.tap_on_coordinates(120, 210)

            if p.wait_for_activity(“com.kms.free.antivirus.gui.AppCheckerVirusAlert”,
                                   30, critical=False):
                p.refresh()
                threat_view = p.get_view_by_id(“ObjectType”)
                # Extract the threat name
                self.result[‘detected_threat’] = threat_view.mText.strip()
            else: # No threat found
                self.result[‘detected_threat’] = config.NO_THREAT_FOUND
        else: # No threat found
            self.result[‘detected_threat’] = config.NO_THREAT_FOUND
```

FIGURE 2. Python code for two implementations of the 10 currently implemented AndroPilot adapter modules. Our libraries enabled adapter implementation with an average of only 36 LOC per adapter.

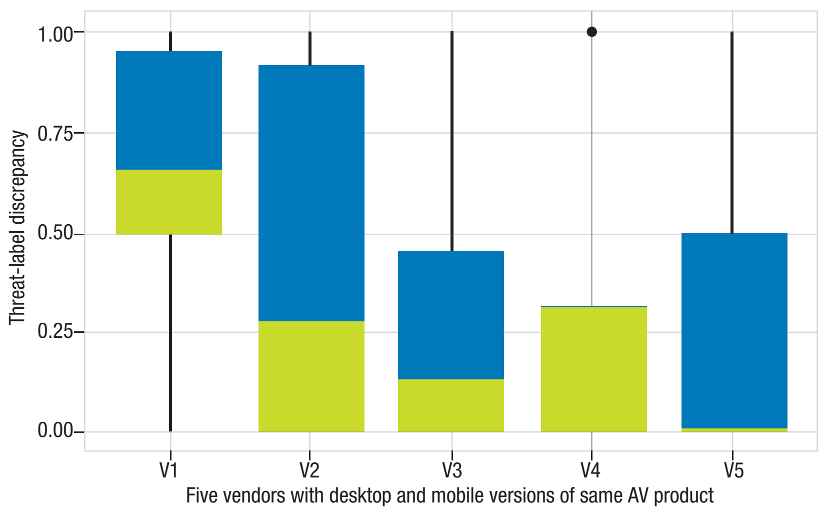

FIGURE 3. Discrepancies in threat labels between AV products offered by the same vendor. Of five vendors that supported both a desktop and mobile version of the same AV product, only one (V5) had nearly zero discrepancies between its two versions. Label discrepancy is the result of dividing the edit distance between two labels by the length of the longer label and then mapping the result to a number in an interval from 0 to 1.

Android applications. Several researchers and research groups and five major AV product vendors have requested access to this dataset. The data, along with user feedback about AndroTotal use, has revealed interesting issues, including the need for full emulation as well as discrepancies in the mobile and desktop versions of AV applications.

Full versus partial emulation

One use case involved implementing both VirusTotal and AndroTotal to evaluate the detection rate of mobile AV applications against an advanced proof-of-concept malware that contains a call-home function to download additional malicious code after the first execution.[4] None of the AV tests that VirusTotal executed triggered the call-home function, demonstrating that VirusTotal's testing environment did not create the necessary conditions for the malware to work. In contrast, during AndroTotal tests, the malware was able to complete the procedure as if it were working on a real device.

AV application mismatch

It is difficult to conduct a direct and fair comparison of VirusTotal and AndroTotal because of inherent differences in desktop programs and mobile applications, such as the procedure for behavioral checks and malware detection (on the device as opposed to in the APK).

With these comparison limitations in mind, we automatically and manually compared VirusTotal and AndroTotal performance on 300 randomly chosen malware samples, focusing on five AV application vendors (V1–V5) that were common to both.

For each sample and vendor, we calculated the edit distance (number of different characters between strings) between the threat label that VirusTotal and AndroTotal detected (`INI:SMSSend-A [Trj]` versus `Android:FakeNotify-A [Trj]`, for example). Although we expected the same vendor to label the same samples consistently, as Figure 3 shows, only one of the five chosen vendors did this. This discovery validates the results of a 2011 study, which also found naming inconsistencies in malware.[10]

In addition to showing labeling discrepancies, this comparison reinforces the need for full-emulation AV testing approaches.

Time requirements

A worker server takes between 50 seconds (s) and 5 minutes (min) to run a test on eight malware detectors, including the time required to launch all the emulators. On average, a single test takes 1 to 3 min to complete; the standard deviation is too wide to rate relative speed among vendors.

Threat labels

By querying the AndroTotal database for the number of tests of each AV APK, grouped by output label, we determined the most popular threat labels.

Table 2 shows the 15 most popular threat labels detected as of early October 2015. We assumed that popularity was proportional to the number of distinct (in terms of the MD5 message-digest algorithm) sample APKs uploaded with that label. Several of these labeled threats are adware, not malicious applications. It is difficult to differentiate between the two. As others have noted,[11] the boundary is becoming increasingly gray, and defining it will require further investigation.

AndroTotal has been well received by users, researchers, and vendors, and we are already evaluating physical parameters such as battery consumption. Although we plan to continue performing tests on physical mobile devices, it is challenging to create self-contained, repeatable tests on physical hardware because of the time required to "freeze" the device state and restore the same testing conditions. Reflashing a chosen NAND partition on an Android device can take from 2 to 10 min, depending on the device model, the partition to reflash, and the need to reconfigure the system after reflashing is complete.

Recent hardware virtualization support of ARM processors is a possible alternative to conducting repeatable tests on physical hardware that will eventually allow AndroTotal to provide information about accurate AV application performance on real devices.

We are also working to address AndroPilot's slight performance limitations. At present, retrieving the displayed view tree takes up to 30 s for a complete screen dump in extreme cases—a delay due to Android's ViewServer component. We are currently working on patching this component by adapting an existing but outdated patch (http://code.google.com/p/android-app-testing-patches), which should yield a 20× to 40× speedup.

Because of its modular architecture, AndroTotal could also be a valuable starting point to build a more generic testing framework for mobile applications, beyond the specific scope of mobile AV applications. With an open, noncommercial tool such as AndroTotal, researchers can execute repeatable, scientifically sound tests at scale, simplifying application validation and verification and greatly aiding the maturation of AV application testing. 🅲

TABLE 2. Top 15 threat labels as of October 2015.

Label	No. of application programming kits
UDS:DangerousObject.Multi.Generic	7,496
Android.Trojan.140227	2,745
HEUR:Trojan-SMS.AndroidOS.Opfake.bo	2,439
Adware.Airpush.origin.7	2,374
Trojan!Depositmobi.A@Android	2,302
Android.Trojan.Generic	2,176
TrojanSMS.Boxer.AQ.Gen	1,555
Trojan.FakeInst.M	1,330
HEUR: Trojan-SMS.AndroidOS.Opfake.a	1,213
Trojan!RuWapFraud.D@Android	1,204
HEUR:Trojan-SMS.AndroidOS.FakeInst.a	1,077
AndroidOS_Opfake.CTD	1,046
Adware.AndroidOS.Airpush-Gen	1,041
Android.SmsSend.origin.281	965
Android:FakeNotify-A [Trj]	947

ACKNOWLEDGMENTS

We thank the anonymous reviewers who significantly contributed to the improvement of our research. This research was funded in part by the EU Seventh Framework Programme (FP7/2007-2013) under grant 257007 and by the Italian Ministry of Education, University, and Research through the TENACE PRIN Project (20103P34XC).

REFERENCES

1. IDC, "Smartphone OS Market Share, 2015 Q2," 2015; www.idc.com/prodserv/smartphone-os-market-share.jsp.
2. B. Uscilowski, "Security Response: Mobile Adware and Malware Analysis," Symantec, Oct. 2013; www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/madware_and_malware_analysis.pdf.
3. ESET, "Trends for 2013: Astounding Growth of Mobile Malware," Dec. 2012; http://go.eset.com/us/resources/white-papers/Trends_for_2013_preview.pdf.
4. S. Poeplau et al., "Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications," *Proc. 21st Ann. Network and Distributed System Security Symp.* (NDSS 14), Feb. 2014; doi:10.14722/ndss.2014.23328.
5. D. Harley, "There's Testing, Then There's VirusTotal," blog, 10 Dec. 2012; http://blog.isc2.org/isc2_blog/2012/12/theres-testing-then-theres-virustotal.html.
6. H. Pilz, "Building a Test Environment for Android Anti-Malware Tests," Virus Bulletin Conf., Sept. 2012;

www.avtest.org/fileadmin/pdf/publications/vb_2012_avtest_paper_building_a_test_environment_for_android_anti-malware_tests.pdf.

7. R. Ramachandran, T. Oh, and W. Stackpole, "Android Anti-Virus Analysis," *Proc. 7th Ann. Symp. Information Assurance and Secure Knowledge Management* (ASIA 12), 2012; www.albany.edu/iasymposium/proceedings/2012/11-Ramachandran_Oh%26Stackpole.pdf.
8. M. Zheng, P.P.C. Lee, and J.C.S. Lui, "ADAM: An Automatic and Extensible Platform to Stress Test Android Anti-Virus Systems," *Proc. 9th Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment* (DIMVA 12), 2013, pp. 82–101.
9. F. Maggi, A. Valdi, and S. Zanero, "AndroTotal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors," *Proc. 3rd ACM Workshop Security and Privacy in Smartphones and Mobile Devices* (SPSM 13), 2013, pp. 49–54.
10. F. Maggi et al., "Finding Nontrivial Malware Naming Inconsistencies," *Proc. 7th Int'l Information Systems Security* (ICISS 11), 2011, pp. 144–159.
11. LookOut, "Uncovering Privacy Issues with Mobile App Advertising," blog, 9 Jul. 2012; https://blog.lookout.com/blog/2012/07/09/mobile-privacy-app-advertising-guidelines.

ABOUT THE AUTHORS

ANDREA VALDI is a software engineer at dubizzle.com, a large classified portal for the Middle East and North Africa (MENA) region. His research interests include Android malware and antivirus solutions, Web application security, and distributed systems. While conducting the research reported in this article, he was a graduate student at Politecnico di Milano and lead developer of AndroTotal. Valdi received an MSc in computer engineering from Politecnico di Milano. Contact him at andrea.valdi@mail.polimi.it.

EROS LEVER is an information security engineer at Secure Network S.r.l. His research interests include penetration testing and code review for both mobile devices and Web application security, and the development of security platforms and automated tools for security analysis. While conducting the research reported in this article, he was a graduate student at Politecnico di Milano. Lever received an MSc in computer engineering from Politecnico di Milano. Contact him at e.lever@securenetwork.it.

SIMONE BENEFICO is an IT security analyst at Moviri SpA. His research interests include application security, intrusion-detection systems, and side-channel attacks on mobile devices. While conducting the research reported in this article, he was a graduate student at Politecnico di Milano. Benefico received an MSc in computer engineering from Politecnico di Milano. Contact him at simone.benefico@moviri.com.

DAVIDE QUARTA is a PhD student in the Department of Electronics, Information, and Bioengineering at Politecnico di Milano and the current lead developer of AndroTotal. His research interests include mobile malware, embedded systems security, and exploitation techniques. Quarta received an MSc in computer engineering from Politecnico di Torino. Contact him at davide.quarta@polimi.it.

STEFANO ZANERO is an associate professor in the Department of Electronics, Information, and Bioengineering at Politecnico di Milano. His research interests include systems security, computer virology, and applications of machine learning to security data. Zanero received a PhD in computer engineering from Politecnico di Milano. He is a Senior Member of IEEE and ACM, a member of the Board of Governors of the IEEE Computer Society, a member of the international board of directors of the Information Systems Security Association (ISSA), and an ISSA Fellow. Contact him at stefano.zanero@polimi.it.

FEDERICO MAGGI is an assistant professor in the Department of Electronics, Information, and Bioengineering at Politecnico di Milano. His research interests include the analysis of Android malware, botnets, Web services, and social network abuses; anomaly-based intrusion detection; and machine-learning-based approaches to detect unexpected network payloads. Maggi received a PhD in computer engineering from Politecnico di Milano. He is a member of ACM, general chair of the 2015 International Conference on the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 15), and a peer reviewer for *IEEE Transactions on Dependable and Secure Computing*. Contact him at federico.maggi@polimi.it.