TriggerZoo: A Dataset of Android Applications Automatically Infected with Logic Bombs

Jordan Samhi, Tegawendé F. Bissyandé, Jacques Klein
SnT, University of Luxembourg, Luxembourg, {jordan.samhi, tegawende.bissyande, jacques.klein}@uni.lu

ABSTRACT

Many Android apps analyzers rely, among other techniques, on dynamic analysis to monitor their runtime behavior and detect potential security threats. However, malicious developers use subtle, though efficient, techniques to bypass dynamic analyzers. Logic bombs are examples of popular techniques where the malicious code is triggered only under specific circumstances, challenging comprehensive dynamic analyses. The research community has proposed various approaches and tools to detect logic bombs. Unfortunately, rigorous assessment and fair comparison of state-of-the-art techniques are impossible due to the lack of ground truth. In this paper, we present TriggerZoo, a new dataset of 406 Android apps containing logic bombs and benign trigger-based behavior that we release only to the research community using authenticated API. These apps are real-world apps from Google Play that have been automatically infected by our tool AndroBomb. The injected pieces of code implementing the logic bombs cover a large pallet of realistic logic bomb types that we have manually characterized from a set of real logic bombs. Researchers can exploit this dataset as ground truth to assess their approaches and provide comparisons against other tools.

1 INTRODUCTION

The Android operating system is the most used worldwide in mobile devices [17]. Hence, Android security and privacy have become one of the major concerns of researchers. Every year, several thousands of threats are identified by antivirus companies spanning a wide range of maliciousness (e.g., trojan, adware, spyware, ransomware, etc.). To cope with malicious code proliferation, researchers set up several approaches that rely on static analysis [11, 12, 18, 28], dynamic analysis [25, 32, 35], machine-learning based analysis [8, 24, 26], or hybrid approaches [6, 7, 34].

Nowadays, malicious developers build their codebase to avoid detection from analyzers [5, 9, 13, 27, 29]. A notable technique used to bypass dynamic analyses consists in employing *logic bombs* that allow the malicious code to be triggered only under specific circumstances (e.g., at a specific date). In recent years, researchers have therefore proposed various techniques to uncover logic bombs in Android applications (apps) [13, 23, 29]. However, a common challenge in advancing state of the art is the lack of shared benchmarks for the assessment and fair comparison of literature approaches.

The research literature already proposed various datasets of Android apps to encourage reproducibility and comparison between different approaches. For instance, Allix et al. proposed Androzoo [1], a growing repository now including about 18 Million Android apps. Arzt et al. released DroidBench [3], a test suite to evaluate Android taint analysers. Nielebock et al. proposed AndroidCompass [22] as a dataset of Android compatibility checks. Recently, Wendland et al. [33] released AndroR2, a dataset of bug reports related to Android apps and Li et al. [20] released AndroCT, a large-scale dataset of runtime traces of benign and malicious Android apps. However, in the research directions related to logic bombs, the community faces a challenge to build a comprehensive dataset due to the known difficulties in detecting logic bombs. Indeed, even if an app is detected as malware, identifying a logic bomb in malware requires extensive manual inspection and strong expertise. Logic bombs are often simple *if statements* with "unusual" conditional expressions. Yet, it is far from being trivial to distinguish a "logic bomb condition" from a "legitimate and normal condition". The research community lacks an important artifact in the logic bomb detection domain, i.e., an Android app dataset that contains logic bombs with information about their localization in the apps.

In this work, we propose a new dataset of Android apps containing logic bombs and benign trigger-based behavior to the research community. This dataset, named TriggerZoo, contains 406 apps, from which 240 are infected with logic bombs, and 166 apps contain benign trigger-based behavior. It was generated by applying our dedicated tool, AndroBomb, on 2000 apps from Google Play. TriggerZoo is meant to facilitate research on logic bomb detection. Specifically, TriggerZoo will serve as a base for new approaches to detect logic bombs to assess new tools and compare with other approaches. Besides, since AndroBomb has been developed with a modular approach, it is easy to add new trigger and behavior types.

The main contributions of our work are as follows:

- We propose TriggerZoo, a new reusable dataset of 406 Android apps infected with trigger-based behavior, and their localization with 10 trigger types and 14 behavior types.
- We also propose AndroBomb, an extensible framework to inject trigger-based behaviors into Android apps automatically.
- We provide performance results of two state-of-the-art works Difuzer and TriggerScope on the TriggerZoo dataset.

TriggerZoo apps are made available in the AndroZoo repository, where they are responsibly shared with authenticated researchers only. TriggerZoo apps' hashes, and labels are available at:

https://github.com/JordanSamhi/TriggerZoo[1]

In the same way, and to avoid encouraging malware development, AndroBomb is only available to authenticated researchers. AndroBomb's instructions and appropriate files are available at:

https://github.com/JordanSamhi/AndroBomb[2]

2 LOGIC BOMBS

A logic bomb is a piece of malicious code triggered under very specific circumstances. It means that the malicious code is segregated from the normal execution of the benign code and is only triggered under specific criteria. For instance, if attackers want to target devices in Russia *and* that have two cameras (i.e., a back and a front one), they can rely on the code illustrated in Listing 1.

```
1  public class MainActivity extends Activity {
2    protected void onCreate(Bundle b) {
3      TelephonyManager tm = (TelephonyManager) this.getSystemService("phone");
4      String countryCode = tm.getNetworkCountryIso();
5      if(countryCode.equals("RU")){
6        Camera.PictureCallback cb = new Camera.PictureCallback(){
7          public void onPictureTaken(final byte[] b, Camera c)
8            {/* send photo to http server + data theft*/}
9        }
10       if (Camera.getNumberOfCameras() >= 2)
11         {Camera.open(1).takePicture(null, null, cb);}
12     }[...] // benign behavior
13   }
14 }
```

Listing 1: An illustration of a logic bomb in an Android app

Indeed, on line 5, the developer verifies if the device is connected to a Russian mobile network (the ISO-3166-1 alpha-2 country code RU represents the Russian country). On lines 11-13, a photograph is taken from the front camera if the device has at least two cameras and is sent over the network, breaking the user privacy. In this case, the first specific circumstance is that the device is connected to a Russian mobile network (i.e., the Russian population is targeted), and the second one is that the device contains two cameras (i.e., eliminating emulators and old devices.). This example shows that the malicious code is only executed on specific conditions that are unlikely to happen when dynamically analyzing the app in a sandbox. The app's maliciousness will therefore escape detection.

In the rest of the paper, we refer to *trigger type* for the type of the condition that triggers the logic bomb or the benign trigger-based behavior (e.g., at a specific time), and to *guarded code type* for the type of the behavior triggered (e.g., data theft). The interested reader can refer to [29] for further details on the logic bomb concept.

3 DATASET CONSTRUCTION METHODOLOGY

In this section, we first describe AndroBomb used to construct TriggerZoo. Secondly, we give details about the process we followed to generate our dataset and describe it.

3.1 AndroBomb: Automatically Infect Apps

AndroBomb has been designed to inject a trigger-based behavior (malicious or benign) in a specific location in an Android app. This trigger-based behavior is characterized by a *trigger type* (e.g., time check) and a *guarded code type* (e.g., the stealing of private information). In Figure 1, we present an overview of the AndroBomb approach. AndroBomb is made of three main parts: ❶ a mechanism for pinpointing an insertion point based on callgraph construction and control flow analysis that serves to identify a method in which a trigger-based behavior can be inserted; ❷ an infection step where a trigger-based behavior is generated given a *condition type* and a *guarded code type*, and inserted in the insertion point; and ❸ a repackaging step where the APK is updated with new permissions (if required), new native code files (if required), aligned and signed to generate an infected APK file.

3.1.1 Insertion point Pinpointing Mechanism. We intend to inject trigger-based behavior into Android apps. The idea is to inject these trigger-based behavior in methods ❶ highly likely to be executed at runtime, ❷ present in the developer code. To find these methods, AndroBomb relies on FlowDroid [3] and Soot [30] which provide a control flow analysis that is used to generate a callgraph. We set FlowDroid to use the SPARK algorithm [19] to generate a callgraph. Then, AndroBomb builds a set of methods M that contains all the methods in the APK that are declared in a class for which the fully qualified name starts with the app's package name, i.e., it is a developer class. Indeed, we want to inject the logic bomb in the developer code to simulate malicious intent from the developer. M is then filtered to produce a new set of methods M_{cg} that only retains methods that are present in the callgraph previously generated, i.e., they may be called during execution. Eventually, AndroBomb randomly choses a method in M_{cg} that will act as the *insertion point*.

3.1.2 Infection. To infect an app, AndroBomb needs two information: ❶ the trigger type used to activate the trigger-based behavior; and ❷ the guarded code type. This information is not static and is given as options to AndroBomb. As already mentioned, AndroBomb relies on FlowDroid, hence AndroBomb manipulates Jimple code [31] which is the language used to perform code instrumentation [2] and code injection. After dynamically generating the class and the method in which the code generated will lie, AndroBomb generates the trigger (according to the given type) and the guarded code (according to the given type). These pieces of code are merged to constitute a single entity (i.e., condition + code triggered) and injected into the insertion point. Note that *trigger types* and *guarded code types* have been chosen from existing logic bombs found in previous research and reverse-engineering, as well as benign trigger-based behavior found in the same way.

3.1.3 Packaging. After infecting the app's code, one has to take care of any collateral effect. Therefore, AndroBomb adds any permission needed for the code injected [4]. For instance, if AndroBomb injects a piece of code that steals the current location of the device and sends it over HTTP, AndroBomb also injects the following permissions to the AndroidManifest.xml file:

- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.INTERNET

This is to ensure that no error occurs at execution time.

Besides, AndroBomb can inject pieces of code that might invoke native code [16]. However, to invoke a native function, the APK should contain the related .so file (i.e., native libraries). Thus, AndroBomb also injects the adequate .so files needed to invoke a native library. The authors of this paper have developed these libraries, their source code is available in the project's repository.

Eventually, the resulting APK is aligned [15], and signed [14]. Therefore, it can be installed on any device or emulator.

As AndroBomb only injects pieces of code that do not change the state of the initial app, the overall behavior remains unchanged. The infected app can be dynamically analyzed and monitored using emulators or statically analyzed to search for *potential* logic bombs. In addition, AndroBomb has been developed in a modular manner which allows the community to easily add new *trigger types* and new *guarded code types* to generate new apps for this dataset which is by design prone to evolve. We believe that this work will serve the community and advance the logic bomb detection research field.

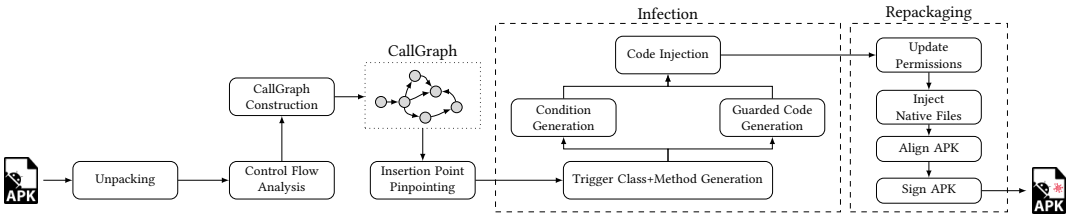

Figure 1: Overview of the AndroBomb approach to infect an Android app.

3.2 TriggerZoo

To generate TriggerZoo, we relied on AndroBomb which can, at the time of writing, handle the trigger and guarded code types described in Tables 1 and 2. The authors have carefully chosen the trigger and guarded code types from existing logic bomb and benign trigger-based behavior that have been reverse-engineered. In addition, the literature [13, 27, 29] describes several use-cases for which the authors extracted the trigger and guarded code types. Even if we cannot guarantee to cover all possible types of trigger and guarded code types, by relying on these state-of-the-art works, we are confident that TriggerZoo covers a large proportion of logic bomb types that could exist in the wild.

3.2.1 Dataset Construction. We randomly collected 2000 Google Play apps from the AndroZoo dataset as our initial set. Then, for each app, we applied AndroBomb with the trigger and guarded coded types randomly generated among those available in Tables 1 and 2. Each app is instrumented to receive one single trigger-based behavior. The resulting dataset, namely TriggerZoo, comprises 406 Android apps infected with trigger-based behavior. There are several reasons why AndroBomb was not able to generate an infected app for all of the 2000 apps:

- No insertion point was found in the app due to our strong constraint: we only consider methods that are in classes for which the fully qualified name starts with the app package name. (28.9%).
- The infected APK could not be repackaged due to some limitations of third-party software. For instance, (1) Soot could not handle multi dex APKs for apps using an Android API level lower than 22, or (2) The ManifestEditor library crashes due to buffer underflow (54.5%).
- AndroBomb crashes since for some apps the methods added during infection do not exist yet because they were added in a subsequent Android API level (16.6%).

However, these AndroBomb limitations are not critical since its final goal is not to be 100% operational for a specific task (e.g., malware detection [21] and GDPR compliance [10]) but to construct a valuable dataset for the community, which it was able to achieve.

3.2.2 Dataset Description. TriggerZoo is composed of several files referenced in the project's repository:

- original_apps: SHA256 hashes of the 406 original apps collected from AndroZoo.
- infected_apps: SHA256 hashes of the 406 infected apps.
- original_to_infected_correspondence: links the original and infected apps.
- triggerzoo_labeled_dataset: the labeled dataset.

Table 1: Trigger types handled by AndroBomb to generate TriggerZoo

Trigger Types	
Type	**Description**
time	at a specific time or date
location	at a specific location
sms	if a specific sms is received
network	if Wi-Fi available or specific http response received
build	if specific Build.MODEL/PRODUCT/FINGERPRINT are set
camera	if the device possesses cameras
addition	a dummy test with a simple addition
music	if some music is active
is_screen_on	if device in interactive state
is_screen_off	if device not in interactive state

Table 2: Guarded Code types handled by AndroBomb to generate TriggerZoo

○ = benign behavior, ⍟ = malicious behavior

Guarded Code Types		
Type	**Description**	
return	no behavior	○
sms_imei	send the device imei number by sms	⍟
stop_wifi	deactivate the device Wi-Fi connection	⍟
write_string	write a constant to a file in the device's memory	○
write_phone_number	write the phone number to a file in the device's memory	⍟
set_text	set a constant to be dispayed on the screen	○
sms_string	send a constant by sms	○
http_location	sends the current location to remote server using http	⍟
set_text_reflection	set a constant to be dispayed on the screen using reflection	○
exit	exits the app	⍟
native_log_string	log a constant using native code	○
native_log_model	log the Build.MODEL information using native code	⍟
native_write_phone_number	writes the phone number to a file using native code	⍟
native_phone_number_network	sends the phone number to a remote server using native code	⍟

TriggerZoo is only available to authenticated researchers to have an access to AndroZoo. Indeed, AndroZoo offers the authentication proxy for serving only the research community.

Format of the labeled dataset. The *triggerzoo_labeled_dataset* file in the project's repository describes in detail TriggerZoo with the fields available in Table 3. Each line of this file is composed of these 7 fields describing an app that has been infected.

Table 3: TriggerZoo fields

Field	
Type	**Description**
sha256_original_app	The sha256 hash of the original app
class_infected	The class infected
component_type	The component type of the class infected
method_infected	The method infected
trigger_type	The trigger type used to infect the app
guarded_code_type	The guarded code type used to infect the app
depths	Depths of the insertion point method in the app callgraph

Trigger and behavior types. TriggerZoo's repository shows two plots illustrating the number of apps infected with specific trigger and guarded code types. We can see that TriggerZoo covers a larger panel of trigger and guarded code types, and their combination. Note that TriggerZoo covers 137 unique combinations of trigger and guarded code types. Besides, based on the guarded code types, TriggerZoo comprises 240 apps with malicious trigger-based behavior, and 166 with benign trigger-based behavior.

Apps categories. TriggerZoo's project repository shows a third plot illustrating the different categories of TriggerZoo's apps. We were able to retrieve the category of 263 apps from the 406 in TriggerZoo. This is because the remaining 143 apps were not available on Google Play (i.e., they were removed from Google Play after being crawled by AndroZoo). We can see that TriggerZoo covers a large panel of categories, i.e., 34 unique categories.

Component types and Insertion Point Depth. We remind that TriggerZoo is built with AndroBomb that injects logic bombs or benign trigger-based behavior at random locations in the callgraph of the app developer code. With this process, among our 406 apps, the trigger-based behavior have been inserted in methods of the `Activity` components for 381 apps, and `Service` components for 25 apps. Figure 2 shows that for most apps, the callgraph depth of the insertion point is low.

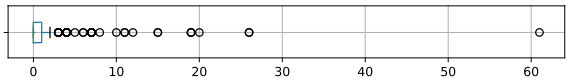

Figure 2: Infected methods' callgraph depths in TriggerZoo

3.2.3 Dataset Validation. To ensure that the apps are infected by AndroBomb, we randomly selected a statically significant sample of 118 apps from the 406 supposedly infected apps, with a confidence level of 99% and a confidence interval of ± 10%, to manually inspect them. We confirm that 100% of the apps manually analyzed are infected, i.e., they contain the newly added code, the AndroidManifest.xml file is updated with new permissions (if needed in function of the APIs injected), and native files are indeed present in the app (if required in function of the APIs injected). Also, to ensure that the apps are not faulty and can be dynamically analyzed, we tried to install and run them on emulators. We confirm that 100% of the 118 apps manually analyzed can be installed and run without any problem. Hence, the instrumentation and repackaging processes do not impact the installation or the runtime processes.

4 IMPORTANCE OF TRIGGERZOO

Several state-of-the-art approaches have been proposed to detect logic bombs. TriggerScope was proposed in [13] as an automated tool to detect logic bombs. It relies on static analysis techniques such as: symbolic execution, control flow analysis, predicate recovery, and control dependency. Recently, Samhi et al. [27] have released an open-source version of TriggerScope that they named TSOpen. Pan et al. [23] proposed HsoMiner, an approach relying on static analysis techniques such as: control flow analysis, backward dependency graph, and trigger analysis. In addition, the authors proposed a machine-learning approach to automatically sort *hidden sensitive operations* (HSO). Recently, Samhi et al. [29] proposed Difuzer, an approach to triage logic bombs among *suspicious hidden sensitive operations* (SHSO). Difuzer relies on instrumentation techniques, taint tracking to identify SHSO entry points and triggers an anomaly detection engine to detect abnormal triggers.

The common points for these approaches are: ❶ they do not assess their tool on existing benchmarks. Hence they cannot measure standard precision, recall, and f-1 score measures; ❷ they cannot compare their respective approach against each other tools. To cope with these limitations, TriggerZoo can be used to measure existing and future tools performances and will allow fair comparison.

In a first attempt to compare state of the art approaches, and to assess TriggerZoo's usefulness, we executed Difuzer [29] and TSOpen [27] on the 406 apps present in TriggerZoo to search for logic bombs. Results are available in Table 4. We note that in the original publication of Difuzer, precision and recall metrics were provided based on a-posteriori manual checking, given the lack of benchmark. Thanks to TriggerZoo, comparisons such as the one presented in Table 4 will be readily possible. Furthermore, note that the yielded results, with recall at 58%, suggest that TriggerZoo is "difficult" and will contribute to challenge future approaches. As a last remark, the a-posteriori comparison results presented in the Difuzer's paper [29] are confirmed on the TriggerZoo dataset. Indeed, as shown in Table 4, Difuzer clearly outperforms TriggerScope.

Table 4: Difuzer & TSOpen results on TriggerZoo

	# apps	Difuzer		TSOpen	
		# analyzed	# flagged	# analyzed	# flagged
Malicious triggers	240	230	134	215	32
Benign triggers	166	156	41	148	15
Precision		76.6%		68.1%	
Recall		58.3%		14.9%	
F_1 score		66.2%		24.4%	

5 CONCLUSION

In this paper, we presented two artifacts: ❶ TriggerZoo: a new evolving dataset of Android apps containing logic bombs and benign trigger-based behavior; ❷ AndroBomb: a new framework to infect Android apps with logic bombs and benign trigger-based behavior. TriggerZoo is meant to facilitate future logic bomb detectors assessment and comparison. We also provide results of two state-of-the-art approaches, i.e., Difuzer and TriggerScope, on TriggerZoo, and confirm previous literature results. We believe that the research community will rely on this dataset to propose new approaches to detect logic bombs, thus improving Android apps security and privacy. TriggerZoo is not frozen. It can evolve using AndroBomb to generate new samples with new logic bomb schemes. Both TriggerZoo and AndroBomb serve the community and support the logic bomb detection research direction.

6 ACKNOWLEDGMENT

This work was partly supported (1) by the Luxembourg National Research Fund (FNR), under projects Reprocess C21/IS/16344458 the AFR grant 14596679, (2) by the SPARTA project, which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 830892, (3) by the Luxembourg Ministry of Foreign and European Affairs through their Digital4Development (D4D) portfolio under project LuxWAyS, and (4) by the INTER Mobility project Sleepless@Seattle No 13999722.

REFERENCES

[1] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[2] Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2013. Instrumenting Android and Java Applications as Easy as abc, Vol. 8174. 364–381. https://doi.org/10.1007/978-3-642-40787-1_26

[3] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. *SIGPLAN Not.* 49, 6 (June 2014), 259–269. https://doi.org/10.1145/2666356.2594299

[4] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. PScout: Analyzing the Android Permission Specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 217–228. https://doi.org/10.1145/2382196.2382222

[5] Harel Berger, Chen Hajaj, and Amit Dvir. 2020. Evasion Is Not Enough: A Case Study of Android Malware. In *Cyber Security Cryptography and Machine Learning*, Shlomi Dolev, Vladimir Kolesnikov, Sachin Lodha, and Gera Weiss (Eds.). Springer International Publishing, Cham, 167–174.

[6] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. 2008. Automatically identifying trigger-based behavior in malware. In *Botnet Detection*. Springer, 65–88.

[7] M. Choudhary and B. Kishore. 2018. HAAMD: Hybrid Analysis for Android Malware Detection. In *2018 International Conference on Computer Communication and Informatics (ICCCI)*. 1–4. https://doi.org/10.1109/ICCCI.2018.8441295

[8] Nadia Daoudi, Jordan Samhi, Abdoul Kader Kabore, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2021. DexRay: A Simple, yet Effective Deep Learning Approach to Android Malware Detection Based on Image Representation of Bytecode. In *Deployable Machine Learning for Security Defense*, Gang Wang, Arridhana Ciptadi, and Ali Ahmadzadeh (Eds.). Springer International Publishing, Cham, 81–106.

[9] Shuaike Dong, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, XiaoFeng Wang, and Kehuan Zhang. 2018. Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild. In *Security and Privacy in Communication Networks*, Raheem Beyah, Bing Chang, Yingjiu Li, and Sencun Zhu (Eds.). Springer International Publishing, Cham, 172–192.

[10] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. 2020. An Empirical Evaluation of GDPR Compliance Violations in Android mHealth Apps. In *ISSRE*. 253–264. https://doi.org/10.1109/ISSRE5003.2020.00032

[11] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti. 2016. ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 1–5. https://doi.org/10.1109/NTMS.2016.7792435

[12] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna. 2016. TriggerScope: Towards Detecting Logic Bombs in Android Applications. In *2016 IEEE Symposium on Security and Privacy (SP)*. 377–396. https://doi.org/10.1109/SP.2016.30

[13] Yanick Fratantonio, Antonio Bianchi, William Robertson, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2016. Triggerscope: Towards detecting logic bombs in android applications. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 377–396.

[14] Google. 2021. *Sign your app.* https://developer.android.com/studio/publish/app-signing

[15] Google. 2021. *Zipalign.* https://developer.android.com/studio/command-line/zipalign

[16] Google. 2022. *Android NDK.* https://developer.android.com/ndk

[17] IDC. [n.d.]. *Smartphone Market Share, https://www.idc.com/promo/smartphone-market-share/os.* Accessed January 2022.

[18] Hyunjae Kang, Jae wook Jang, Aziz Mohaisen, and Huy Kang Kim. 2015. Detecting and Classifying Android Malware Using Static Analysis along with Creator Information. *International Journal of Distributed Sensor Networks* 11, 6 (2015), 479174. https://doi.org/10.1155/2015/479174 arXiv:https://doi.org/10.1155/2015/479174

[19] Ondrej Lhoták. 2003. Spark: A flexible points-to analysis framework for Java. (2003).

[20] Wen Li, Xiaoqin Fu, and Haipeng Cai. 2021. AndroCT: Ten Years of App Call Traces in Android. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 570–574. https://doi.org/10.1109/MSR52588.2021.00076

[21] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail Joon Ahn. 2017. Deep Android Malware Detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (Scottsdale, Arizona, USA) *(CODASPY '17)*. Association for Computing Machinery, New York, NY, USA, 301–308. https://doi.org/10.1145/3029806.3029823

[22] Sebastian Nielebock, Paul Blockhaus, Jacob Krüger, and Frank Ortmeier. 2021. AndroidCompass: A Dataset of Android Compatibility Checks in Code Repositories. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 535–539. https://doi.org/10.1109/MSR52588.2021.00069

[23] Xiaorui Pan, Xueqiang Wang, Yue Duan, XiaoFeng Wang, and Heng Yin. 2017. Dark Hazard: Learning-based, Large-Scale Discovery of Hidden Sensitive Operations in Android Apps.. In *NDSS*.

[24] N. Peiravian and X. Zhu. 2013. Machine Learning for Android Malware Detection Using Permission and API Calls. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. 300–305. https://doi.org/10.1109/ICTAI.2013.53

[25] Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Michalis Polychronakis, and Sotiris Ioannidis. 2014. Rage against the Virtual Machine: Hindering Dynamic Analysis of Android Malware. In *Proceedings of the Seventh European Workshop on System Security* (Amsterdam, The Netherlands) *(EuroSec '14)*. Association for Computing Machinery, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/2592791.2592796

[26] J. Sahs and L. Khan. 2012. A Machine Learning Approach to Android Malware Detection. In *2012 European Intelligence and Security Informatics Conference*. 141–147. https://doi.org/10.1109/EISIC.2012.34

[27] J. Samhi and A. Bartel. 2021. On The (In)Effectiveness of Static Logic Bomb Detector for Android Apps. *IEEE Transactions on Dependable and Secure Computing* 01 (aug 2021), 1–1. https://doi.org/10.1109/TDSC.2021.3108057

[28] J. Samhi, J. Gao, N. Daoudi, P. Graux, H. Hoyez, X. Sun, K. Allix, T. F. Bissyande, and J. Klein. 2022. JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. Association for Computing Machinery. https://doi.org/10.1145/3510003.3512766

[29] J. Samhi, L. Li, T. F. Bissyande, and J. Klein. 2022. Difuzer: Uncovering Suspicious Hidden Sensitive Operations in Android Apps. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. Association for Computing Machinery. https://doi.org/10.1145/3510003.3510135

[30] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 2010. Soot: A Java Bytecode Optimization Framework. In *CASCON First Decade High Impact Papers* (Toronto, Ontario, Canada) *(CASCON '10)*. IBM Corp., USA, 214–224. https://doi.org/10.1145/1925805.1925818

[31] Raja Vallee-Rai and Laurie J Hendren. 1998. Jimple: Simplifying Java bytecode for analyses and transformations. (1998).

[32] Victor Van Der Veen, Herbert Bos, and Christian Rossow. 2013. Dynamic analysis of android malware. *Internet & Web Technology Master thesis, VU University Amsterdam* (2013).

[33] Tyler Wendland, Jingyang Sun, Junayed Mahmud, S. M. Hasan Mansur, Steven Huang, Kevin Moran, Julia Rubin, and Mattia Fazzini. 2021. Andror2: A Dataset of Manually-Reproduced Bug Reports for Android apps. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 600–604. https://doi.org/10.1109/MSR52588.2021.00082

[34] Lifan Xu, Dongping Zhang, Nuwan Jayasena, and John Cavazos. 2018. HADM: Hybrid Analysis for Detection of Malware. In *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, Yaxin Bi, Supriya Kapoor, and Rahul Bhatia (Eds.). Springer International Publishing, Cham, 702–724.

[35] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. 2012. Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*.