DroidAuditor: Forensic Analysis of Application-Layer Privilege Escalation Attacks on Android (Short Paper)

Stephan Heuser[1], Marco Negro[2], Praveen Kumar Pendyala[1], and Ahmad-Reza Sadeghi[1]

[1] Intel CRI-SC, TU Darmstadt, Darmstadt, Germany
{stephan.heuser,praveen.pendyala,ahmad.sadeghi}@trust.cased.de
[2] University of Padua, Padua, Italy
mnegro@studenti.math.unipd.it

Abstract. Smart mobile devices process and store a vast amount of security- and privacy sensitive data. To protect this data from malicious applications mobile operating systems, such as Android, adopt fine-grained access control architectures. However, related work has shown that these access control architectures are susceptible to application-layer privilege escalation attacks. Both automated static and dynamic program analysis promise to proactively detect such attacks. Though while state-of-the-art static analysis frameworks cannot adequately address native and highly obfuscated code, dynamic analysis is vulnerable to malicious applications using logic bombs to avoid *early* detection.
In contrast, the *long-term* observation of application behavior could help users and security analysts better understand malicious apps. In this paper we present the design and implementation of DroidAuditor, which observes application behavior on real Android devices and generates a graph-based representation. It visualizes this *behavior graph*, which enables users to develop an intuitive understanding of application internals. Our solution further allows security analysts to query the behavior graph for malicious patterns. We present the design of the DroidAuditor framework and instantiate it using the Android Security Modules (ASM) access control architecture. We evaluate its capability to detect application-layer privilege escalation attacks, such as confused deputy and collusion attacks. In addition, we demonstrate how our architecture can be used to analyze malicious spyware applications.

1 Introduction

Smart mobile devices, such as smartphones and tablets, host a vast number of third-party applications of varying quality and trustworthiness. These applications access, store and process security- and privacy-sensitive data, ranging from personal contacts, location information to high-profile enterprise assets, which makes these devices valuable targets for attacks. Unsurprisingly, the number of newly discovered malware families targeting these devices is rising [11].

To mitigate such attacks operating systems for smart mobile devices use fine-grained access control architectures. For example, Android uses *permissions* to restrict access to privacy- and security-sensitive data. However, related work has shown that Android's access control model is susceptible to application-layer privilege escalation attacks, ranging from insufficiently protected system settings [7] to accessing the Internet [10] or sending SMS [4] without holding corresponding permissions.

The systematic detection, analysis and mitigation of such attacks is an active area of research today: On one hand, system-centric access control architectures [8,3] attempt to mitigate these attacks using carefully designed use-case specific policies. On the other hand, both static and dynamic program analysis promise to proactively detect such attacks, but either do not adequately address native and highly obfuscated code, or are susceptible to malware using logic bombs [12] to avoid early detection.

This inability to proactively and reliably detect application-layer privilege escalation attacks mandates tools for long-term observation and analysis of application behavior. In this paper, we present DroidAuditor, a forensic application behavior analysis toolkit targeting application-layer privilege escalation attacks. DroidAuditor adopts the *Android Security Modules (ASM)* [8] access control architecture to observe application behavior at all layers of the Android operating system. Our solution organizes these observations in a *behavior graph* and generates an interactive visualization. It further allows forensic analysts to query this graph for suspicious patterns using a graph query language.

Our main contributions are as follows:

- We present the design and implementation of DroidAuditor, a solution for application behavior analysis using interactive behavior graphs.
- We evaluate DroidAuditor's capabilities by analyzing application-layer privilege escalation attacks as well as malicious spyware apps.
- We show that sophisticated access control frameworks, such as the ASM framework, are a valid basis for application behavior analysis.

DroidAuditor differs from related work on application behavior analysis in two major aspects: First, to the best of our knowledge, it is the first solution that adopts a modular access control framework for application behavior analysis. Second, the behavior graph generated by DroidAuditor serves as an important building block for further research on application behavior analysis, which we describe in more detail in our technical report [9] due to space constraints.

2 Background

Android is a Linux-based operating system for smart mobile devices. It hosts system and third-party applications, which consist of the following main components: *Activities* (GUI elements), *Services* (background tasks without any user interface), *ContentProviders* (data stores with SQL semantics) and *Broadcast-Receivers* (mailboxes for messages *(Intents)* from other components). Applications are executed in isolated least-privilege sandboxes, and they share data and

functionality via inter-process communication (IPC). Standard operating system components located on the middleware and application layer provide access to security- and privacy-sensitive resources, such as location information or contacts data. To control access to these components Android uses *permissions* granted by the user to applications.

However, this permission-based access control model is prone to application-layer privilege escalation attacks, such as confused deputy and collusion attacks [4]. In a confused deputy attack, an adversary abuses non-malicious but vulnerable software components via IPC to perform privileged security- and privacy sensitive operations: On Android, for example, only apps holding the INTERNET permission are able to open network sockets. However, any app can contact arbitrary web servers by deputizing the web browser via IPC [10]. In contrast, in a collusion attack multiple seemingly benign applications operate in concert to share their permissions towards a common goal. Colluding applications coordinate their attack via overt (e.g., Android's Binder IPC mechanism) or covert communication channels, such as shared system settings or files.

3 Adversary Model and Objectives

The main goals of DroidAuditor are the systematic monitoring of application behavior and the detection as well as forensic analysis of potential application-layer privilege escalation attacks. The adversary is capable of deploying one or more malicious applications on a target device, for example via social engineering or by gaining temporary physical access to the device. We place no restrictions on the type of code the adversary executes and thus allow managed bytecode as well as native and self-modifying code. Since DroidAuditor is a system-centric application behavior analysis framework, we have to assume that malicious applications do not gain administrative device management privileges, and that the trusted computing base of the Android device (bootloader, operating system kernel, middleware layer and system applications) remains intact. Otherwise, no correct app behavior analysis can be guaranteed. Finally, we assume that all DroidAuditor software components are trusted.

4 DroidAuditor

The high-level idea of DroidAuditor is to observe application behavior using the system-centric Android Security Modules (ASM) access control framework [8]. DroidAuditor stores these observations in a *behavior graph*, where vertices represent applications and resources, and edges represent data- or control flows.

Consider the following confused deputy attack: A malicious Android application holds the READ_CONTACTS permission and abuses the web browser to exfiltrate sensitive contacts information to a remote server without holding the INTERNET permission. Figure 1 shows this attack as a behavior graph: Upon start (Step 1) the malicious app reads sensitive data from the contacts database (Step 2). It then starts the web browser via an Intent (Step 3) and instructs it to

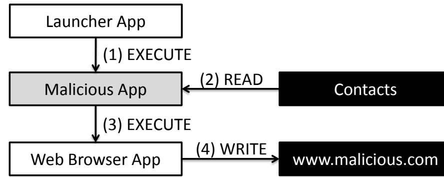


Fig. 1. Example confused deputy attack, where a malicious app deputizes the web browser to exfiltrate sensitive contacts information.

exfiltrate contacts data on its behalf. The web browser opens a network socket to a remote server and uploads the collected contacts information (Step 4).

DroidAuditor generates such behavior graphs using three main components (see Fig. 2). On the mobile device, the ASM for DroidAuditor is notified by the ASM framework whenever Android applications access security- or privacy-critical resources (Steps 1 - 4). The DroidAuditor ASM forwards these events to the DroidAuditor Database via an authenticated and encrypted channel (Step B), where they are stored in the behavior graph. Finally, security analysts can interact with the behavior graph using the DroidAuditor Client (Step C).

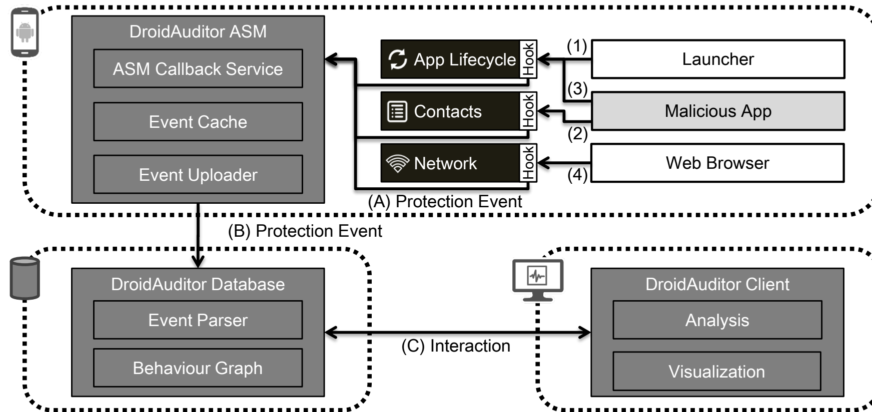


Fig. 2. DroidAuditor High-Level Architecture.

4.1 DroidAuditor ASM

The Android Security Modules framework places hooks in all security- and privacy-sensitive kernel- and middleware-layer operating-system components. These hooks generate aforementioned protection events, which the ASM framework forwards to all installed security modules. Each module can then decide whether to allow or deny the corresponding operations. Our DroidAuditor Android Security Module however does not enforce any access control rules, but

collects protection events to obtain a *global view* of all privacy- and security-critical operations performed by *all* applications. Accordingly, it allows every access control query and periodically uploads protection events to the DroidAuditor Database, where they are stored for analysis.

4.2 DroidAuditor Database

The DroidAuditor Database stores security- and privacy-sensitive protection events for offline analysis. It parses events uploaded by the DroidAuditor ASM and generates the *behavior graph* $G = \langle V, E \rangle$: The vertex set $V = A \cup R$ is composed of two subsets A and R, which represent applications A and resources R. For each application vertex $a \in A$ the DroidAuditor Database stores an identifier as well as additional metadata, for instance the permissions the application holds. Each resource vertex $r \in R$ models a security- or privacy-sensitive operating system resource. Important examples are Android's ContactsProvider, LocationManagerService or CameraService, as well as files and network sockets.

Every edge $e \in E$ is directional and describes a data- or control flow between two vertices $v_i, v_j \in V$. Each edge contains descriptive metadata, such as the time and date a flow was observed. Edges are grouped into categories, which model Android component interaction as well as file system and network operations (CREATE, READ, WRITE, UPDATE, DELETE, EXECUTE).

4.3 DroidAuditor Client

The DroidAuditor Client is a desktop application that interacts in real-time with the DroidAuditor Database. Its purpose is twofold:

First, the DroidAuditor Client generates an interactive visual representation of the behavior graph, which allows forensic analysts to intuitively understand an application's runtime behavior. Analysts can inspect the type and metadata for each vertex and edge as well as observe changes in the graph over time.

Second, the DroidAuditor Client allows analysts to query the behavior graph for specific patterns using the Cypher query language.[3] Listing 1 demonstrates how to query the behavior graph for signs of the previously described confused deputy attack, where a malicious app deputizes the web browser to exfiltrate sensitive contacts information. The depicted query identifies subgraphs starting with apps reading the Contacts resource (Lines 1-2). We only consider applications which then execute the Android web browser (Line 3) and do not hold the INTERNET permission (Line 5). Finally, this query expects the web browser to write data to a network socket (Line 4). Matching subgraphs are highlighted using the visualization plugin.

```
1  MATCH confuseddeputy = (contacts:Resource {type:'contacts'})
2  - [event1:READ] -> (app1:App {systemApp:false})
3  - [event2:EXECUTE] -> (app2:App {package:'com.android.browser'})
4  - [event3:WRITE] -> (socket:Resource {type:'socket'})
5  WHERE NOT 'internet' IN app1.permissions
```

Listing 1. Cypher query to detect the confused deputy attack.

[3] http://neo4j.com/developer/cypher-query-language/

5 Evaluation

DroidAuditor inherits the performance and energy consumption overhead of the underlying Android Security Modules framework, which has been scrutinized in [8]. In this work we primarily focused on DroidAuditor's effectiveness to analyze malicious application behavior. To this end, we implemented the DroidAuditor architecture using the Java programing language. We place the DroidAuditor ASM on a Nexus 4 smartphone running the ASM architecture version 4.4.4 r2. The Neo4J-based DroidAuditor graph database[4] and the DroidAuditor client communicate opportunistically when Wifi connectivity is available using the Kryonet[5] network communication stack. Real-world DroidAuditor deployments however should consider more firewall-friendly communication channels, such as HTTPS. The behavior graph is visualized using the GraphStream[6] library. We then deployed applications which implement confused deputy and collusion attacks as well as malicious spyware applications on the device and analyzed their behavior.

5.1 Application-Layer Privilege Escalation Attacks

Confused Deputy Attacks. We implemented the confused deputy attack described in Section 4, where a malicious app not holding the INTERNET permission deputizes the web browser to exfiltrate sensitive contacts data to a remote server. We then verified that the query described previously in Listing 1 indeed correctly identifies this confused deputy attack.

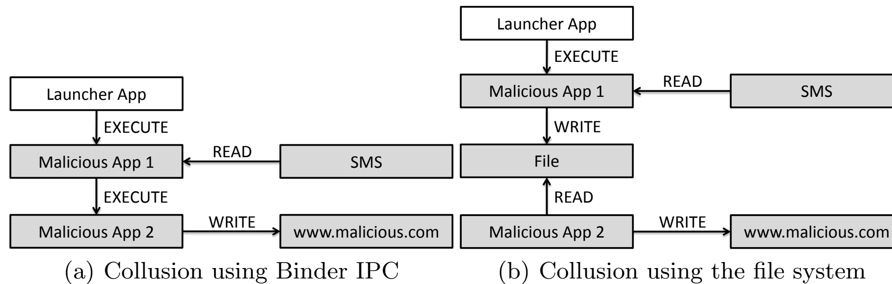
Collusion Attacks. We further implemented two variants of a collusion attack, where two malicious apps coordinate their actions towards a common goal, which in our example is to exfiltrate the SMS database over the Internet. The first malicious application only possesses the READ_SMS permission, and the second application only the INTERNET permission.

Collusion via Binder IPC. In a simple collusion attack two malicious apps communicate using overt channels, such as Android's Binder IPC mechanism. Listing 2 shows a Cypher query which targets this behavior. We query the behavior graph for non-system apps (Line 3), which do not hold the INTERNET permission (Line 5) and read from the SMS database (Line 1 and 2). We search for paths leading to another non-system app, which writes data to a remote server (Line 3 and 4). The corresponding subgraph is highlighted in in Fig. 3(a).

```
1  MATCH collusion1 = (sms:Resource {type:'sms'})
2  - [event1:READ] -> (app1:App {systemApp:false})
3  - [event2:EXECUTE] -> (app2:App {systemApp:false})
4  - [event3:WRITE] -> (socket:Resource {type:'socket'})
5  WHERE NOT 'internet' IN app1.permissions
```

Listing 2. Cypher Query to detect the collusion attack depicted in Fig. 3(a).

[4] http://www.neo4j.com
[5] https://github.com/EsotericSoftware/kryonet
[6] http://graphstream-project.org/



(a) Collusion using Binder IPC (b) Collusion using the file system

Fig. 3. Example collusion attacks where two malicious apps coordinate their behavior to exfiltrate the SMS database.

Collusion via File-based Communication. In this obfuscated collusion attack two applications share a file on the file system to exchange sensitive data. Note that no direct inter-process communication between both apps occurs in this scenario. Starting from the previous query in Listing 2, we add a file resource node to the query which matches files written to and read from the two colluding applications. Listing 3 shows the resulting query, and Fig. 3(b) depicts a visualization of the discovered subgraph.

```
1  MATCH collusion2 = (sms:Resource {type:'sms'})
2  - [event1:READ] -> (app1:App {systemApp:false})
3  - [event2:WRITE] -> (file:Resource {type:'file'})
4  - [event3:READ] -> (app2:App {systemApp:false})
5  - [event4:WRITE] -> (socket:Resource {type:'socket'})
6  WHERE NOT 'internet' IN app1.permissions
```

Listing 3. Cypher Query to detect the collusion attack depicted in Fig. 3(b).

DroidAuditor can similarly be used to detect signs of collusion attacks via other operating-system resources, such as domain or network sockets, Content-Providers or Services. However, it should be noted that DroidAuditor is limited by the granularity of the underlying ASM framework, which is unable to observe app collusion via hardware side channels, such as the CPU cache.

5.2 Identifying spyware applications

To demonstrate that DroidAuditor is a valid basis for generic application behavior analysis beyond application-layer privilege escalation attacks we installed two popular spyware applications, namely "TheTruthSpy"[7] and "LetMeSpy"[8], on a DroidAuditor device. By analyzing the behavior graph we found that these apps silently access privacy-sensitive resources, such as the CallLog and SMS/MMS ContentProviders as well as location data, and upload this data to a remote server. We further noticed that these apps only have very limited user interfaces (Activities), which are exclusively used for initial configuration.

[7] http://thetruthspy.com/
[8] http://www.letmespy.com/

To detect such behavior, we first labeled all privacy-sensitive resources in the behavior graph. In Listing 4, we query the graph for non-system apps accessing these resources (Lines 1 and 2) and writing data to a remote server (Line 3). The WHERE clause (Line 4) limits our query to apps which silently access privacy-sensitive resources. A visualization of the behavior graph and corresponding DroidAuditor client screenshots can be found in our technical report [9].

```
1  MATCH spyware1 = (res:Resource {privacySensitive:true})
2  - [event1:READ] -> (app:App {systemApp:false})
3  - [event2:WRITE] -> (socket:Resource {type:'socket', addr:'69.64.81.49'})
4  WHERE NOT event1.foregroundApp = app.package
```

Listing 4. Cypher query to detect the behavior of the TheTruthSpy app.

6 Related work

Our DroidAuditor architecture shares functionality with dynamic program analysis solutions [2,6,15,16,13,14], which observe app behavior in instrumented Android environments. Common techniques adopted by these frameworks are system call tracing, dynamic taint analysis and virtual machine introspection. While taint analysis can provide fine-grained tracing of privacy sensitive data while it is processed on a device, current designs do not adequately handle native code. Further, all approaches adopting dynamic program analysis are prone to logic bombs, where apps delay their malicious behavior to avoid detection [12]. DroidAuditor avoids these limitations by observing application behavior on real Android devices without imposing restrictions on the analyzed applications.

Some work has proposed the use of IPC call chain verification [7,5,1] to mitigate application-layer privilege escalation attacks. Other work more related to DroidAuditor [3] records app interaction in a graph structure and enforces access control policies targeting confused deputy and collusion attacks. Since DroidAuditor is based on the system-centric ASM access control framework, it is conceivable to implement this functionality by generating the behavior graph on the mobile device itself and applying a corresponding access control policy.

7 Conclusion

In this paper, we presented DroidAuditor, a toolkit for forensic long-term application behavior analysis. DroidAuditor uses the system-centric ASM mandatory access control framework to generate a graph-based model of application behavior. The preliminary evaluation of our proof-of-concept implementation demonstrates that modular access control frameworks are a valid building block for application behavior analysis and motivate us to extend our work in multiple directions: First, we plan to conduct a usability study to better understand how users interact with DroidAuditor. Second, we are implementing a policy enforcement architecture based on DroidAuditor's behavior graph by storing and evaluating the behavior graph on the mobile device. Finally, we aim to integrate dynamic taint analysis into DroidAuditor, which would allow us to support more precise data flow analysis for applications which do not contain native code.

References

1. M. Backes, S. Bugiel, and S. Gerling. Scippa: System-Centric IPC Provenance on Android. In *30th Annual Computer Security Applications Conference*, pages 36–45. ACM, 2014.
2. T. Blsing, L. Batyuk, A.-D. Schmidt, S. Camtepe, and S. Albayrak. An Android Application Sandbox system for suspicious software detection. In *5th International Conference on Malicious and Unwanted Software*, pages 55–62, 2010.
3. S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry. Towards Taming Privilege-Escalation Attacks on Android. In *19th Annual Network & Distributed System Security Symposium*, 2012.
4. L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege Escalation Attacks on Android. In *13th International Conference on Information Security*, pages 346–360. Springer, 2011.
5. M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: Lightweight Provenance for Smart Phone Operating Systems. In *20th USENIX Security Symposium*. USENIX, 2011.
6. W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information Flow Tracking System for Real-time Privacy Monitoring on Smartphones. *Commun. ACM*, 57(3):99–106, 2014.
7. A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission re-delegation: Attacks and defenses. In *20th USENIX Security Symposium*. USENIX, 2011.
8. S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi. ASM: A programmable interface for extending android security. In *23rd USENIX Security Symposium*. USENIX, 2014.
9. S. Heuser, M. Negro, P. K. Pendyala, and A.-R. Sadeghi. DroidAuditor: Forensic Analysis of Application-Layer Privilege Escalation Attacks on Android. Technical report, TU Darmstadt, 2016.
10. A. Lineberry, D. L. Richardson, and T. Wyatt. These Aren't the Permissions You're Looking For. DefCon 18, 2010.
11. McAfee. Threats Report May 2015. `http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2015.pdf`, May 2015.
12. S. Rasthofer, I. Asrar, S. Huber, and E. Bodden. How Current Android Malware Seeks to Evade Automated Code Analysis. In *9th International Conference on Information Security Theory and Practice*, pages 187–202. Springer, 2015.
13. V. Rastogi, Y. Chen, and W. Enck. AppsPlayground: Automatic Security Analysis of Smartphone Applications. In *Third ACM Conference on Data and Application Security and Privacy*, pages 209–220. ACM, 2013.
14. M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann. Mobile-sandbox: Having a Deeper Look into Android Applications. In *28th Annual ACM Symposium on Applied Computing*, pages 1808–1815. ACM, 2013.
15. K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro. CopperDroid: Automatic Reconstruction of Android Malware Behaviors. In *22nd Annual Network & Distributed System Security Symposium*, 2015.
16. L. K. Yan and H. Yin. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. In *21st USENIX Security Symposium*. USENIX, 2012.