DroidAlarm: An All-sided Static Analysis Tool for Android Privilege-escalation Malware

Yibing Zhongyang, Zhi Xin, Bing Mao and Li Xie
State Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University
Sophie.xuer@gmail.com, {zxin,maobing,lixie}@nju.edu.cn

ABSTRACT

Since smartphones have stored diverse sensitive privacy information, including credit card and so on, a great deal of malware are desired to tamper them. As one of the most prevalent platforms, Android contains sensitive resources that can only be accessed via corresponding APIs, and the APIs can be invoked only when user has authorized permissions in the Android permission model. However, a novel threat called privilege escalation attack may bypass this watchdog. It's presented as that an application with less permissions can access sensitive resources through public interfaces of a more privileged application, which is especially useful for malware to hide sensitive functions by dispersing them into multiple programs. We explore privilege-escalation malware evolution techniques on samples from Android Malware Genome Project. And they have showed great effectiveness against a set of powerful antivirus tools provided by VirusTotal. The detection ratios present different and distinguished reduction, compared to an average 61% detection ratio before transformation. In order to conquer this threat model, we have developed a tool called DroidAlarm to conduct a full-spectrum analysis for identifying potential capability leaks and present concrete capability leak paths by static analysis on Android applications. And we can still alarm all these cases by exposing capability leak paths in them.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General—*Protection mechanisms*; D.4.6 [**Operating Systems**]: Security and protection—*Invasive software*

General Terms

Security

Keywords

Android; Capability Leaks; Privilege Escalation Attack; Malware Transformation; Static Analysis

1. INTRODUCTION

It is generally agreed that users are shifting their primary Internet accessing terminals from PCs to mobile devices. And they can download plenty of applications from diverse markets, such as Google Play, an official Android Market that contains over 600,000 applications. However, these applications are also carrying a huge number of unprecedented threats. According to F-Secure's Mobile Threat Report Q3 2012 [10], mobile malware cases on Android platform almost increased 16 times in the first three quarters of 2012, and the total number in the third quarter of 2012 is as high as 51,447. Many malware have shown great interests in user's privacy information for profit, including SMS messages [17], credit card information [9], audio records [18] and so on. In order to restrict potential privacy information abuse, Android raises a permission-based security model [4]. Every application must explicitly propose a list of required permissions to users before the installation procedure, which may expose its malicious intention. However, a new emerging threat model called *privilege escalation attack* [7] can bypass this permission authorization mechanism. It aims at the event that a authorization list contains suspicious sensitive permissions, which may lead to malicious usage. By utilizing the threat model, sensitive permissions and relevant malicious usage will be dispersed into different applications.

According to existing references [19, 22, 3, 13], antivirus scanners may first calculate hash values of the tested file to decide whether it is a known malware. If it is an unknown sample, scanners then filter sensitive permissions and then use CFG analysis, heuristics-based methods and so on to verify whether it is malicious. So in the above threat model, it is hard for scanners to identify potential malicious intentions with multiple individual permission lists and applications.

There are some analysis and detection methods against *privilege escalation attack*. Dynamic methods focus on call-graph analysis [11], the call-chain of IPC [8] or a customized monitoring framework [5] to address part or all of the attack. However, these methods have problems with path coverage and input effectiveness. There are also some static techniques to analyze the attack in the stock image [12] or applications [6, 16]. However, they only care about public interfaces on middleware layer and do not cover other forms of public interfaces on kernel layer, including file system and network sockets, which may miss some capability leak cases. For example, NickyBot [14] puts its audio files on SDcard, which may be obtained by its potential cooperator. And Chan et.al. [6] use decompiled applications to analyze, which suffers unsatisfactory decompiled results and potential obfuscation interruption.

In this paper, we show the seriousness of privilege-escalation evolution on Android malware and also have developed an automatic static analyzer to alarm these cases. We build three privilege-escalation transformation models to explain the former issue based on different functionalities. Different functionalities are mapped to implement in different cooperation ways. The communication between applications makes full use of different forms of public interfaces.

To alarm the attack, we have developed DroidAlarm to perform the automatic static analysis process. Relevant files (i.e. the configuration file and the executable file) are parsed for sensitive permissions and public interfaces. If an application has both sensitive permissions and public interfaces, relationships between sensitive permissions and public interfaces will be built to reveal its capability leaks. API calls corresponding with these permissions will be located. Then, based on the cooperation ways concerning these APIs, existing public interfaces will be checked and the data flow will be followed to get concrete capability leak paths, which will raise users' vigilance on potential *privilege escalation attacks*. Since DroidAlarm is off-line, there is a limited overhead.

We analyze 49 well-known malware from Android Malware Genome Project [20] with DroidAlarm. These 49 covers each malware family infected by the similar payload, so they already involve all malware patterns in this project. They have not widely adopted privilege escalation to advance themselves yet. Only 6% have unintentional capability leaks. However, we explore the construction models to transform them into privilege-escalation malware based on different functionalities. After transformation, the detection ratios have significantly reduced, compared to an average 61% before transformation. So these privilege-escalation malware bypass most antivirus tools on VirusTotal [1]. However, we can still alarm all these cases.

In particular, our contributions are as follows:

- We explore three styles of privilege-escalation malware transformation techniques based on their different functionalities.
- We build the first bytecode-based static capability leak analyzer, DroidAlarm, on all kinds of communication channels, including ICC, file system and network sockets.

The rest of this paper is arranged as follows: Section 2 gives a whole background of Android and *privilege escalation attack*; Section 3 presents the design and implementation of our malware transformation techniques and DroidAlarm in detail; Section 4 provides the evaluation results; Section 5 summarizes related work and the distinguishing features of our work; Section 6 illustrates the limitations and future work; Section 7 concludes the paper.

2. BACKGROUND

Before presenting our work, the relevant background of Android and the threat model will be briefly highlighted.

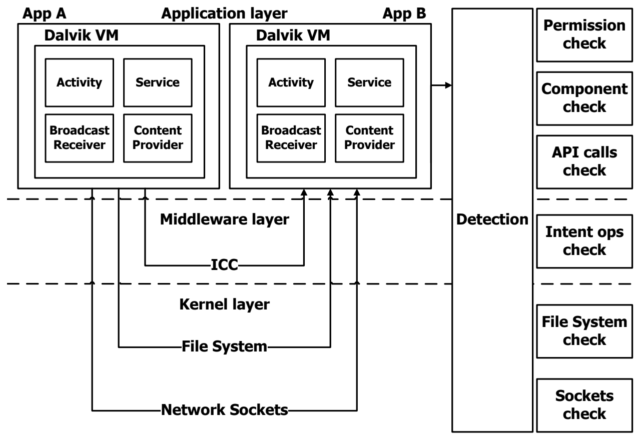


Figure 1: System Overview

2.1 Android

Fig. 1 shows Android internals from the viewpoints of possible communication channels and our check spots. Android is a free, open source, Linux-based OS designed for smartphones. Possible communication channels are built at multiple layers, including middleware layer and kernel layer. Communication channels at kernel layer contain Linux file system and network sockets. At middleware layer, the communication channel is usuallly built at the granularity of components, called *inter component communication* (ICC). An Android application consists of four types of components, which are Activities, Services, Broadcast Receivers and Content Providers. The ties between different components are Intents, which contain data and actions to transmit. At application layer, each application has a unique uid and runs on an independent Dalvik Virtual Machine (DVM). In Android, APIs related with sensitive resources, like sending SMS messages, are protected by different permissions.

2.2 Privilege Escalation Attack

Privilege escalation attack needs to be distinguished from *privilege escalation* or root exploit [21]. For root exploit, malware intentionally utilize Android platform-level vulnerabilities to get root privileges. However, *privilege escalation attack* mostly focuses on borrowing others' permissions to reach its malicious goal instead of actually getting them.

We divide privilege escalation attacks into code cooperation and data cooperation. In code cooperation, app A with less permissions can invoke components of a more privileged app B, like Activity launching function startActivity, to conduct sensitive behaviors. And in data cooperation app A can get sensitive data from app B through various communication channels, such as intent.getExtras().

There may exist a dispute that the attacker needs to install at least two malicious apps on a single user's device to complete the procedure. However, it can be easily handled in two aspects. On one hand, an attacker can explore capability leaks in other apps [12, 6, 16] and utilize them to reach their malicious goals. On the other hand, if one attack app has been installed, it can use the function of app recommendation to wheedle the user to install other ones. Then they can cooperate to achieve their malicious intention.

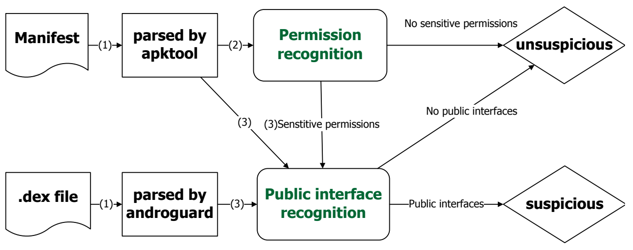


Figure 2: overview of suspicious analysis

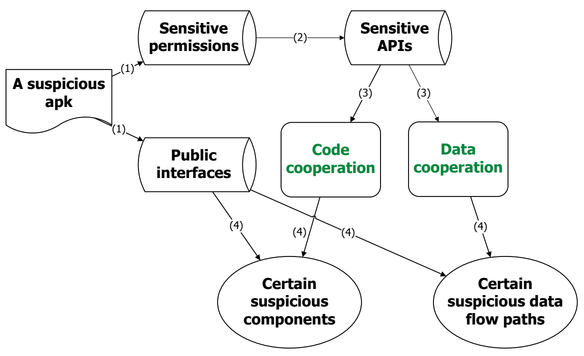


Figure 3: overview of leak path analysis

3. DESIGN

After presenting the threat model, the design of DroidAlarm and our malware transformation techniques will be introduced.

3.1 DroidAlarm

Fundamentally, in order to reduce privilege escalation attack, DroidAlarm aims at alarming capability leaks of an application that potentially give rise to the attack. The whole solution is divided into two modules and four submodules.

3.1.1 permission recognition

Pemission recognition is a submodule in Fig. 2. In our threat model, sensitive resources is the purpose of capability leaks. Since sensitive permission is related with accessing sensitive resource, it is an important target in this module.

Android categorizes the protection levels of permissions into four levels: normal, dangerous, signature and signatureOrSystem. The threat level increases one by one. In our solution, as we do not think normal permissions can be actual risks, we define all other permissions as sensitive permissions. The threat level of permissions in a third-party application is generally no higher than dangerous, and malware are no exceptional. With permissions extracted from AndroidManifest by apktool [2], we check every item with permission protection levels file and maintain those whose protection levels are not normal. If sensitive permissions are found, analysis procedure jumps to the next step.

3.1.2 public interface recognition

Based on our threat model, public interfaces are vital targets in the following submodule (Fig. 2). They have various forms in AndroidManifest and the dex file. In AndroidManifest they are mostly related with components. Firstly, Activities, Services and BroadcastReceivers containing a user-defined intent-filter can be triggered by other applications. So this kind of intent-filter is a striking feature. Secondly, Content Providers have a “true” exported value unless they are designed for Android 4.2 or higher versions. Finally, as multiple applications that have a same uid can share their resources to make privilege escalation attack, another conspicuous attribute is sharedUserId. However, more public interfaces at three communication channel levels are extracted from the dex file by androguard [3]. For ICC, Intent is the key. For file system, all the four types of file, SharedPreference, traditional Java file, SQLite and SDcard file, can be considered as public interfaces. Ultimately, all the network socket classes also need to be recognized.

Combined with the results of permission recognition, if a malware has such interfaces, it is suspicious. However, suspicious analysis module is not sufficient to settle this issue. When a malware has public interfaces that do not reveal sensitive resources protected by corresponding permissions, it is a false positive case. To reduce the rate of false positive, we need to filter the preliminary result set.

3.1.3 code cooperation

Based on suspicious analysis module, we construct a concrete path from a sensitive permission to a public interface. We locate sensitive APIs corresponding to the above permissions, and divide them on the cooperation way.

Code cooperation that focuses on APIs with some sensitive behaviors is a submodule in Fig. 3. Regarding public interfaces obtained, we firstly check whether components containing the above APIs have such an interface or not. If so, a leak path will be built. Besides, we also need to check the above components to see if other components that invoke one of them have public interfaces. Then we repeat the process iteratively. Eventually, we may build a leak path, a sensitive API with relevant permissions releasing resources during a series of component invoking, where the component at the highest invoking level contains a public interface.

3.1.4 data cooperation

Data cooperation is a more complex submodule in Fig. 3, where we concentrate on APIs concerning sensitive data and APIs that provide sensitive behaviors to generate sensitive data. A simple algorithm is presented as Algorithm 1.

As variables are stored in registers on Dalvik bytecode, we extract every method that initially calls one of the above APIs, and gets a sensitive register set with related variables. Then we analyze each instruction as a sequence of intraprocedural CFG of these methods. Based on different types of instructions, we make four decisions. Firstly, if any register in the set is changed, we remove it. Secondly, if data of any sensitive register is passed to another one that is not in the set, we record the influenced register in the set. Thirdly, if there is a public interface, which exposes all or part of the sensitive register set, we can then build a certain capability leak path. Fourthly, if a function call involves any sensitive registers, we track it iteratively. In the end, if the return value of the method is in one register of the set, we refer to interprocedural CFG and check its invokers similarly. Consequently, there will be a conclusion that whether a malware has a capability leak path where sensitive data obtained from an API with sensitive permissions are transmitted to a public interface during a series of function calls.

Algorithm 1 algorithm of data flow tracking

```
1:  [method, SOURCE] = extract sensitive API()
2:  for every instruction I as a sequence of intraprocedural
    CFG of method do
3:      if i changes va in SOURCE then
4:          SOURCE.remove(va)
5:      end if
6:      if i assigns vb in SOURCE to vc then
7:          SOURCE.merge(vc)
8:      end if
9:      if i provides a public interface and its parameters in
        SOURCE then
10:         build a certain suspicious data flow path
11:     end if
12:     if i is a function call and its parameters in SOURCE
        then
13:         track the function and its parameters in SOURCE
14:     end if
15: end for
16: if method has return value then
17:     expand CFG to find its invokers and check them with
        the return value
18: end if
```

3.2 Malware transformation techniques

Since existing malware have not widely adopted privilege-escalation construction techniques as discussed in Section 4, we are inspired to explore the techniques based on their different functionalities. According to the survey [21], there are mainly four payload functionalities, namely root exploit, remote control, financial charges, and personal information stealing. However, root-exploit malware aims at getting root privilege by invoking particular native codes, without requiring any sensitive permissions. So we focus on other three kinds of functionalities. As malware may have multiple functionalities, they may need to be split into different parts that achieve different functionalities.

Remote control is usually the prerequisite of financial charges and personal information stealing. This kind of malware receives network or SMS bot commands. Since this behavior may need INTERNET or RECEIVE_SMS permission, we separate this procedure from malware. We implement it as a way of code cooperation. When an application receives bot commands, it will start the exported components with corresponding behaviors in other applications.

Financial charges is a typical functionality for profit. In this scenario, malware send SMS messages or make background phone calls to premium-rate numbers. However, some malware hard-code premium-rate numbers and message contents, so we disperse content preparation and SMS sending into different applications. Background phone calls making can be transformed similarly. This kind of transformation can be also implemented as code cooperation. Intent containing these relevant information in one application starts the exported components with sensitive behaviors in other applications.

Personal information stealing is another functionality that is mostly related with malware motivations. In personal information stealing, malware are interested in collecting privacy information. Since information collecting and sending are the two main aspects in these malware, we can put these two procedures into two applications to reduce requested permissions in each one and crack the complete semantic. It is a typical scenario of data cooperation.

4. EVALUATION

In this section, we demonstrate the effect of our malware transfromation techniques against traditional methods on VirusTotal [1] and the evaluation results of DroidAlarm. We present our experiment results in Section 4.1. Furthermore, we put forward case studies for how to utilize unintentional leak in malware and how to construct privilege-escalation malware in Section 4.2.

Our test set picks up 49 malware from Android Malware Genome Project [20]. All experiments of DroidAlarm are performed on Ubuntu 10.04, which has a 512MB of RAM.

4.1 Result Overview

Firstly, we check each existing malware on VirusTotal [1] and their detection ratio is 61% on average. Secondly, we run DroidAlarm on these samples and verified the correctness manually. Thirdly, we recheck transformed malware.

Table 1 shows our progressive analysis results. Suspicious analysis module reports 33 samples are suspicious, which means they have both sensitive permissions and public interfaces. But sensitive permissions can be used independently from public interfaces in many scenarios. So in order to reduce false positive, we filter the results by building concrete paths from sensitive resources protected by permissions to public interfaces.

According to the results, we found the leak paths for three malware, namely AnserverBot, Bgserv and NickyBot. On one hand, most malware use sensitive resources internally. On the other hand, some malware send sensitive data to a fixed remote destination, and do not provide an open interface for other local applications. Compared with 67% alarm rate before leak path analysis to 6% after it, the number of false positive cases is largely reduced. For all leak paths in these three malware, we confirm them. Since we do not think they are actually designed that way, we define them as unintentional paths, which may also lead to *privilege escalation attack*. NickyBot will be discussed in detail in Section 4.2 to explore the potential exploitation.

In our experiments, we construct privilege-escalation malware by dispersing sensitive operations into multiple applications with three models introduced in Section 3.2. This transformation evade part of detections on VirusTotal [1], and Table 1 has showed detection ratios after our transformation. Because most of these methods only target a single application and cannot get a complete malicious pattern.

Based on transformable functionalities, we choose FakePlayer, GPSSMSSpy, DroidDream and GamblerSMS to perform all the three transformation models. From Table 1, we find the detection ratios have really declined a lot, which shows the techniques are very effective. We will describe the leak paths of DroidDream in detail in the following case studies. In order to compare different specific implementation techniques, we do further experiments, such as reducing the number of APIs, and removing auxiliary classes or its host app. And we find the detection ratios have different changes as shown in Table 1.

Some previous research works [12, 6] do detect part of transformed malware we build, by analyzing public interface forms of Intent and component. However, through analysis, we find that the forms of network sockets and file system are also used frequently. So these two forms cannot be ignored. As we recognize public interfaces on all the communication channels, we can easily find these public interfaces, and build concrete paths to show the capability leaks.

Table 1: Results on Existing Malware

Malware	O.r.	S.p.	P.i.	C.p.	F.m.	T.r.	Detected
ADRD	59%	✓	✓	✗	2	-	-
AnserverBot	70%	✓	✓	✓	2,3	-	-
Asroot	45%	✓	✗	✗	1	-	-
BaseBridge	86%	✓	✓	✗	1,2,3	-	-
BeanBot	39%	✓	✓	✗	2,3,4	-	-
Bgserv	70%	✓	✓	✓	2,3,4	-	-
CoinPirate	61%	✓	✓	✗	2,3,4	-	-
CruseWin	70%	✓	✗	✗	2,3,4	-	-
DogWars	75%	✓	✓	✗	3	-	-
DroidC.	34%	✓	✓	✗	1,2	-	-
DroidDeluxe	73%	✓	✗	✗	1	-	-
DroidDream	79%	✓	✗	✗	1,4	0%-77%	✓
DDlight	59%	✓	✓	✗	2,4	-	-
DroidK.F.1	64%	✓	✓	✗	1,2,4	-	-
DroidK.F.2	66%	✓	✓	✗	1,2,4	-	-
DroidK.F.3	61%	✓	✓	✗	1,2,4	-	-
DroidK.F.4	64%	✓	✓	✗	2	-	-
DroidK.F.S.	32%	✓	✓	✗	1,2,4	-	-
DroidK.F.U.	33%	✓	✓	✗	-	-	-
Endofday	68%	✓	✓	✗	2,3,4	-	-
FakeNetflix	68%	✓	✗	✗	4	-	-
FakePlayer	70%	✓	✗	✗	3	25%-38%	✓
GamSMS	46%	✓	✗	✗	3	20%	✓
Geinimi	61%	✓	✓	✗	2,3,4	-	-
GGTracker	73%	✓	✗	✗	3,4	-	-
GinMaster	80%	✓	✓	✗	1,2,4	-	-
GoldDream	61%	✓	✗	✗	2,3,4	-	-
Gone60	48%	✓	✓	✗	4	-	-
GPSSMSSpy	55%	✓	✓	✗	2,3,4	5%	✓
HippoSMS	50%	✓	✓	✗	3	-	-
Jifake	70%	✓	✗	✗	3	-	-
jSMSHider	66%	✓	✓	✗	2,3,4	-	-
Kmin	73%	✓	✗	✗	2,3	-	-
LoveTrap	66%	✓	✓	✗	3	-	-
NickyBot	45%	✓	✓	✓	2,3,4	-	-
NickySpy	61%	✓	✓	✗	2,3,4	-	-
Pjapps	58%	✓	✓	✗	2,3,4	-	-
Plankton	59%	✓	✓	✗	2	-	-
RogueLemon	33%	✓	✓	✗	2,3,4	-	-
RogueSPP.	71%	✓	✓	✗	3	-	-
SMSRep.	58%	✓	✗	✗	3,4	-	-
SndApps	59%	✓	✓	✗	4	-	-
Spitmo	69%	✓	✗	✗	2,3,4	-	-
Tapsnake	50%	✓	✗	✗	4	-	-
Walkinwat	72%	✓	✗	✗	3	-	-
YZHC	59%	✓	✓	✗	2,3,4	-	-
zHash	56%	✓	✓	✗	1	-	-
Zitmo	73%	✓	✗	✗	4	-	-
Zsone	64%	✓	✓	✗	3	-	-

S.p., P.i., C.p., F.m., O.r., T.r. and Detected mean sensitive permissions, public interfaces, certain path and functionality model, original ratio, transformed ratio and detected by DroidAlarm respectively.

1, 2, 3 and 4 mean root exploit, remote control, financial charges and personal information stealing respectively.

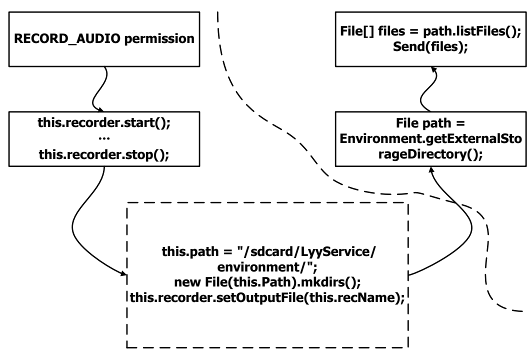
DroidK.F.U. is not malicious itself, but wheedles the user to install a malicious update, which is not included in our models.

4.2 Case Studies

To have a profound understanding of our work, we discuss two cases in depth. NickyBot is used to show unintentional capability leak in existing malware. And DroidDream variants demonstrate the practical pattern of personal information stealing.

4.2.1 NickyBot

NickyBot [14] records the surrounding sounds and stores audio files on SDcard. The lack of protection against SDcard files leads to the capability leak. The left part of the broken line in Fig. 4 included in the malware illustrates the capability leak path from RECORD_AUDIO permission to storing files on SDcard. An unprivileged attack app without the RECORD_AUDIO permission may obtain the audio files on SDcard and sends them out. It is a special



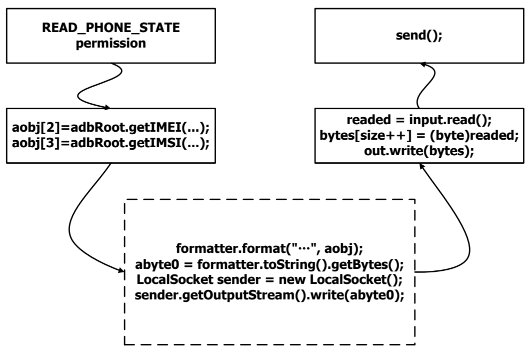
```
this.recorder.start();
...
this.recorder.stop();
```

```
this.path = "/sdcard/LyyService/
environment/";
new File(this.Path).mkdirs();
this.recorder.setOutputFile(this.recName);
```

```
File path =
Environment.getExternalStorageDirectory();
```

```
File[] files = path.listFiles();
Send(files);
```

Figure 4: Capability Leak Path of NickyBot



```
aobj[2]=adbRoot.getIMEI(...);
aobj[3]=adbRoot.getIMSI(...);
```

```
formatter.format("···", aobj);
abyte0 = formatter.toString().getBytes();
LocalSocket sender = new LocalSocket();
sender.getOutputStream().write(abyte0);
```

```
readed = input.read();
bytes[size++] = (byte)readed;
out.write(bytes);
```

```
send();
```

Figure 5: Capability Leak Path of DroidDream Variant

case that an API with sensitive behavior generates sensitive data. The file type can be changed to one of other three forms mentioned before with an operation mode as MODE_WORLD_READABLE.

4.2.2 DroidDream Variant

DroidDream [15] collects users' IMEI and IMSI, and sends them through URL network socket connection. It is a behavior of personal information stealing. As using URL network socket connection needs INTERNET permission, which is too loud and may be easily caught by traditional methods, we change it to LocalSocket that does not need such a permission. The leak path is demonstrated in Fig. 5. When the attack app obtains IMEI and IMSI, it codes and sends them through a LocalSocket. The unprivileged app without READ_PHONE_STATE permission receives the data from a same SOCKET_ADDRESS LocalSocket and sends them to our ftp server. The sensitive data can be also transmitted through other public interfaces.

5. RELATED WORK

There are several detection methods against *privilege escalation attack*. To discover the attack, IPC Inspection [11] combines call-graph analysis with communication tracking between applications, while QUIRE [8] tracks the call-chain of IPC and creates lightweight signatures for applications. A system-centric solution [5] monitors applications by extending relevant components in the OS system and decide whether there is a *privilege escalation attack* or not by customized permission policies. Compared to the above three methods, DroidAlarm does not require modifying the OS system or applications. And we use static analysis methods to avoid the low rate of path coverage and input effectiveness in dynamic methods. Woodpecker [12], CHEX [16] and DroidChecker [6] use static analysis techniques to detect capability leaks only at ICC level but not file system or network sockets. Woodpecker [12] allows for stock phone images but not applications. And DroidChecker [6] aims at source codes of applications, facing the difficulty of obtaining source codes and unsatisfactory decompiled results. However, DroidAlarm analyzes Dalvik bytecode directly. Meanwhile, we concentrate on all the communication channels to achieve an all-sided analysis.

6. LIMITATIONS AND FUTURE WORK

Our work has two aspects to improve. Firstly, since most auxiliary resources have a certain version, DroidAlarm is currently limited to Android 2.2. However, we can add a module to recognize the version and choose proper resources. Secondly, DroidAlarm ignores using data through file system or network sockets from other apps to do sensitive behaviors. However, we can settle it by building leak paths from pulic interfaces to sensitive permissions with existing methods.

7. CONCLUSIONS

In this paper, we demonstrate DroidAlarm to alarm *privilege escalation attack* by analyzing capability leaks in Android malware statically. Three malware out of 49 are considered to expose capability leaks. We then transform some malware with *privilege escalation attack* technique in three models to bypass traditional detections on VirusTotal [1]. And the detection ratio decreases in different levels on each variant. However, these cases can still trigger our alarm.

8. ACKNOWLEDGMENTS

We would like to thank our shepherd, Xuxian Jiang, and the anonymous reviewers for their comments. This work was supported in part by grants from the Chinese National Natural Science Foundation (61073027, 61272078, 90818022, and 61021062), and the Chinese National 863 High-Tech Program (2011AA01A202).

9. REFERENCES

[1] Virustotal. *https://www.virustotal.com/*.

[2] android-apktool. *http://code.google.com/p/android-apktool/*, 2011.

[3] androguard. *http://code.google.com/p/androguard/*, 2012.

[4] Permissions. *http://developer.android.com/guide/-topics/security/permissions.html#arch*, 2012.

[5] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastry. Towards taming privilege-escalation attacks on android. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.

[6] P. P. Chan, L. C. Hui, and S. Yiu. Droidchecker: Analyzing android applications for capability leak. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2012.

[7] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on android. In *Proceedings of the 13th Information Security Conference*, 2010.

[8] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S.Wallach. Quire: Lightweight provenance for smart phone operating systems. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[9] F-Secure. Warning on possible android mobile trojans. *http://www.f-secure.com/weblog/archives/00001852.-html*, 2010.

[10] F-Secure. Mobile threat report q3 2012. *http://www.f-secure.com/static/doc/labs_global/Research/Mobile%20 Threat%20Report%20Q3%202012.pdf*, 2012.

[11] A. P. Felt, H. Wang, A. Moschuk, S. Hanna, and E. Chin. Permission re-delegation: Attacks and defenses. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[12] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic detection of capability leaks in stock android smartphones. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.

[13] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications and Services*, 2012.

[14] X. Jiang. Security alert: New nickibot spyware found in alternative android markets. *http://www.csc.ncsu.edu/faculty/jiang/NickiBot/*, 2011.

[15] Lookout. Update: Security alert: Droiddream malware found in official android market. *https://blog.mylookout.com/2011/03/security-alert-malware-found-in-official-android-market-droiddream/*, 2011.

[16] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[17] D. Maslennikov. First sms trojan for android. *http://www.securelist.com/en/blog/2254/First_SMS_-Trojan_for_Android*, 2010.

[18] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *Proceedings of the 18th Network and Distributed System Security Symposium*, 2011.

[19] V. Svajcer. Deceiving permissions - rules for android malware detection. In *RSA Conference*, 2012.

[20] Y. Zhou and X. Jiang. Android malware genome project. *http://www.malgenomeproject.org/*, 2012.

[21] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[22] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.