Protection Against Broadcast Based Runtime Information Gathering on an Unrooted Android Operating System

Prashant Hari Narayan Rajput, Altaf Shaikh, Priyanka Sapka
Sinhgad College of Engineering, India

Abstract

Data privacy has been one of the greatest threats to mobile operating systems like Android. These operating systems have been susceptible to attacks on user data due to certain shortcomings in their implementation. For instance, consider Runtime Information Gathering (RIG) in which background malicious applications try to steal the information generated by the foreground application. Malicious applications leverage the permissions granted to them to their advantage, without needing any specific malicious implementation. So these attacks are partly a consequence of the user’s neglect while assigning permissions - a neglect which stems from lack of insight into the working of the Android OS. Antivirus and malware detection softwares are generally signature-based and hence prove ineffective when it comes to preventing RIG attacks. An application designed to protect a mobile device from such attacks should be based on behaviour, as it is easier to detect a RIG-based malicious application by observing its operation during runtime rather than analysing its signature. Our solution gives user an in-depth look into the working of the Android OS, enabling the user to monitor the working of all applications, which in turn makes detection of malicious behaviour easier. Moreover, we have developed a heuristic based approach for detecting suspicious applications at runtime. These heuristics use parameters such as CPU Usage, Dangerous Permissions, etc. for detecting malicious applications. Since our solution uses behaviour for identification of suspicious applications, this method proves successful in preventing RIG attacks.

1. Introduction

Android is one of the most popular operating system, owing to the fact that it provides a user Interface which is easy for users to grasp and which abstracts away the complexities of the Linux Kernel from the user. While this abstraction favours Android in the share market, it provides the user with less control over the working of the kernel. Users don’t have root control over the Android OS and rooting the phone voids the warranty of the smartphone. Android is a mobile OS based on Linux kernel, which is one of the most secure operating systems. While the Linux kernel performs well with desktop-based applications, it doesn’t translate well into the mobile platform. This kernel doesn’t have strict control over permissions and hence it is defenseless towards RIG.

From Marshmallow versions of Android OS, additional facilities have been provided to the users for restricting permissions that can be granted to a particular application. Users of Marshmallow can provide individual permissions to an application and deny granting others if they seem unnecessary. For example, granting GPS permissions to a calculator application can be avoided. Even after providing such a tight control over granting of permissions, users generally skim through such processes and grant unnecessary permissions to applications. It is generally because users are not well-educated about the different types of attacks that can be done leveraging on the permissions granted to them.

2. Motivation

Our project started with providing the user with adequate information about all the running processes on an Android OS so that the user can monitor all applications for malicious activities. Android doesn’t provide all the required information to the user directly. Instead, it uses pseudo files for storing temporary information and uses this information for control and administration of the OS. Our application bridges this gap by providing the user with required information for monitoring all the applications.

Providing access to this information can help the user monitor the working of all applications installed on the mobile device. This information is extracted from the pseudo files of the OS and some information is also extracted from APIs. Moreover, this information can also be used to analyze the behaviour of all the installed applications. Hence the information used for monitoring can also be used for detecting malicious applications based on behaviour. Similar to Guardian, our application also detects malicious application on runtime and prevents it from running in the background while runtime information is being produced by the foreground application [1].

3. Runtime Information Gathering

Runtime Information Gathering refers to the type of attack in which a malicious application tries to
steal the users’ runtime information while running in the background using side channels such as Bluetooth, UI Inference, etc [2], [3] & [4]. Generally, these types of malicious applications are based on broadcasts and start running when a certain broadcast is received.

**Call Recorder Attack.** This malicious application requires nothing but “RECORD_AUDIO” permission, using which it can record all calls the user receives or makes. The malicious application comes to know about the incoming or outgoing call using the broadcast “android.intent.action.PHONE_STATE”. Using this broadcast the malicious application starts running in the background and steals the users’ sensitive information. This recorded audio is stored in external storage temporarily and then can be sent to the remote server.

**Photo Sniffer Attack.** This malicious application receives the broadcast “android.hardware.action.NEW_PICTURE” whenever a new picture is captured, detecting which the application also gets the URI to the image stored in the external storage. The malicious application then uses this URI and uploads the newly captured image to a remote server without informing the user about it. For handling bursts shots, the malicious application uses multithreading, where each thread uploads individual shots from the burst mode onto the remote server.

**Location Stealer Attack.** This malicious application receives a broadcast whenever the GPS is turned ON/OFF, the broadcast being “android.location.PROVIDERS_CHANGED”. After its reception, the malicious application starts stealing the current location of the user. The application steals the user’s location every 10 seconds. Continuous information-fetching is not required due to the fact that there can’t be significant change of location within 10 seconds. This malicious application then stores this temporarily in a file which can later be sent to the remote server.

**Gyrophone Attack.** This malicious application receives the broadcast “android.intent.action.PHONE_STATE” for detecting incoming and outgoing calls and uses the vibrations recorded by the gyrophone coupled with machine learning for detecting user’s speech. The text file containing the gyroscope vibration recording can be sent to the remote server and machine learning can be applied on it for inferring the user’s conversation [6].

4. **Prior Work on Runtime Information Gathering**

Significant amount of work has been done in the field of RIG attacks [5] but very less amount of attention has been paid to the prevention of such types of attacks. While various techniques for exploiting the Android security sandbox have been explored and various others are the current research interests of many, a solution to this problem has often been ignored.

One of the first solutions to the Runtime Information Gathering problem was the Guardian Application [1] which prevented attacks by detecting malicious applications using the behaviour of such applications.

4.1 **Guardian Application**

Guardian Application is one of the first solution given to prevent RIG, and identifies malicious applications while data is being produced at runtime on the basis of behaviour.

4.1.1 **Overview.** The basic philosophy of Guardian Application is to prevent malicious application from running in the background when a principal application that needs to be protected is running in foreground. Such a simple strategy can be very effective in preventing RIG. This is because no malicious application is allowed to run in the background, hence preventing the misuse of the user’s runtime generated information. Guardian application has 3 main phases:

**Monitoring.** This is the first phase of operation of the guardian application, in which guardian application monitors all the background applications. It uses “getRunningTasks” to get the name of the application which is running in the foreground. Then the background applications are checked for dangerous permissions, based on the foreground application. For example, consider a malicious application that steals the user’s location whenever the user turns on GPS. In such situation the background applications should be checked for permissions such as “android.permission.ACCESS_COARSE_LOCATION” or “android.permission.ACCESS_FINE_LOCATION”. Any background application which has an increase in CPU Usage after receiving the broadcast with a corresponding increase in scheduling rate gives a fair indication that this particular background application might be a malicious application.


**Entering the Ward.** After detection of background malicious application, these applications have to be killed so that the data can be protected. Since users don’t have the root permission in Android OS, the “kill -STOP pid” command cannot be used. Instead, “killBackgroundProcesses(PackageName)” is used to kill the suspicious malicious application. Since all the malicious application run in background for stealing user’s information, their oom_adj score is generally more towards the positive side (range 17 to 15). Hence this method can be successfully used to kill those malicious applications. This method is used internally by the android OS for killing applications when the RAM memory becomes less than the minimum amount required by it. Hence only the applications that don’t have significant impact on performance on the system are killed using this command in Android.

**Exiting the Ward.** Guardian Application also restores the context of the killed suspicious application after the foreground application stops working. This is helpful because if a non-malicious application is killed after being detected as suspicious, it can again be restored and it can start functioning as normal. Guardian application uses a mechanism to let the app to be stopped and save its runtime state using onSaveInstanceState and restore the state once it is launched again using onRestoreInstanceState. This improves the overall performance of the Android OS while the Guardian Application works in the background [1].

4.1.2. **Shortcomings.** While Guardian Application is one of the best solutions against RIG, it has its shortcomings. For one, it uses the function “getRunningTasks” for detecting the foreground application, which has been deprecated from Marshmallow and the future versions of Android. From the marshmallow version onwards only the application which calls this function is displayed in the output. It does not provide the name of the actual application running in the foreground and hence this function can no longer be used for finding out the foreground application.

Secondly, Guardian application’s monitoring phase runs in the background as a service and for detecting malicious application, it uses certain information like Permission matrix, Scheduling rate, CPU Usage, etc. This information has to be collected from the pseudo files that have been created by the operating system. Accessing such information leads to high memory utilisation, leading to a performance bottleneck.

Thirdly, the Guardian Application maintains various heuristics on the remote server for preventing different types of attacks. Maintaining a remote server demands less amount of hardware from the mobile device, but it also leads to a delay in the detection of malicious application. This is because communication has to be established between the mobile device and the remote server. This will require the signature of all the applications to be sent to the remote server for detecting the malicious application. This leads to a delay in detection of malicious application.

Moreover, Guardian does not store the list of the malicious application in a database, which requires it to detect the malicious application each time it runs in the background. Even if it does, having the database containing the malicious application on the remote server creates delay in killing these malicious applications. Such a delay is crucial in the case of runtime information gathering as malicious applications have to be detected in runtime and killed before they can steal sensitive user information.

Monitoring phase is often considered as one of the most important part of preventing various attacks. Various attacks such as Denial of Service Attacks, RIG, etc all require monitoring of various installed background applications. Providing users a view inside the working of the android OS can help them make an informed decision in detecting malicious applications.

5. **Defender Application**

Defender Application is the solution we propose for preventing Runtime Information Gathering. We have prevented some RIGs such as Call Recorder Attack, Photo Sniffer Attack, Gyrophone Attack and Location Stealer Attack using the Defender Application. These attacks are explained below in brief:

**Call Recorder Attack.** In this type of attack the malicious application creates a thread for recording the user’s call which is then sent to the remote server. The recording thread is created when the calling application is running in the foreground.

**Photo Sniffer Attack.** In case of Photo Sniffer Attack a thread is created for each photo that is captured and then uploaded to the remote server. In general, the foreground application can be an application such as Camera Application.

**Gyrophone Attack.** Gyrophone Attack is an attack in which vibrations are recorded when a user makes or receives a call. These vibrations can then be studied to infer the user’s speech using machine learning [6].

**Location Stealer Attack.** This attack tries to steal user’s location each time the user turns on the GPS. This information is then sent to the remote server and the attacker can obtain the location of the user.
Prevention of all these attacks will be explained in the following sections.

![Figure 1. Defender Application’s Flow](image)

5.1. Overall Architecture

Defender Application is a solution to RIG and is inspired from the Guardian Application. It predominantly works on the feature of broadcast receiver, which is also used by the malicious applications. So the Defender application starts protecting the mobile each time it receives a broadcast that may lead to leakage of information from some type of RIG. For instance, we have implemented Defender application to prevent a mobile device against 4 types of RIG attacks which have been explained in the previous sections.

Defender application receives the broadcast and based on the broadcast it decides the type of solution to be applied to it. Since each type of solution has a different broadcast associated with it, this can be used for differentiating between the foreground application. Hence Broadcast Receivers play a crucial part of our solution system.

Based on the type of broadcast received, corresponding foreground application can be inferred. Then appropriate heuristics are applied for preventing a particular type of attack. After the user closes the foreground application which can be detected by another broadcast, the solution system can stop protecting the users' data.

Hence this system protects the user’s data whenever there is a foreground application which sends a broadcast that can be detected by our application. Moreover, a monitoring system is provided to the user for keeping a check on all the installed applications. As we all know monitoring phase of an attack prevention is the most crucial step. Many attacks can be prevented, if this step can be performed with full efficiency. Hence Defender Application provides users with insight into the working of the Android OS. All information that can be made available to the user without rooting of the phone is made available in our solution. There are various steps involved in our application which are explained in the following sections.

5.1.1. Detection of Broadcast. Prevention of RIG are based on behaviour of the malicious application. Since the attack involved varies depending upon the foreground application, we have to identify the foreground application so that we can apply appropriate measures for preventing the attack. So, detection of the broadcast is a preliminary step in attack prevention, and also plays one of the most important role in it.

**Call Recorder Attack and Gyrophone Attack.** For Call Recorder Attack a broadcast receiver with an intent filter of “android.intent.action.PHONE_STATE” is set up. This broadcast receiver is activated each time a user makes a call or receives an incoming call. Function onCallStateChanged(int state, String incomingNumber) is overridden and all the operations are performed in it. “PhoneStateListener” is used to get the current state and the various values of the state variable specify different state of the call. For instance, while the phone is ringing, the value of state is “TelephonyManager.CALL_STATE_RINGING”, whereas “TelephonyManager.CALL_STATE_OFFHOOK” is the state when the user is in call and “TelephonyManager.CALL_STATE_IDLE” when the call ends.

The main protection mechanism starts when the user goes into call. This is when the Defender Application searches for suspicious applications using the heuristics that are developed specifically for such type of attack. After finding the suspicious application an entry is made into the DangerousApplication table if the application entry is not already present in the safe application. Then this suspicious application is continuously killed until the call ends. Hence the state corresponding to the call end is the terminating state of the protection mechanism for this attack.

**Photo Sniffer Attack.** For prevention of the Photo Sniffer Attack, a broadcast receiver with the intent filter of “android.hardware.action.NEW_PICTURE” is used. This broadcast is received whenever the user...
captures a new photo from the camera application. This attack works with all the system camera application. There is also a possibility that the user’s camera might have burst mode facility and hence multithreading is used in this malicious application such that each thread is responsible for uploading one photo at a time.

So, when the broadcast of the aforementioned type is obtained, the protection mechanism starts executing.

**Location Stealer Attack.** For the prevention of the Location Stealer Attack, a broadcast receiver with the intent filter of “android.location.PROVIDERS_CHANGED” is used. Whenever GPS is turned On/Off the same broadcast is received. “isProviderEnabled(LocationManager.GPS_PROVIDER)” is used to check if GPS is enabled or not and this is used to differentiate between the broadcast received when the GPS is turned On and the one which is received when the GPS is turned Off.

Appropriate protection mechanism is started when the broadcast is received corresponding to when the GPS is turned On and the protection mechanism is turned Off when GPS is disabled.

**5.1.2. Finding Malicious Application.** Finding malicious application is the second phase in the prevention of RIG-based attacks. Since our solution is based on behaviour rather than signature like some antivirus out there, it is bound to make mistakes. Hence the malicious applications that are detected using the heuristics are the suspicious applications. They have a high probability of doing malicious activity in the background. But it cannot be said for sure that those applications have to be malicious.

We have used the combination of CPU Usage, Scheduling Rate and Permission Matrix for creating our heuristics. We have not included some parameters into our heuristics such as Ram Usage for reducing the number of calculation that have to be performed in runtime for detecting malicious applications. This helps to improve the overall performance of the system, moreover if an application is using CPU currently, then it is bound to have memory allocated currently in RAM, proving that parameter is redundant in our understanding.

Call Recorder and Gyrophone attack both receive the same broadcast, hence the heuristics used for preventing these attacks can be similar. We have used a combination of CPU Usage, Scheduling Rate - which is the average of all the observed reading - and Permission Matrix corresponding to these attacks.

CPU Usage can be parsed from the top command or it can be calculated from the values present in the file “/proc/{pid}/stat”. We have chosen the file parsing method over the top command method because it gave better performance.

Scheduling rate has been set based on the averages of all the observed rise in scheduling rate. Whenever a broadcast is received by the malicious application, they start executing their code for stealing user’s information. Since they start executing just after the broadcast is received there is sudden rise in the scheduling of the malicious application. This is because the Android OS schedules the malicious application for executing. This is an important heuristic parameter because it separates other applications that also receive the broadcast but don’t execute any malicious code. They are not scheduled for execution similar to malicious applications, making this one of the most important parameters in the heuristics.

**Call Recorder Attack and Gyrophone Attack.** Dangerous Permission to be checked in the background application for this particular attack is “android.permission.RECORD_AUDIO”. This is because an application with RECORD_AUDIO permission can record the complete conversation of a user while running in the background. Hence, after receiving a call or after dialing a call, if a background application with RECORD_AUDIO permission starts executing in the background, it could possibly be a malicious application.

**Photo Sniffer Attack.** Dangerous permissions used in the case of Photo Sniffer Attack are “android.permission.READ_EXTERNAL_STORAGE” and “android.permission INTERNET”. This is because the malicious application only gets the URI from the broadcast it receives. Then a new thread is created for uploading the image which is stored in the external storage over the remote server. Hence the malicious application needs both the READ_EXTERNAL_STORAGE for accessing the photo that needs to be sent to the remote server and the INTERNET permission for sending it to the remote server.

**Location Stealer Attack.** Dangerous permissions used in the case of Location Stealer Attack are “android.permission.ACCESS_FINE_LOCATION”, “android.permission.ACCESS_COARSE_LOCATION” and “android.permission.INTERNET”. This is because the ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION are used for accessing the GPS whereas INTERNET permission is used for sending the location of the user to the remote server.

**5.1.3. Strengthening Heuristics.** For optimising the heuristics and making it less susceptible to errors we implemented various improvements over the initial versions of the heuristics, out of which some are mentioned below:
We used a local database for keeping track of both Dangerous and Safe applications so that each time the protection mechanism runs, the already detected dangerous applications can be killed instantaneously so that malicious applications do not get a chance to steal the user’s information. This is particularly important because detection of malicious application at runtime using heuristics takes considerable amount of time and malicious applications can leverage from this time, a phenomenon which we call "Detection Latency". This detection latency can be exploited by certain applications to steal users' information while the detection process is still running. Hence, to avoid this each time, we have used a local database of Dangerous Applications which are killed as soon as a broadcast is received. This helps reduces the Detection Latency to a great extent.

We have also used Safe Applications table which consists of all the system applications installed in the mobile and also contains the list of applications that are sure to be safe. Therefore, while detecting malicious applications, if one of the background suspicious application is from this list, then its processing can be avoided and hence Detection Latency can be further reduced.

We also have remote database consisting of Dangerous and Safe Applications in which we store the information about dangerous and safe applications respectively. The local database present on the user’s mobile device can connect with the remote database and can fetch the list of safe and dangerous applications. Moreover, the local database can push the list of new malicious applications that has been detected by the heuristics. This creates a community in which enables other users using the Defender application to fetch this updated list of dangerous application onto their mobile and make their devices more secure. The community feature that we have provided in the Defender Application is an integral part of it because it helps other users avoid valuable computation time by updating the Dangerous Application Table from the remote Server. This in turn further reduces the Detection Latency.

Last but not the least, we have implemented separate local and remote database for reducing the delay that occurs in communication hence optimising the detection mechanism. If only a remote server had been used, then the protection mechanism would have required network resources all the time. Furthermore, there would have been considerable latency when we consider 2G networks. Hence, providing separate local and remote servers was beneficial in reducing the Detection Latency.

5.1.4. Preventing Malicious Application. Since our solution, similar to the Guardian Application, is aimed towards providing a solution to RIG without rooting of the mobile device, we cannot use commands that require root permissions. For instance, applications can be killed with simple “kill -STOP pid” if the user has root permissions, which is generally not the case because rooting a phone voids the warranty. Hence simple kill command cannot be used for killing suspicious malicious background applications.

One characteristic that is helpful in the process of killing suspicious applications is that these applications try to steal information stealthily. They require a foreground application to generate information that they can steal, and that in turn requires them to run in the background. Because of this, their oom_adj score is usually more towards the positive side. This oom_adj is the score that is used by the Android OS internally for killing various background applications when the primary memory gets below a threshold. The same idea can be used for killing malicious applications as these also run in background when the principal application is running in foreground.

For killing background applications “killBackgroundProcesses(PackageName)” function can be used which requires as a parameter the package name of the application that needs to be killed. This method as stated above is used to kill applications with oom_adj score which is towards the positive side in the range of -17 to 15. There is also a possibility that a malicious application might try to kill Defender Application with the same method we are using to kill it. For avoiding such a situation we have implemented our application to be run on the foreground and hence there will always be a notification displayed on the notification stack regarding our application. This decreases the oom_adj score of our application, hence preventing any malicious application from using the same method for killing our Defender Application.

There is no need for restoring the context of such applications because the applications that are killed using this method are generally the ones that are not performing any delicate tasks and hence are expendable. So, for reducing the overall cleanup time in the Defender application we have decided not to restore the applications that are killed in the background. This also saves useful computation time for the Defender application and hence improves the performance of the mobile while Defender application is installed on it.

5.1.5. User Control. Monitoring phase of attack prevention is generally considered one of the most important step, which is the reason we have focused our attention on it. We have collected reliable information from pseudo-files of the Android OS and provided the user with an overview of the complete state of the OS. This overview includes information about every installed application on the mobile
device. There are two types of information that are made available to the user:

Basic Information. Basic information about the application is collected from the pseudo-files, and gives the user general information about the applications. This information includes:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Basic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Application Name</td>
</tr>
<tr>
<td>2.</td>
<td>Process ID</td>
</tr>
<tr>
<td>3.</td>
<td>State</td>
</tr>
<tr>
<td>4.</td>
<td>CPU Usage</td>
</tr>
<tr>
<td>5.</td>
<td>Memory</td>
</tr>
<tr>
<td>6.</td>
<td>User ID</td>
</tr>
<tr>
<td>7.</td>
<td>Package Name</td>
</tr>
<tr>
<td>8.</td>
<td>Permission Matrix</td>
</tr>
</tbody>
</table>

Network Information. Network information about the application is collected from the pseudo-files, and gives the user insight into the network activity of all the applications. This information is crucial as it informs the user if an application is using network resources to send sensitive data even if it is not supposed to. Hence network information plays a crucial part in user control. Network information includes:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Network Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Application Name</td>
</tr>
<tr>
<td>2.</td>
<td>User ID</td>
</tr>
<tr>
<td>3.</td>
<td>Local Address</td>
</tr>
<tr>
<td>4.</td>
<td>Remote Address</td>
</tr>
<tr>
<td>5.</td>
<td>Transmitted Bytes</td>
</tr>
<tr>
<td>6.</td>
<td>Received Bytes</td>
</tr>
</tbody>
</table>

6. Evaluation and Analysis

6.1. Effectiveness

For evaluating the effectiveness of the Defender application, it was tested against four of the broadcast-based RIG attacks. Defender uses a behaviour based runtime detection mechanism for detecting malicious applications. Once a malicious application is detected on a mobile, it is guaranteed to be detected all the subsequent times based on our RIG protection module design. This is because Defender uses a local table for storing the information about the detected malicious application. Table 3 states all the attacks and their evaluations. All the experiments have been performed on a Micromax Canvas A1 mobile with 1.3Ghz CPU and 1GB Ram.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Recorder Attack</td>
<td>Defeated</td>
</tr>
<tr>
<td>Gyrophone Attack</td>
<td>Defeated</td>
</tr>
<tr>
<td>Photo Sniffer Attack</td>
<td>Defeated</td>
</tr>
<tr>
<td>Location Sniffer</td>
<td>Defeated</td>
</tr>
</tbody>
</table>

Call Recorder Attack. In the case of Call Recorder attack, the malicious application tries to run in background and record audio of the user while in a call. Our application tries to kill the malicious application as soon as it starts recording users’ audio. Since the malicious application is killed before it can record, users’ data is kept secure and hence even if the malicious application tries to send the recorded files to the remote server, empty files would be sent. Moreover the broadcast “android.intent.action.PHONE_STATE” is received multiple times and hence the malicious application has to be killed multiple times.

Gyrophone Attack. For the Gyrophone attack, our application prevents the malicious application from writing the values of the vibrations sensed from the gyroscope to a file saved in the external storage. Since this file is empty, no relevant information will be sent to the remote server.

Photo Sniffer Attack. Photo Sniffer malicious application gets the URI of the image that is captured and it uses ContentResolver for obtaining the path to that image. Since it is not possible to prevent the malicious application from getting the path to the photo, we kill the malicious application when it tries to upload the image to a remote server. This leads to some amount of data being uploaded to the remote server, but in our analysis we found out that this amount was insignificant for representing a complete image or even a part of an image.

Location Stealer Attack. Location Stealer tries to tap into the GPS Sensor and steal user's position when the GPS is turned On. For preventing this attack, we kill the malicious application before it can get any value from the GPS.

Defender has successfully defeated all the above-mentioned RIG Attacks. Moreover, the same algorithm can be extended for preventing other RIG attacks like ScreenMilker, UI Inference [2], etc making it reusable.

6.2. Impact on Performance

Preventing malicious applications at runtime has impact on the performance of the Android OS. This is because Defender has to collect information about
all the background applications at runtime and then detect malicious applications using our heuristics. Collecting this information from the background applications requires reading the pseudo files of the applications and hence significant amount of CPU is used.

Table 4. Performance Analysis of Defender Application

<table>
<thead>
<tr>
<th>Attack</th>
<th>CPU Usage (%)</th>
<th>Ram Usage (MB)</th>
<th>Scheduling Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Recorder</td>
<td>3.6</td>
<td>58</td>
<td>3099</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>3.6</td>
<td>58</td>
<td>3099</td>
</tr>
<tr>
<td>Photo</td>
<td>20</td>
<td>58</td>
<td>10</td>
</tr>
<tr>
<td>Sniffer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location Stealer</td>
<td>8.8</td>
<td>55</td>
<td>442</td>
</tr>
</tbody>
</table>

For the prevention of Location Stealer attack, our application has to create the object of LocationManager and hence it requires a CPU Usage that is more than what is required for preventing other RIG attacks. Hence it has a CPU Usage of an average of 20%. Defender application for all the other attacks do not require as much CPU and hence can function in background without impacting the performance of the mobile device.

7. Discussion

For preventing Broadcast based RIG attacks, Defender starts its protection mechanism as soon as a broadcast which has its broadcast receiver implemented is received. This broadcast is also received by the malicious application and hence the protection module starts executing parallelly with it. This is important for improving the performance of the Defender application because implementing the protection module as a service would utilise more amount of CPU Usage as compared to when implemented as a broadcast receiver. This is because services run continuously in the background for detecting malicious application, whereas the broadcast receivers are triggered by broadcasts and have a terminating condition for exiting the protection mechanism when needed.

For killing malicious applications quickly and for improving performance, the algorithm keeps track of all the detected suspicious applications by storing information about those in a local table. This table helps in killing these malicious applications quickly and for optimizing CPU Usage as well. Moreover, the community feature of the Defender application allows different users to share information about the malicious application and update their local databases by synchronising it with the remote Database. Defender maintains a separate local and remote database for detecting malicious applications quickly by reducing the communication lag.

The heuristics used to detect these malicious applications are static and hence have to be updated regularly for protecting against any type of new malicious application entering the android market. This can be achieved by using machine learning with our heuristics making it dynamic.

8. Conclusion

In this paper, we report our study on protection against broadcast-based RIG on Android. These attacks implement broadcast receiver for a particular event and after receiving the broadcast, they exploit the foreground application for sensitive user data. For preventing such attacks, we have implemented Defender, which receives same broadcasts as malicious apps. After receiving the broadcast, Defender detects suspicious applications that are currently running in the background based on CPU usage, Scheduling Rate and Permission Matrix. If a suspicious application is found then these are killed using the same function which is used by android OS for freeing the RAM. Defender works on non-rooted android devices so it does not void the warranty of the device.

Our evaluation shows that our solution works efficiently against all the four attacks mentioned in this paper with minimum performance cost. we believe our technique raises the concern about broadcast-based RIG on unrooted android operating system and inspire the follow-up effort on developing app-level protection against the various broadcast-based RIG threats.

9. Acknowledgments

We thank Dr. P. R. Futane and Prof. D. D. Gatade for their guidance.

10. References


