Malware Detection in Android

Geethu M Purushothaman, G Gopinadh, Nihar SNG Sreepada

Abstract— ***Android is currently the most popular smartphone operating system. However, users feel their private information at threat, facing a rapidly increasing number of malware for Android which significantly exceeds that of other platforms. The number of malicious applications targeting the android system has exploded in the recent years. And with the smartphone usage rapidly increasing, the operating system on which it runs is also becoming highly sophisticated. Starting from the discovery of a malware called Cabire in the year 2004 by Kaspersky, the mobile platform has seen numerous malwares down the years. Being the fastest growing market in smartphone operating systems, it has become the most viable target of security threats. While the security community, well aware of this fact has put together several methods for detection of android malware, many are based on permission and API usage or the identification of expert features. In this paper, we provide a framework for detecting malware namely Android.bgserv on Android mobile devices.***

Index Terms— **Android, Permissions, ProcessPackages, Bg_serv .**

I. INTRODUCTION

A malware which generally targets mobile phones or wireless-enabled Personal digital assistants (PDA), can cause collapse of the system and loss or leakage of confidential information. As the number of wireless phones and PDA networks have become surplus and have grown in complexity, it has become increasingly difficult to ensure their safety and security against attacks which are in the form of

viruses or other malwares. To protect the mobile users from the severe threat of Android malwares, many solutions have been proposed. Based on the report produced by the AV-Test, free Android anti-malware apps are simply not useful. As per their report six of the seven free apps tested failed to get above 10% detection. Only Zoner AntiVirus did any better, but it could only manage 32% detection. AV-Test also took a look at two paid anti-malware solutions for Android, the paid apps were able to scan and detect about half of all installed threats. That still leaves a huge number of malicious packages in the clear.

This Malware can also be termed as all kind of intrusions that is disastrous to the computer software and hardware system. Malware creators' write malware for various reasons and purposes ranging from economic gain to destruction to the mobile resources. Its growth is highly alarming in volume.

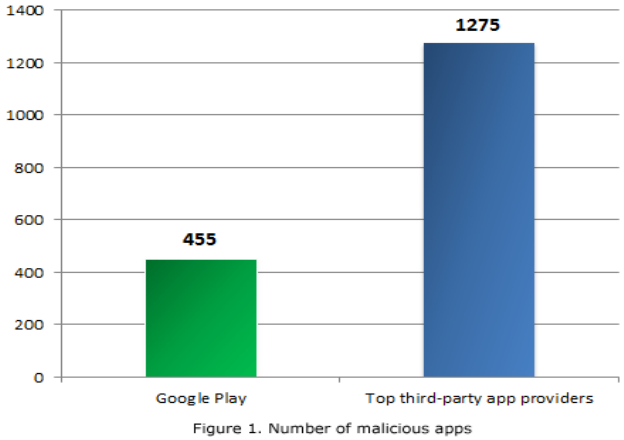


Figure 1. Number of malicious apps

Fig 1, shows the statistics of number of malicious applications in Google Play and Top third-party app providers. A malware can get itself into the system by different means like copying of files from external sources or devices onto the system and mostly by

Manuscript received April, 2014.
***Geethu M Purushothaman**, Computer Science and Engineering, ,Amrita Vishwa Vidyapeetham Coimbatore, India, 9677616801*
***G Gopinadh**, Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India, 7200549439.,*
***Nihar SNG Sreepada**, Computer Science and Engineering,Amrita Vishwa Vidyapeetham, Coimbatore, India,9003501816.,*

downloading files from the internet, it checks the vulnerabilities of the system and infects the system at the points of its vulnerability. Smartphone malwares are capable of doing many things, such as: stealing and transmitting the contact list and other data, locking the device completely, giving remote access to hackers, sending SMS and MMS messages without user notice or permission etc[2]. Mobile malwares is of great public concern as the population of mobile phones is much larger than the population of PCs.

Android which is an open source Operative System (OS) designed for mobile devices, such as smartphones and tablets, has currently the largest share of the mobile device market. The main reason of success is due to the large number of applications (or apps) that are available for Android devices, which can be developed using the Standard Development Kit (SDK). Android SDK is free to download and to use: hence, virtually anyone can develop applications in Android platform, from professional developers to any programmers with limited experience. Hence, it is always possible to download low-quality applications, or malicious ones, when surfing Android markets or the several unofficial markets found on the web, especially because these unofficial markets applications can be distributed without any kind of control. Android users can always face the risk of downloading and installing bad applications onto their devices. In fact, many applications may either hide malware, or their malicious behavior are not noticed.

Fig 2 shows the basic architecture of Android OS, Starting from the bottom, the lowest layer is the Linux Kernel. Android uses Linux for its device drivers, process management, memory management, and networking. The next level up contains the Android native libraries. They are all written in C/C++ internally, but will be calling them through Java interfaces. In this layer you can find the Surface Manager (for compositing windows), 2D and 3D graphics, Media codecs (MPEG-4, H.264, MP3, etc.), the SQL database (SQLite), and a native web browser engine(WebKit). Next is the Android runtime, including the Dalvik Virtual Machine. Dalvik runs dex files, which are converted at compile time from standard class and jar files. Dex files are more compact and efficient than class files, an important consideration for the limited memory and battery powered devices that Android targets. The core Java libraries are also part of the Android runtime. They are written Java, as is everything above this layer. They are written in Java, as is everything above this layer. Here, Android provides a substantial subset of the Java 5 Standard Edition packages, including Collections, I/O, and so forth. The next level up is the Application Framework layer. Parts of this toolkit are provided by Google, and parts are extensions or services that you write. The most important component of the framework is the Activity Manager, which manages the life cycle of applications. Finally, the top layer is the Applications layer. Programmers write code to this layer, alongside built-in applications such as the Phone and Web Browser.

Fig 2 Architecture of Android OS

Malicious applications are the greatest security threat for Android systems and, hence, to prevent such applications which damage smartphones, Android implements two security-control mechanisms: sandboxing and permissions. Sandboxing is achieved by means of application isolation: each application runs in its own instance of the Dalvik Virtual Machine (DVM), an optimization of the Java Virtual Machine, and each DVM is treated as a different UNIX user, by the Android's underlying Linux kernel. The isolation ensures that malicious applications do not interfere with the activity of the good ones. The permission system is a mechanism of access control to protect resources and critical operations. At install-time, permissions required by an application are shown to the user, which can decide whether to grant or to deny them. The main problem of this approach is that the acceptance policy for an

application's requested permission is "all or nothing", that is, the user cannot accept only a subset of the required permissions. Then, if the user does not agree even with a single permission, the installation is not performed. Furthermore, due to the large number of existing permissions, even an expert user may not fully understand all of them and, as a consequence, several users install applications without caring about the required permissions and without questioning about the potential security threats. In our paper we are focusing on one such malware which does not directly specify its hidden danger in its permission list or manifest file, but they still causes harmful effects to the system when these vulnerable application is installed [1] .The malware chosen was Adroid.bgserv (bg_serv) which is a Trojan horse which specifies the use of Bluetooth and wifi services just like any other android applications like Viber etc but they switch on Bluetooth and wifi automatically when the application containing the virus is installed. Automatic switching of Bluetooth and wifi can cause severe damage to the user device as the attacker gets direct access to the private details of the user. The acquired information is send by the attacker to C&C server which collects the following information and saves it in the file [INSTALLATIONPATH]/.hide/upload.xml:

1. IMEI
2. Phone Number
3. SMS Center
4. Install Time
5. System Version

It then uploads the collected information to the following remote site using the HTTP POST method: [http://] www.youlubg.com:81/Coop/reques[REMOVED] . Next, it receives commands from the reply to the POST and saves the commands the following file: [INSTALLATION PATH]/hide/serverInfo.xml .This allows the remote attacker to send SMS messages from the compromised device. The threat also has the capability to block incoming SMS messages.

So our goal is to create a framework which detects this Trojan, which is not detectable by an average user just by scanning through the manifest file. Each time an application is about to get installed, our detector which is otherwise in sleep mode becomes active and checks if the input .apk file is malicious or not.

II. PERMISSION SETTINGS IN ANDROID

This section explains the Android permission system in detail. Currently, Android defines 120 permissions. Out of these, each permission is related to a specific resource of the device or to an operation this is considered critical which can possibly harm the user privacy, her money, or her device itself. Permissions required by an application are specified in the AndroidManifest.xml file that is written as part of the an application itself and that is bound to it by means of digital signature[1].

The dangerous permissions are automatically shown to the user, whereas the normal ones are listed in a separate sublist addressed as "Other Permissions". If the user accepts *all the permissions* required by an application, then this application is installed and, at run-time, they are allowed to use the critical resources and operations granted to the permissions without asking for further authorizations.

Several criticisms have been raised against the Android permission system. Firstly, since the user has to choose between whether to accept all of the permissions specified by an application or to discard to install the application. Furthermore, the user may not be able to determine if an application can be trusted, based upon its listed permissions. Some of these permissions are really difficult to understand even to any expert user. It is often the case that an average user does not care about such permissions and their security hazards mainly due to lack of knowledge, thus installing potentially malicious applications. In short, Android users, seeing a very long permission list when installing a new application, pays less attention to read and understand them.

III. PROCESS AND THREAD SYSTEM IN ANDROID

When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution. By default, all components of the same application run in the same

process and thread (called the "main" thread). If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution.However, you can arrange for different components in your application to run in separate processes, and you can create additional threads for any process.

Processes

By default, all components of the same application run in the same process and most applications should not change this. However, if you find that you need to control which process a certain component belongs to, you can do so in the manifest file.
The manifest entry for each type of component element—<activity>, <service>, <receiver>, and<provider>—supports an android:process attribute that can specify a process in which that componentshould run. You can set this attribute so that each component runs in its own process or so that some components share a process while others do not. You can also set android: process so that components of different applications run in the same process—provided that the applications share the same Linux user ID and are signed with the same certificates.The <application> element also supports an android:process attribute, to set a default value that applies to all components. Android might decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user. Application components running in the process that's killed are consequently destroyed. A process is started again for those components when there's again work for them to do. When deciding which processes to kill, the Android system weighs their relative importance to the user. For example, it more readily shuts down a process hosting activities that are no longer visible on screen, compared to a process hosting visible activities. The decision whether to terminate a process, therefore, depends on the state of the components running in that process. The rules used to decide which processes to terminate is discussed below.

Process Lifecycle

The Android system tries to maintain an application process for as long as possible, but eventually needs to remove old processes to reclaim memory for new or more important processes. To determine which processes to keep and which to kill, the system places each process into an "importance hierarchy" based on the components running in the process and the state of those components. Processes with the lowest importance are eliminated first, then those with the next lowest importance, and so on, as necessary to recover system resources. There are five levels in the importance hierarchy. The following list presents the different types of processes in order of importance (the first process is most important and is killed last):

A. Foreground process

A process that is required for what the user is currently doing. A process is considered to be in the foreground if any of the following conditions are true:
It hosts an Activity that the user is interacting with (the Activity's onResume() method has been called).
It hosts a Service that's bound to the activity that the user is interacting with. It hosts a Service that's running "in the foreground"—the service has called startForeground().It hosts a Service that's executing one of its lifecycle callbacks (onCreate(), onStart(), or on Destroy()). It hosts a BroadcastReceiver that's executing its onReceive() method. Generally, only a few foreground processes exist at any given time. They are killed only as a last resort—if memory is so low that they cannot all continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive.

B. Visible process

A process that doesn't have any foreground components, but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions are true: It hosts an Activity that is not in the foreground, but is still visible to the user(its on Pause () method has been called). This might occur, for example, if the foreground activity started a dialog, which allows the previous activity to be seen behind it.It hosts a Service that's bound to a visible (or foreground) activity. A visible process is considered extremely

important and will not be killed unless doing so is required to keep all foreground processes running.

C. Service process

A process that is running a service that has been started with the startService() method and does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about (such as playing music in the background or downloading data on the network), so the system keeps them running unless there's not enough memory to retain them along with all foreground and visible processes.

D. Background process

A process holding an activity that's not currently visible to the user (the activity's on Stop() method has been called). These processes have no direct impact on the user experience, and the system can kill them at any time to reclaim memory for a foreground, visible, or service process. Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed. If an activity implements its lifecycle methods correctly, and saves its current state, killing its process will not have a visible effect on the user experience, because when the user navigates back to the activity, the activity restores all of its visible state. See the Activities document for information about saving and restoring state.

E. Empty process

A process that doesn't hold any active application components. The only reason to keep this kind of process alive is for caching purposes, to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches. Android ranks a process at the highest level it can, based upon the importance of the components currently active in the process. For example, if a process hosts a service and a visible activity, the process is ranked as a visible process, not a service process. In addition, a process's ranking might be increased because other processes are dependent on it—a process that is serving another process can never be ranked lower than the process it is serving. For example, if a content provider in process A is serving a client in process B, or if a service in process A is bound to a component in process B, process A is always considered at least as important as process B. Because a process running a service is ranked higher than a process with background activities, an activity that initiates a long-running operation might do well to start a service for that operation, rather than simply create a worker thread—particularly if the operation will likely outlast the activity. For example, an activity that's uploading a picture to a web site should start a service to perform the upload so that the upload can continue in the background even if the user leaves the activity. Using a service guarantees that the operation will have at least "service process" priority, regardless of what happens to the activity. This is the same reason that broadcast receivers should employ services rather than simply put time consuming operations in a thread.

IV. THE PROPOSED SYSTEM TO DETECT ANDROID.BGSERV

An Anti-malware application was developed and the corresponding .apk file of this application (like .exe file in Windows OS) was installed in your mobile phone. During the build process, the android projects are compiled and packaged into an .apk file, the container for the application binary. It contains all of the information necessary to run the application in a the device or emulator including the .dex files(.class files converted into Dalvic Byte code), a binary version of Android Manifest.xml file, compiled resources (resources.arsc) and uncompiled resource files for the application .

Once the application is successfully installed in the system, then processpackages have to be installed . These processpackages contains the resource classes used by applications included in the platform and defines application permissions for system features. For example, few processpackage classes which are generally used are androidaccessibiltyservice, android.backup, android.graphics etc. The application then gets registered in device OS kernel.

The proposed Anti-malware application is set in such a way that they are active only when a new foreign

application or .apk file approaches the system to get installed. The system otherwise they remain inactive or in sleep mode, this is to save battery consumption by this application. If they are active always, they may consume a lot of power as they always work in the background and does unwanted checking.

Every application in Android runs as a separate kernel-level user, with a unique user ID and group ID. Every app gets its own directory for saving data. The Linux user id system is used to make sure that apps can't read each others' data. But all these data directories are inside one directory of the system ie, /data/user/0 . The proposed sytem makes sure that a user's app can't read other app's used by the user as well as the user's app can't read the apps of the other users. This is the usual process followed. But apps can be written to share the data directory for multiple applications written by the same user.

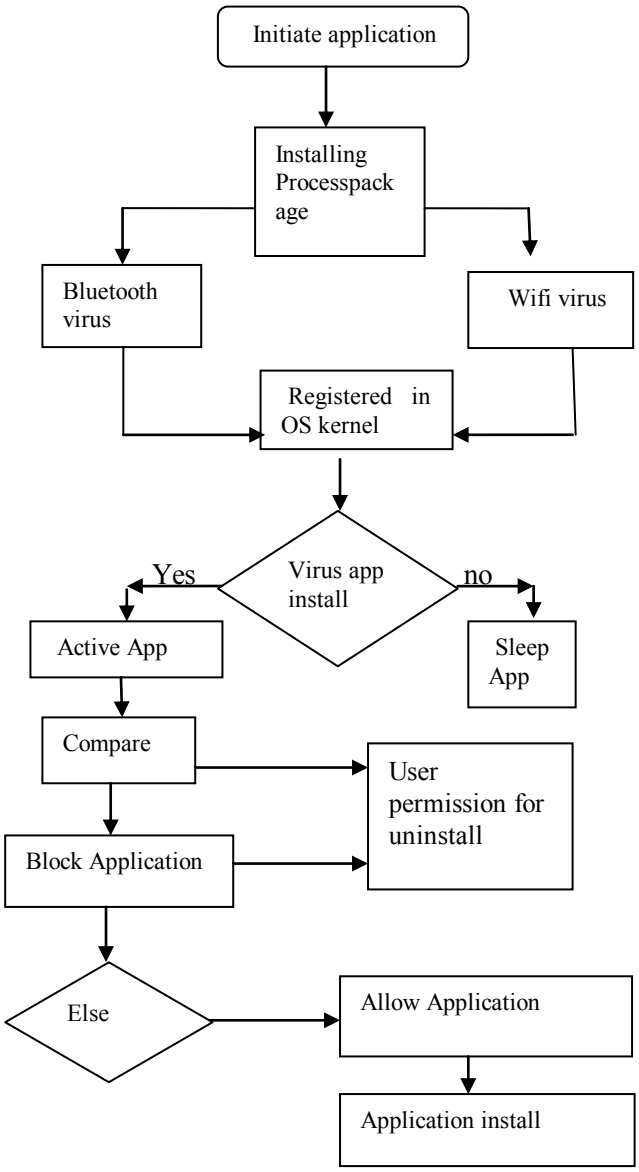
Android.bgserv is a trojan which is found in android based systems, which has characteristics of automatically switching on Bluetooth and wifi options of the device without the user concern. Switching on Bluetooth and wifi options can cause severe damage to the system. The proposed framework realizes a Host-based Malware Detection System that checks any new application before installing into your smartphones. Here,in this detection is done at the root level. If the application is found malicious, the system would compel the user to uninstall the application for the further safety of user's mobile device. For testing purposes, two malicious 'hello world' applications were manually developed, the system's ability to detect the malicious content was evaluated. The system would work for any android application containing the same vulnerabilities.

Now a new foreign application has to be installed in the same system. The antimalware application which was still in sleep mode becomes active and they checks through the .apk file of this new application. The packageprocess analyze the installation application and alerts the user if any malware found. For that we use Broadcast receiver declared as follows.

```
<receiver
android:enabled="true" android:exported="true"
android:label="BootService"
android:name=".PackageInstallationReceivr">
<intent-filter>
<action
android:name="android.intent.action.PACKAGE_ADDED"/>
<data android:scheme="package"/>
</intent-filter>
<intent-filter>
<action
android:name="android.intent.action.PACKAGE_INSTALL"/>
<data android:scheme="package"/>
</intent-filter>
<intent-filter>
<action
android:name="android.intent.action.PACKAGE_CHANGED"/>
<data android:scheme="package"/>
</intent-filter>
<intent-filter>
<action
android:name="android.intent.action.PACKAGE_REPLACED"/>
<data android:scheme="package"/>
</intent-filter>
</receiver>
```

As specified it calls the PackageInstallationReceiver class when new application installs. This compares the current installation .apk file package name with the predefined package name . If it equals alert displayed to the user that this is the malware and uninstall the application for user safety. If the user still installs the application , it is at user's own risk.

SYSTEM FLOW DIAGRAM



V. CONCLUSION

Android systems are exposed to malwares as anyone can develop an application and upload in Google Play. As per research still more than 17300 applications are malware applications. Hence malware detection techniques are important. In this paper, we have created a framework which detects Android.bgserv malware.This application automatically switches on the Bluetooth and wifi services of the system , and makes the system vulnerable to attacks. Our system on detection displays to the user an alert messenger which indicates the presence of malware and forces the user to uninstall the application. The application that is prepared will be runnable on devices having Android 2.2 and above. The phone would require a working internet connection and a good memory , that is main limitation of our system. Eventhough the system works for all applications which have Android.bgserv , our system does not consider more malwares .As further scope of this system , more malwares can be considered , signatures of these malwares can be added to the antimalware database .

VI. REFERENCES

[1] Zarni Aung and Win Zaw., "Permission-Based Android Malware Detection".,International Journal Of Scientific & Technology Research, March 2013.

[2] Asaf Shabtai et al., "Andromaly : a behavioral malware detection framework forandroid devices," Research Paper in Springer Science & Business Media., LLC2011.

[3] Vanja Svajcer., "Deceiving Permissions - Rules for Android Malware Detection", a talk at RSA Conference USA 2012.

[4] Carlos A. Castillo, "Android Malware Past, Present, and Future", Performance report published in Mobile Security Working Group, June 2010.

[5] Iker Burguera and Urko Zurutuza.,"Crowdroid: Behavior-Based Malware Detection System for Android", SPSM Conference Chicago, USA, October 2011.

[6] Xuetao Wei, Lorenzo Gomez et al., "Malicious Android Applications in the Enterprise: What Do They Do and How Do We Fix It?", University of California,February 2012.

[7] Android: From Reversing to Decompilation -Anthony Desnos, Geoffroy Gueguen – 2011 .

[8]Vulnerabilities faced by Smart Phones - John R Dilworth, Walter Mirtle – 2011.

[9] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.R. Sadeghi, B. Shastry: Practical and Lightweight Domain Isolation on Android. In ACM, ed.: 1st ACM workshop

on Security and privacy in smartphones and mobile devices (SPSM11). (2011) 51 – 61.

[10] Y. Zhou, X. Zhang, X. Jiang, V. W. Freeh: Taming information-stealing smartphone applications (on android). In: 4th International Conference on Trust and Trustworthy Computing (TRUST 2011). (2011).

[11] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner: Android permissions: User attention, comprehension, and behavior. Technical report, Electrical Engineering and Computer SciencesUniversity of California at Berkeley (2012) http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-26.html.

[12] A.P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner: Android Permissions Demystified. In ACM, ed.: 8th ACM conference on Computer and Communications Security (CCS'11). (2011) 627 – 638.

[13] Xuxian Jiang: Multiple Security Alerts: New Android Malware Found in Official and Alternative AndroidMarkets(2011)http://www.csc.ncsu.edu/faculty/jiang/pubs/index.html.

[14] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, D. Sgandurra: A Multi-Criteria-Based Evaluation of Android Applications. Technical report, Istituto di Inforatica eTelematica, CNR,Pisa(2012)url: http://www.iit.cnr.it/node/17019.

[15] Saaty, T.L.: Decision-making with the ahp: Why is the principal eigenvector necessary.European Journal of Operational Research 145(1) (2003) 85–91.

[16] Saaty, T.L.: Decision making with the analytic hierarchy process. International Journal of Services Sciences 1(1) (2008).

[17] Saaty, T.L.: How to make a decision: The analytic hierarchy process. European Journal of Operational Research 48(1) (1990) 9–26.

[18] Saaty, T.L.: A scaling method for priorities in hierarchical structures. Journal of Mathematical Psychology 15(3) (1977) 234–281.

[19] Adam, P. F., Chaudhuri, A., & Foster, J. S. (2009). SCanDroid: Automated security certification of android applications. In *IEEE symposium of security and privacy*.

[20] Bose,A., Hu, X., Shin, K. G.,&Park, T. (2008). Behavioral detection of malware on mobile handsets. In *Proc. of the 6th international conference on mobile systems, applications, and services*.

[21]Botha, R. A., Furnell, S. M., & Clarke, N. L. (2009). From desktop to mobile: Examining the security experience. *Computer & Security, 28*, 130–137.

Geethu M Purushothaman is pursuing BTech degree in the department of Computer Science and Engineering from Amrita Vishwa Vidyapeetham, Coimbatore, Tamilnadu, India. Her areas of interest include Information Security, and Networking.

G Gopinadh is pursuing BTech degree in the department of Computer Science and Engineering from Amrita Vishwa Vidyapeetham, Coimbatore, Tamilnadu, India. His areas of interest include Cyber Security and Database Management Systems.

Nihar SNG Sreepada is pursuing BTech degree in the department of Computer Science and Engineering from Amrita Vishwa Vidyapeetham, Coimbatore, Tamilnadu, India. His areas of interest include Information Security, and Software engineering.

