International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 02 Issue: 06 | Sep-2015 www.irjet.net p-ISSN: 2395-0072

Android Anti-malware Against Transformation Attacks

Ajinath N. Pawar[1], Saiprasad K. Malekar[2], Rupali A. Holkar[3], Poonam S. Ahire[4], Prof. Kavita R. Wagh[5]

[1234]UG Student, Computer Engg. Department, B.V.C.O.E & R.I, Maharashtra, India
[5]Lecturer, Computer Engg. Department, B.V.C.O.E & R.I, Maharashtra, India

---

Abstract *- **Android is presently the most popular and useful operating system for mobile. Attack of malware threats have recently became a real problem in smartphone.In this paper, we have stated a simple and high efficient technique for detecting malware Android applications on Play store which need to be installed. In addition, a majority of them can be find by applying risk score over known malware with less effort.If the applications is having some malicious intention; it might be possible that most of these applications come from an unknown developer and so there is higher possibility of them being malicious. To overcome these two problems, we have to developed a system in which we may consider different sources to collect the information about the applications like information from the labels (application name), from search engine, contextual usage history of the application collected from the users usage record and the permissions of the applications, which they have request at the time of installation giving us a secure and effective classification of the applications. We have compared our results with the exiting categories of the applications given on a play store; it provides appropriate results with defined categories**.*

Key Words: *Risk, Malware, Mobile, Android, Anti-Malware, Security, Mobile Apps.*

1. INTRODUCTION

Mobile devices such as smartphones, tablets and palmtop computers are becoming more popular.Unfortunately, this popularity attracts malicious attacks too.Currently, mobile malware has already become a serious concern.It has seen that in Android, one of the most popular smartphone platforms, malware has constantly been on the increase.With the rise of malware attacks, the platform has seen an evolution of anti-malware tools, with a range of free and paid service that is now available in the official Android mobile app. Market called Google Play Store.

In this paper, we aim to evaluate the capacity of anti-malware tools on Android on various evasion techniques.For eg., polymorphism is a technique used to avoid detection tools by changing a malware in different forms but with the exact code.Also there is another technique called metamorphism which can change the code when it no longer remains the same but still has the same action.For making simple presentation in this paper, we use the word 'polymorphism' to express both obfuscation techniques.Additionally, we have use the term 'transformation' deeply for reference of various polymorphic or metamorphic changes. Our domain of study is different from that we exclusively focus on mobile devices like smartphones, tablets that require various ways for anti-malware design.Malware attacks on mobile devices have recently increased in extent their evolution but the capabilities of existing anti-malware tools are difficult to understand.

To evaluate existing anti-malware software, they have developed number of systematic framework such as Droid Chameleon [1] with different transformation techniques that may be used in a system which can change Android applications automatically. Some of these changes are highly specific for the Android platform.Based on the framework, we pass known malware samples (from different families) through these changes we generate new variants of malware, which verifies to possess the' original malicious functionality. We use these variants to evaluate the effective popular anti-malware tools.

Polymorphic attacks have long been a problem for traditional desktop-server systems. Previous studies on the effectiveness of anti-malware tools on PC's [5], our domain of study is different in that we have exclusively focus on mobile devices like smartphones, tablets and palmtop computers which require different ways for anti-malware design. Also, malware on mobile devices have recently escalated their evolution but the capability of existing anti-malware tools are not yet understood.In the meantime, simple forms of polymorphic attacks already takes place in the wild [6].

We regularly and systematically evaluate anti-malware products for android regarding its resistance against various transformation techniques in known malware space. So we developed Droid Chameleon, a regular and

systematic framework with various transformation techniques.

We have implemented a prototype of Droid Chameleon and used it to evaluate ten popular anti-malware products for Android.Our findings show that all of them are vulnerable to common evasion techniques.The signatures studied do not require static analysis of byte code.

We have been studying the evolution of anti-malware tools over a period of two year.Our basic findings show that some anti-malware tools try to strengthen their signatures with a trend towards content-based signatures while previously they escaped by certain transformations not involving code-level changes.The improved signatures still show to be vulnerable..

Based on our evaluation results, we explored possible ways to improve current anti-malware solutions.To be precise we highlighted out that android eases developing modern detection techniques because much code is high-level byte codes rather than native and primary codes. Lastly, certain platform support can be enlisted to cope with advanced transformations.

2. LITERATURE SURVEY

An automated and extended platform to stress test Android anti-virus systems" was developed by M. Zheng, P. Lee, and J. Lui in July 2012 known as ADAM.[2]. It was an automated and extended system that evaluates the usefulness of anti-virus using various malware for Android platform. It automatically changes an Android malware samples into different variants through various repackaging and difficult techniques, while preserving the original malicious behavior.

ADAM can automatically change an original malware sample to different variants via repackaging and difficult techniques in order to test the effectiveness of different anti-virus systems against malware mutation [2]. ADAM is designed by connecting different building blocks. These blocks are tested using different anti-viruses against malware samples

Advantages -It can be used for study of very large-scale malware samples and changes is done manually so there is no need to change manual modification of malwares.

ADAM is not capable to prevent an anti-malware tool. It implements only some of changes, such as renaming methods, introducing junk methods. ADAM will never provide the best sensing mechanism which is also its main limitation of this system.

“A taxonomy of obfuscating [3] transformations”, stated by C. Collberg, C. Thomborson, and D. Low, Dept. Computer. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997. It has been the focus of much interest due to its relevance. This helps to preserve privacy policies between sender and receiver. In this technique Executer does the actual execution.

Advantages-Obfuscation can be easily used to trace software pirates.

Limitations- The obfuscated software remains secret and hidden until the powerful removal tool is to be built. Therefore, there must be very little time lengths between the releases of obfuscated software and its new versions.

Some tools like the Malware Detection by Semantics technique which was invented by M. Christodorescu, S. Jha, and C. Kruegel [4], in the year 2007,proposed that malware detector can be used to find out the malicious behavior of a program. Many times hackers use complex techniques to change the malwares. So, here the detectors use pattern-matching technique to search the complex techniques made by hackers. The benefit of this system is that it is fully syntax based technique. Therefore this makes it easy to be understood by detectors and it has relatively low run time overhead. Limitation is compulsory to prevent and save the remnants of harmful instructions into templates which needs large databases.

“Effective and efficient malware detection at the end user,” was developed by C. Kolbitsch[5], P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, in Proc. 18th Conf. USENIX Security Symp in the year 2009. It proposed a new malware detection approach that is effective and 100% effective, therefore can be used to replace old anti-virus tool at the end user. This method uses a malware to build a model that characterizes its behavior. Such designs describe the information motion between the system which is essential to the malware's mission, and therefore, cannot be easily avoided by simple obfuscation or polymorphic methods. one must extract the program slices which are responsible for such information flows. For detection and identification, execute these to match with these models against the run-time behavior of an unknown software.

Advantages- It can effectively sense whether it is running malicious software or code on an end user's host with a small overhead. It generates powerful tool that captures detailed information how behavior of a malwares variation occurs. Scanner can accurately match the activity of an unknown software program against this system.

Disadvantages-It cannot generate system call signs or find or detect a starting point for the process. The new algorithms should be implemented for above limitation.

"Risk-Ranker:accurate 0-day android malware detection" was proposed in year 2012 by M. Grace, Q. Zhang, S. Zou, and X. Jiang[6]. It proposed proactive scheme to spot zero-day Android malware, It does not stay on malware samples and its signatures. It is an automated system called Risk-Ranker which analyzes whether a particular app exhibits harmful behavior (example launching a root exploit or sending background Short messaging system(SMS) messages). It analyses and converts potential security risks into its similar sensing and detection modules of two orders of complexity. The first-order modules handle non-complex apps by analysing and evaluating the risks ; the second-order modules capture different and specific behaviors to search and analyse specific malwares.

Y. Nadiji, J. Giffn and P. Traynor proposed "automatic remote repair of malware" in the year 2011.In this the malicious network traffic increases because of intruders. The diificulty can be analysed and solved by using Air-mid, which is an automated system for remote removal of mobile malware. After the sensing and detection of malicious and harmful traffic.

Disadvantages-It does not stick and stay to device and its security. It is not able to define the traffic of large amount of malwares. "Apps-Play-ground: Automated Security Analysis of Smartphone Applications", was developed in Feb 2013, by V.Rastogi, Chen and W.Enck, to do the automation of security analysis the tool apps playground is used. It incorporates multiple components comprising different detection and automatic exploration techniques for this purpose [8]. The system can be checked using multiple large-small scale experiments involving real cancerous application. The main advantage of this technique is that it gives effective and correct analysis even with huge number of applications, with disadvantage of less correct and effective at automatically checking privacy leaks.
"Hey, you, get off of my market: this was developed by Y. Zhou, Z. Wang, W. Zhou and X. Jiang in the year 2012.

To find out cancerous and malicious applications related to android permission based characteristic foot printing is used. It is used for known malwares. Then a filtering scheme is applied to unknown and suspicious malwares. the total system with different types of malicious and cancerous families is called Droid-Ranger [9].
Benefits- It helps to concentrate on both official and unofficial or unsupported Android markets for detecting malicious applications and softwares .By using known and unknown malicious applications the detection proves to be scalable and efficient.
Limitation- It needs rigorous policies active process especially for unofficial marketplaces which is not satisfied by Droid Ranger yet.

3. EXISTING SYSTEM

In existing malware detection system like anti-virus we first download any file or media then anti-virus scan that files and detect the viruses or malwares.

To evaluate existing anti-malware software, they have developed number of systematic framework such as Droid Chameleon[1] with different transformation techniques that may be used in a system which can change Android applications automatically. Some of these changes are highly specific for the Android platform. Based on the framework, which we pass known malware samples (from different families) through these changes we generate new variants of malware, which verifies to possess the' original malicious functionality. We use these variants to evaluate the effective popular anti-malware tools. Droid-Dream [12] and BaseBridge [13] are malware with root exploits packed into benign applications.DroidDream tries to get out root privileges by using two different root exploits, rage against the cage and exploid exploit. BaseBridge includes only one exploit, rage against the cage. If these exploits are successful, both DroidDream and BaseBridge install payload applications. Geinimi [14] is a trojan packed into benign applications. It communicates with remote C&C servers and exfiltrates user information. Fakeplayer [15], the first known malware on Android, sends SMS messages to premium numbers, thus costing money to the user. Bgserv [16] is a malware injected into Google's security tool to clean out DroidDream and distributed in third party application markets. It opens a backdoor on the device and exfiltrates user information. Plankton [17] is a malware family that loads classes from additional downloaded dex files to extend its capabilities dynamically.

4. PROPOSED SYSTEM

In proposed system we first find out the risk scores of app which you want to download from google play store and check whether the risk score of that particular app is high or low, if we found the low risk score, then download the app but if we found very high risk score then find out the similar kind of app from google play store which having low risk score.

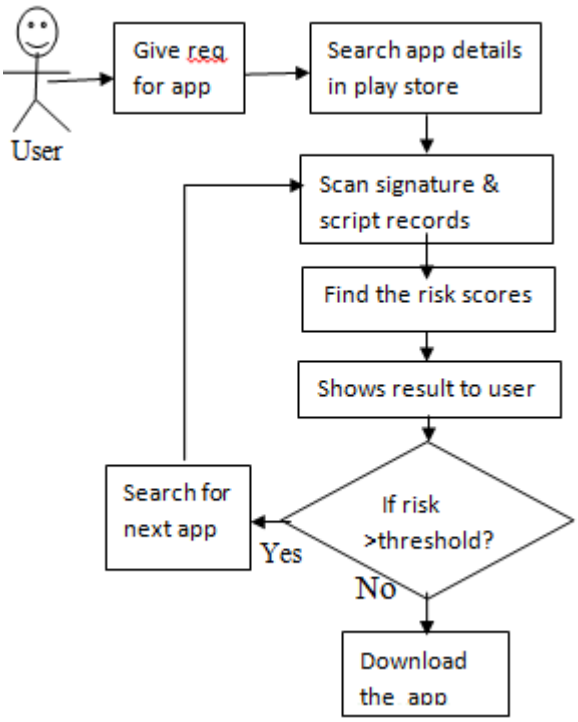

Fig.1

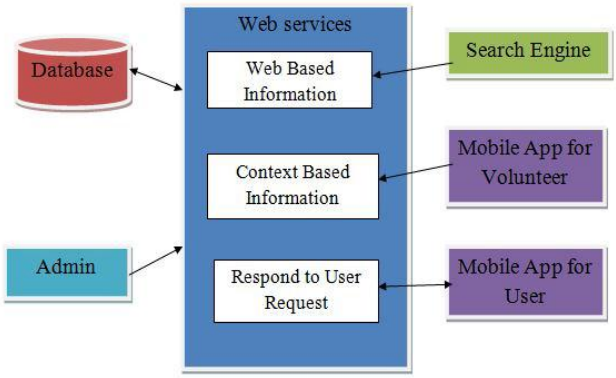

Fig.2

5. ALGORITHMS/TECHNIQUES

Algorithm for Android anti-malware against transformation attack is given below:

Step 1: Start

Step 2: User give request for app.

Step 3: System search app details on Play store.

Step 4: System scans the signature and script record for app which is requested by user.

Step 5: Finding the risk scores of app requested by user.

Step 6: if risk score is higher than threshold then

Search for next app.

Go to step 4

Step 7: Else result is low risk score then

download the app directly.

Step 8: Stop

Algorithm: Naive bayes classifier:

Input: D set of tuples with n number of attributes as Z= $(z_1, z_2, z_3 \ldots z_n)$, 'm' Classes: $(c_1, c_2, c_3 \ldots . c_m)$

Output: Predicts Class c_i of item Z

Processing:

1: Start
2: Calculate probability P of X with each Class c_i
3: Check whether P $(c_i|Z) > P(c_i|Z)$
4: Calculate P $(c_i|Z) = P(Z|c_i)\ P(c_i)\ |P(Z)$
5: Maximize P $(Z|c_i)\ P(c_i)$ as P(Z) as P(Z) is constant
6: Probability can be calculated as:
P $(Z|C_i) = P(z_1|c_i) * P(z_2|c_i) * \ldots * P(z_n|c_i)$
7: Stop
Where,
P (c|Z) = posterior probability of class (target) given predictor (attribute).
P(Z) = prior probability of predictor.
P (c) = prior probability of class.
P (Z|c) = likelihood which is the probability of predictor given class

Naïve Bayes Model:

It is a machine learning model which is similar to Max Entropy model.In this model, it assumes that the features of these model are conditionally independent of each other. It is based on the Bayes theorem which is being used in the final phase for calculating the risk score of the application. It is also explained in generalized form as [18]:

P (c| x) = (P(c|x) P(c)) ÷ P(x)

Where ,

P(c|x) is posterior probability of class (target) given predictor (attribute).

P(c) is the prior probability of class.

P(x|c) is likelihood which is the probability of predictor given class.

P(x) is the prior probability of predictor class.

5.1. Techniques

In our system we are also using the various transformation techniques which are stated below:

A. Trivial Transformations.

1) Repacking: Android packages that are recalled are signed jar files. Files could be unzipped with the regular zip utilities that are repacked again with the tools available in the Android SDK. Once these packages are repacked these applications are signed with custom keys.

2) Disassembling and Reassembling:The Dalvik bytecode which gets compiled in classes.dex of the application package could be disassembled and it will be reassembled back again.The various items such as classes, methods, strings, and so on in a dex file could be arranged or can be expressed in more than one way and a compiled program may be represented in various forms. Signatures which match the whole classes.dex are formed by this transformation.

3) Changing Package Name: Every application which is unique to the application is established by package name. This name is present in the package's AndroidManifest.The change can be made to the package name for a given malicious application to another package name.The package names of applications are unique concepts for Android and quite similar transformations that does not exist for other systems.

B. Transformation Attacks Detectable by Static Analysis (DSA)

DSA transformations having applications do not terminate all types of static analysis.Specifically,forms of such analysis which describe the semantics, such as data flows are still possible.String matching or matching API having simpler checks calls could be thwarted.

1) Identifier Renaming:

We can also rename most class, method, and field identifiers in a byte code.We come across such several free obfuscation tools like ProGuard [10] which provides identifier renaming. In Listing 1, listing 2 presents an example transformation for code.

2) Data Encoding:

All the strings and array data which has been used in the code contains all the dex files. From these strings and arrays,data encoding could be used to develop signatures against malware. To formed such signatures we can keep them in a encoded form. In Listing 1,listing 3 shows code which are changed by string encoding.

3) Call Indirections:

Such changes can be seen in a easier way to maintain call graph of the application to defeat automatic matching. For

a method call which is given, the call is converted to a call to a previously non-existing method that then calls the method in the original call. This can be done for all the calls, that goes out into framework libraries as well as those within the application code. This transformation could be seen as trivial function outlining [11].

4) Code Reordering:

It reorders the instructions in the methods of a program. This transformation is accomplished by reordering the instructions and thus inserting go to instructions for preserving the runtime execution sequence of the instructions.

5) Junk Code Insertion:

These transformations have introduced code sequences which are executed but does not affect the rest of the program. Detection based on analysing instruction (or opcode) sequences can be beaten by junk code insertion. Junk code constitute simple nop sequences or many sophisticated sequences and branches that actually have no effect on the semantics.

6) Crypting Payloads and Native Exploits:

In Android, native code is usually made available for libraries that are accessed via JNI. However, some malware such as DroidDream which packs native code exploits that meant to run from a command line in non-standard locations in the application package. All such files are stored by encryption in the application package and may be decrypted at runtime.Certain malware such as DroidDream carries payload applications which are installed once the system has been compromised[12]. These payloads may also be stored encrypted. We have placed into payload and exploited encryption as DSA because signature based static detection could still be possible based on the main application's byte code.

7) Other Simple Transformations:

There are few other transformations as well that are specific to Android.Debugging information containing such as source file names, local,parameter variable names and source line numbers could be removed off. Moreover, non-code files and resources contained in Android packages can be renamed or modified.

8) Composite Transformations:

Any of the above transformations can be combined with one another to generate stronger obfuscations.While compositions are not independent of order, anti-malware detection results should be not committed to the order of application of transformations in all the cases.

C. Transformation Attacks Non-Detectable by Static Analysis(NSA)

These transformations break all kinds of static analysis.Some encoding or encryption is typically required so that no static analysis scheme can all infer parts of the code. Parts of the encryption keys may even be fetched remotely. In this scenario, interpretation or emulation of the code (i.e. dynamic analysis) could be still possible but static analysis becomes incapable.

1) Reflection:

The Java API reflection allows a program to invoke a method using the name of the methods. We can convert any method call into call to that method via reflection. This makes it difficult to analyze statically which method is being called.A consequent encrypted method name make it impracticable for any static analysis to recover the call.

2) Byte code Encryption:

It tries to make the code not available for static analysis.The applicable piece of the application code is stored in encoding form and is decipher at runtime via a decipherment routine. Code encoding has been used in polymorphic viruses; the only code which is available to signature based antivirus applications remains the decipherment routine which is typically clarification in different ways at each replication of the virus to avoid detection.

6. EXPECTED RESULT

In the work, we have actually focused on the evaluation of anti-malware products for Android. In a specific manner, we have attempted to deduce the kind of signatures that these products used to detect malware and how resistant that signatures are against transformation in the malware binaries. In this paper, we have analysed that android application which we need to install, firstly it check risk

score of malware attacks and then it takes the permission to download the application. If selected applications contain any type of malware or viruses attack then it does not download the application when the risk score is high instead it checks another application similar to it containing no malware having the least risk score.

7. CONCLUSIONS

In this paper, we analysed different anti-malwares which can be used for avoidance of different malware attacks. ADAM tool, complex mechanisms are used for privacy preserving but with fewer transformations malware detectors that use complex techniques requires pattern matching techniques. A framework based on DroidChameleon[1] uses more changes which are more accurate and efficient with anti-malware tools that can be found.It is necessary to protect the mobile device from malware.We stated a simple and high efficient technique for protecting the android devices from malware and finding the risk scores before downloading the apps from play store. This anti-malware application is important for not only measuring the risk scores of mobile malware threats but also propose effective, next generation solutions. We exercise DroidChameleon[1], a systematic framework with various transformation techniques. We have developed this application because in our research it is found that the existing anti malware products are fail to provide protection to common malware transformation techniques. Our results on various popular merchantile anti-malware applications for android are unreassuring none of these tools is tolerant against common malware transformation techniques. In addition, a majority of this can be trivially discomfited by applying slight transformation over known malware with little effort for malware authors. Finally, our results have proposed possible remedies for improving the current state of malware detection on mobile devices.

ACKNOWLEDGEMENT

We wish to express our sincere gratitude to Prof. C. K. Patil, Principal and Prof. H.D Sonawane, H.O.D of Computer Department for providing me an opportunity for presenting Paper on “Android Anti-malware Against Transformation Attacks”. We sincerely thank to our paper guide Prof. K. R. Wagh for her guidance and encouragement for completing the paper work. We wished to express our gratitude to the officials and especially our staff members who render their help during the period of our paper. Last but not least we wish to avail our self of this opportunity, express a sense of gratitude and love to our friends and our parents for their manual support, strength, help and for everything.

REFERENCES

[1] V. Rastogi, Y. Chen, and X. Jiang, *“DroidChameleon: Evaluating Android anti-malware against transformation attacks,”* in *Proc. ACMASIACCS*, May 2013, pp. 329–334.

[2] M. Zheng, P. Lee, and J. Lui, *“ADAM: An automatic and extensible platform to stress test Android anti-virus systems,”* in Proc. DIMVA, Jul. 2012, pp. 1–20.

[3] .C. Collberg, C. Thomborson, and D. Low, *“A taxonomy of obfuscating transformations,”* Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997.

[4] .M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, *“Semantics-aware malware detection,”* in Proc. IEEE Symp. Security Privacy, May 2005, pp. 32-46.

[5] C. Kolbitsch, P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, *“Effective and efficient malware detection at the end host,”* in Proc. 18th Conf. USENIX Security Symp., 2009,pp. 351–366..

[6] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, *“RiskRanker: Scalable and accurate zero-day android malware detection,”* in Proc.10th Int. Conf. Mobile Syst., Appl., Services, 2012, pp. 281–294.

[7] Y. Nadji, J. Giffin, and P. Traynor, *"Automated remote repair for mobile malware,"* in Proc. 27th Annu. Comput. Security Appl. Conf., 2011, pp. 413-422.

[8] V.Rastogi, Y.Chen, and W.Enck *,“AppsPlayground: Automatic security analysis of smartphone applications,”* in Proc. ACM CODASPY, Feb. 2013, pp. 209–220.

[9] 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 Proc. 19th Netw. Distrib. Syst. Security Symp., 2012, pp. 1–13.

[10] (2013, Dec. 3).ProGuard [Online].

[11]R. Komondoor and S. Horwitz, *"Semantics-preserving procedure extraction,"* in Proc. 27th ACM SIGPLAN-SIGACT Symp. POPL, 2000, pp. 155–169.

[12] Lookout, San Francisco, CA, USA. (2011). Update: Security Alert: DroidDream Malware Found in Official Android Market[Online].

[13] (2013, Dec. 3). Android.Basebridge—Symantec [Online].

[14] (2013, Dec. 3). Android.Geinimi—Symantec [Online].

[15] (2013, Dec.3). AndroidOS.FakePlayer—Symantec [Online].

[16] (2013, Dec. 3). Android.Bgserv—Symantec [Online].

[17] (2013, Dec. 3). Plankton [Online].

[18] Naïve Bayes Classification.