INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND CREATIVE ENGINEERING (ISSN:2045-8711)
VOL.11 NO.11 NOV 2021

A Survey on Scalability of source code analysis in malware detection using KOTLIN for Android Application

Dr.A.Edwin Robert

Assistant professor,Department of Computer Science

[1] edwinrobert.sms@cherancolleges.org

***Abstract*— Smart devices are built on a foundation of software components from a variety of vendors. While critical applications may undergo extensive testing and evaluation procedures, the heterogeneity of software sources threatens the integrity of the execution environment for android based application programs. For instance, if an attacker can combine an application exploit with privilege escalation vulnerability, the operating system (OS) can become corrupted. The importance of ensuring application integrity has been studied in existing work . In the proposed work stops the application once corruption is detected. Mandatory Access Control (MAC) in an Android based operating system to tackle malware problem is a grand challenge but also a promising approach. The firmest barriers to apply MAC to defeat malware programs are the incompatible ,unusable and more comples problems in an existing MAC systems. To address these issues, we manually analyse 2000 malware samples and component one by one and two types of MAC enforced operating systems, and then designed a novel Efficient Malware Detection and Tracer design (EMDT) using Hidden Markov model, which incorporates intrusion detection and tracing in an commercial mobile operating system which leverages efficient coding and authentication schemes. The proposed approach conceptually consists of three actions: detecting, tracing, and restricting suspected intruders .The novelty of the proposed work is that it leverages light-weight intrusion detection and tracing techniques to automate security label configuration that is widely acknowledged as a tough issue when applying a MAC system in practice. The other is that, rather than restricting information flow as a traditional MAC does, it traces intruders and restricts only their critical malware behaviours, where intruders represent processes and executables that are potential agents of a remote attacker. Our prototyping and experiments on Android operating system using Kotlin, which shows with minimized coding the Tracer can effectively defeat all malware samples tested via blocking malware behaviours while not causing a significant compatibility problem.**

Keywords*:* Intrusion, Tracing ,Malware, Kotlin, Android, Detection

I. INTRODUCTION

Malicious software (i.e., Malware) has resulted in one of the most severe computer security problems today. A network of hosts which are compromised by malware and controlled by attackers can cause a lot of damages to information systems. As a useful malware defense technology, Mandatory Access Control (MAC) works without relying on malware signatures and blocks malware behaviours before they cause security damage. Even if an intruder manages to breach other layers of defense, MAC is able to act as the last shelter to prevent the entire host from being compromised. However, as widely accepted [1], [2], [4], existing MAC mechanisms built in commercial operating systems (OS) often suffer from two problems which make general users reluctant to assume them. One problem is that a built-in MAC is incompatible with a lot of application software and thus interferes with their running [1], [2], [4], and the other problem is low usability, which makes it difficult to configure MAC properly [1]. Our observations are as follows: The incompatibility problem is introduced because the security labels of existing MACs are unable to distinguish between malicious and benign entities, which causes a huge number of false positives (FP) (i.e., treating benign operations as malicious) thus preventing many benign software from performing legal operations; the low-usability problem is introduced, because existing MACs are unable to automatically label the huge number of entities in OS and thus require tough configuration work at end users. With these investigation results, a novel MAC enforcement approach is proposed, EMDT, which uses a cross-platform, general purpose programming language Kotlin as an implementation tool to provide solution to the above problems. Since, Kotlin runs on JVM(Java virtual Machine) which is more compatible with java[16]. Deployment of web development and desktop applications can be made simple by its concise syntactic structure with the behaviour of structured concurrency. The EMDT approach consists of three actions: detection, tracing, and restriction. Each process or executable has two states, suspicious or benign. The contributions of this paper are as follows: 1. we introduce EMDT, a novel MAC enforcement approach which integrates intrusion detection and tracing techniques to disable malware on a commercial OS in a compatible and usable manner. 2. We have implemented EMDT to disable malware timely without need of malware signatures. 3. We investigate the root reason so find

compatibility and low usability problems of existing MACs. Although not all the observations are brand new, we believe that understanding these reasons more comprehensively and illustrating them through the design of an actual system are useful for other MAC researchers. The rest of the paper is organized as follows: Sections 2 presents the related research, Section 3 introduces in details our investigation and analysis on various behaviours of malware programs existing problems in MAC. Section 4 describes the design of EMDT and prototype approach. Section 5 discusses the approach from the perspectives of security, compatibility, and usability. Lastly, we conclude the work in Section 7.

II. RELATED WORK

DTE proposed by Lee Badger et al. [5] is a classical MAC model to confine process execution,

which groups processes and files into domains and types, respectively, and controls accesses between

S.No	Malware Type	Ranking based on behaviour	Harming Type
1.	Communicate with remote Host	1000	High
2.	Create executable File	950	High
3.	Modify register for start up	900	High
4.	Copy the Important Application and files	800	High
5.	Obtain System Information	750	High
6.	Inject into other Process	500	High
7.	Modify Executable file	450	High
8.	Create or modify OS series	400	High
9.	Change security setting	200	High
10	Add Unwanted Plug ins	100	High

TABLE 1.

Critical Malware Behaviour for Host based system

domains and types. Tracer can be regarded as a simplified DTE that has two domains (i.e., benign and suspicious) and four types (i.e., benign, read- protected, write-protected, and suspicious). Moreover,

Tracer can automatically configure the DTE attributes (i.e., domain and type) of processes and files under the support of intrusion detection and tracing so as to improve usability.PPI automates the generation of information flow policies by analyzing software package information and logs. MIC implements the no-write-up rule of classical BIBA model in Windows Vista kernel, but it does not implement the no-read-down rule in order not to compromise compatibility significantly. PRECIP [6] addresses several practical issues that are critical to contain spyware that intends to leak sensitive information.The risk-adaptive access control [10] that targets to make access control more dynamic so as to achieve a better tradeoff between risk and benefit. Most existing antimalware technologies are based on detection [7], [8]. Tracer tries to combine detection and access control so that it not only can detect but also can block malware behaviours before their harming security. Antimalware technology that resembles Proposed EMDT is behaviour blocking [9], which can confine the behaviours of certain adverse programs that are profiled in advance. Many commercial antimalware tools [10], [11] also have a behaviour-based module to defend against unknown malware programs.

A. Malware Investigation

Malware contribute to most Internet security problems. Antimalware companies typically receive thousands of new malware samples every day. An analyst generally attempts to understand the actions that each sample can perform, determines the type and severity of the threat that the sample constitutes, and then forms detection signatures and creates removal procedures. Symantec Threat Explorer [12] is such a publicly available database which stores the analysis results of thousands of malware samples from various sources and is thus valuable to malware researchers. To have a thorough understanding of the philosophies behind malware design, we have spent considerable amount of time analyzing the behaviours of malware programs. Specifically,we have read, recorded, and analyzed the technical details of 2,000 malware samples of a wide range of formats and varieties, such as viruses, worms, backdoors, root kits, and Trojan horses. We discovered few common characteristics of malware that can guide our subsequent antimalware design: Entrance characteristics. All malware samples break into hosts through two entrances, network and removable drive. Most breaking instances are via network through frequently used protocols such as HTTP and POP3.

B. Problems in MAC

Incompatibility is a well-known problem when enforcing a MAC modeling commercial operating system [1],[2],[4]. To investigate its root reason, in a secure network environment, we set up two machines to run MAC enforced mobile operating systems with MLS policy enabled and MAC module enabled. After a few days, we observed that these MAC systems produced a huge number of log records about denied accesses, which indicated that some applications failed and some acted abnormally. As the operation environment is secure without intrusion and

S.No	Malware	Benign Process			Suspicious Process		
		D	T	R	D	T	R
1	Remote Host	Nil	P	A	P	A	D
2	Exec File	Nil	P	A	P	A	D
3	Modify reg	Nil	P	A	P	A	D
4	Copy Application	Nil	P	A	P	A	D
5	Obtain System Info	Nil	P	A	P	A	D

D-Detected T-Traced R-Restricted

P-Possible A-Allow D-Deny

TABLE 2
Malware Classification based on Behaviour

malware, these denied accesses are thus "false positive." However, from the view of intrusion prevention, these processes do not necessarily represent intruders so that their "read" or "write" accesses to the /tmp should not be simply denied. Although it is possible to resolve this problem by adding "hiding sub directories" under /tmp, it is still difficult to eliminate the FPs resulting from many

other shared entities on an OS Relying on these labels, a MAC system often fails to make correct decisions on intrusion blocking which eventually results in many FPs. Low usability is another problem in a MAC-enabled system, as it often requires complicated configurations and unconventional ways of usage. In a modern OS, there are a wide range of entities including processes, files, directories, devices, pipes, signals, shared memories, sockets, etc.

III. PROPOSED SYSTEM

C. Efficient Malware Detection and Tracer(EMDT) In this section, we present our EMDT approach that aims to disable malware in a Android OS by disallowing malware behaviours. The adversaries of EMDT are malware programs that break into a host through the network or removable drives. As OS is the most popularly attractive to hackers, the description of EMDT is designed applying it to Android operating systems with some changes.

D.Overview

The design of an access control mechanism is to define the security label. We introduce a new form of security label called suspicious label for our EMDT approach. It has two values: suspicious and benign. Meanwhile, EMDT only assigns a suspicious label to a process or an executable, because a process is possibly the agent of an intruder and an executable determines the execution flow of a process which represents an intruder. When a process requests to access these entities, EMDT mainly utilizes their DAC information to make access control decisions, thus a huge amount of configuration work can be reduced while keeping traditional usage conventions unchanged.

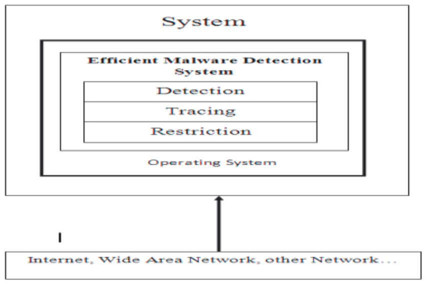

System
Efficient Malware Detection System
Detection
Tracing
Restriction
Operating System
Internet, Wide Area Network, other Network...

Figure 1: EMDT overview

Fig.1. gives an overview of EMDT which consists of three types of actions, detection, tracing, and restriction. Each process or executable has two states, suspicious and benign. The restriction action forbids a suspected intruder to perform malware behaviours in order to protect CIAP. That is to protect confidentiality, integrity and availability, as well as to stop malware propagation. The three actions work as follows: Once detecting a suspected process or executable, EMDT labels it as suspicious and traces its descendent and interacted processes, as well as its generated executables. EMDT does not restrict benign processes at all, and permits suspicious processes to run as long as possible but stops their malware behaviours that would cause security damages.

The object and parameter represent the target and parameter of the operation, respectively. Specific malware behaviours monitored in the current version of EMDT are listed in Table 1, which contains the 30 critical malware behaviours shown in Table 2. Moreover, EMDT allows dynamic addition of new behaviours. The access control decision of EMDT is made in accordance with normal MACs. EMDT uses the subject label and behaviour to make a decision while normal MACs use the subject label, operation, object label, and parameter. As a behaviour consists of operation, object, and parameter, EMDT actually uses the same four factors of normal MAC decision. Moreover, EMDT's decision procedure generates three possible access control results: "allow," "deny," and "change label," which resemble those of normal MACs. The detailed decision logic of Tracer is shown in Table 2. The detection and tracing actions lead to the decision result "change label," while restriction action leads to "deny." All access requests not denied are allowed. As an online approach, Tracer can produce the FP rate lower than that of behaviour-blocking mechanisms in commercial antivirus software. This is achieved by two means. First, as a MAC system, EMDT blocks a behaviour based simultaneously on the behaviour and security label (i.e., the suspicious label of the current process), rather than merely the behaviour as done by a behaviour-blocking system. Second, EMDT does not simply refuse all critical malware behaviours in Table 1.

1) Detecting Intruders

The detecting action is responsible for identifying all potential intruders. We do not intend to design a complex intrusion detection algorithm to achieve a low FP rate at the cost of heavy overhead. Instead, we design a light-weight intrusion detection algorithm that can identify all potential intruders but may have a relatively higher FP rate at the initial step. Tracing and restricting actions, will still allow it to run rather than stop it immediately, but only prevent it from executing featured malware behaviours. As depicted in Fig. 1, the detection works at two levels: entrance and interior.

$$D(P) = \begin{cases} \text{Benign} & \text{otherwise} \longrightarrow \qquad (1) \\ \text{Suspicious} & \text{if s€p} \end{cases}$$

Where D(P) is detection of process, signature s belongs to signature based, it comes in suspicious folder. The detection at entrance attempts to check all possible venues through which a malware program may break into the system. Dangerous protocols are permitted by firewalls, thus malware programs often use these protocols to penetrate firewalls by disguising themselves as popular software that generate benign network traffic. Dangerous protocols mainly include HTTP, POP3, IRC, SMTP, FTP, and ICMP. With these detection approaches enforced, however, two types of system maintenance tasks, i.e., updating software through the network and installing software from a removable drive, cannot be performed because the processes that perform these tasks are treated as suspicious. Hence, we provide two means to facilitate these system maintenance tasks. The Benign Process and Suspicious Process columns represent that the processes requesting the behaviours below are benign or suspicious, respectively. $\text{Label}_{\text{executable}}$ and $\text{Label}_{\text{process}}$ indicate changing the label of related process or executable to suspicious, respectively. Deny indicates denying the behaviour. This type of detection will not bring extra performance overhead since the restricting action of EMDT also needs to monitor such behaviours which will be presented in future.

2) Tracing Intruders

To track intruders within an operating system, one can use OS-level information flow as done in [13], [14]. However, a major challenge for leveraging OS level information flow to trace suspicious entities is that, file and process tagging usually leads the entire system to be floated with "suspicious" labels and thus incurs too many FPs. To address this issue, we propose the following two methods to limit the number of tagged files and processes in a single OS while preventing malware programs from evading the tracing as much as possible. For tagging files, unlike the approaches in [13], [14] and the schemes of many malware detection and MAC systems [2], [4], that trace information flow on OS level, Tracer only focuses on the tagging of executables while ignoring nonexecutables and directories. This is because an executable represents the possible execution flow of the process loading it, thus it should be deemed as an inactive intruder while a process is considered as an active intruder.

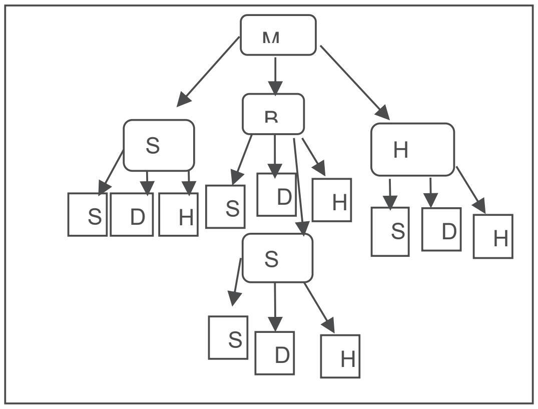


Fig 1.1. The mechanism to dynamically detecting the malware behaviours to OS

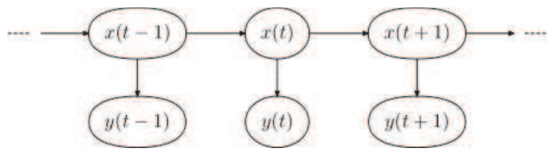
For tagging processes, we observed that the excessive number of tags mainly come from tracing Interposes Communication, i.e., marking a process as suspicious if it receives IPC data from a suspicious process. To address this issue, Tracer only tags a process receiving data from dangerous IPCs that can be exploited by a malware program to take control of the process to perform arbitrary malicious behaviours. Concretely, we analysed all vulnerabilities recorded in security bulletins related to named-pipes, local procedure calls, shared memories, mail slots, IPCs send free-formed data that can be crafted to exploit bugs in the receiving process by hidden morkov model.

The diagram below shows the general architecture of an instantiated HMM. Each oval shape represents a random process that can adopt any of a number of files. The random process(*t*) is the hidden state at time *t* (with the model from the above diagram ,$x(t) \in \{ x_1, x_2, x_3 \}$). The Malware, $y(t)$ is the observation at time *t* (with $y(t) \in \{ y_1, y_2, y_3, y_4 \}$). The arrows in the diagram denote conditional dependencies of the files .In the standard type of hidden Markov model considered here, the state space of the hidden variables is discrete, while the observations themselves can either be discrete (typically generated from a categorical distribution) or continuous (typically from a Gaussian distribution). The parameters of a hidden Markov model are of two types, *transition probabilities* and *emission probabilities* (also known as *output probabilities*). The transition probabilities control the way the hidden state at time t is chosen given the hidden state at time $t-1$. The hidden state space is assumed to consist of one of N possible values, modeled as a categorical distribution. (See the section below on extensions for other possibilities.) This means that for each of the N possible states that a hidden variable at time t can be in, there is a transition probability from this state to each of the N possible states of the hidden variable at time $t+1$, for a total of N^2 transition probabilities. Note that the set of transition probabilities for transitions from any given state must sum to 1. Thus, the $N \times N$ matrix of transition probabilities is aMarkov matrix. Because any one transition probability can be determined once the others are known, there are a total of $N(N-1)$ transition parameters.

In addition, for each of the N possible states, there is a set of emission probabilities governing the distribution of the observed variable at a particular time given the state of the hidden variable at that time. The size of this set depends on the nature of the observed process of malware. For example, if the observed malware is discrete with M possible values, governed by a signature based system distribution , there will be $M-1$ separate parameters, for a total of $N(M-1)$ emission of hidden malware in the system over all hidden states in the code or process. On the other hand, if the observed malware is an M-dimensional vector distributed according to an arbitrary multivariate Gaussian distribution, there will be M parameters controlling the means and $M(M+1)/2$ parameters controlling the covariance matrix, for a total of

$$N(M+\frac{M(M+1)}{2}) = NM(M+3)/2 = O(NM^2)$$

emission parameters. (In such a case, unless the value of M is small, it may be more practical to restrict the nature of the covariances between individual elements of the observation vector, e.g. by assuming that the elements are independent of each other, or less restrictively, are independent of all but a fixed number of adjacent elements.)



3) Probability of an observed sequence of malware

The task is to compute, given the parameters of the model, the probability of a particular output sequence. This requires summation over all possible state sequences:

The probability of observing a sequence

$$Y = y(0), y(1), \dots, y(L-1)$$ of length *L* is given by

$$P(Y) = \sum_X P(Y \mid X)P(X),$$

Where the sum runs over all possible hidden-node sequences

$$X = x(0), x(1), \dots, x(L-1).$$

Applying the principle of dynamic programming, this problem, too, can be handled efficiently using the forward algorithm.Probability of the latent files in Software's: A number of related tasks ask about the probability of one or more of the latent files, given the model's parameters and a sequence of observations $y(1), \dots, y(t)$.

4) Filtering

The task is to compute, given the model's parameters and a sequence of observations, the distribution over hidden states of the last latent contents at the end of the sequence, i.e. to compute $P(x(t) \mid y(1), \dots, y(t))$. This task is normally used when the sequence of latent variables is thought of as the underlying states that a process moves through at a sequence of points of time, with corresponding observations at each point in time. Then, it is natural to ask about the state of the process at the end. The result reveals that in reality it is quite difficult to propagate malware through local IPCs within a OS .Consequently, Tracer employs a Dangerous-IPC-List to record and trace each type of dangerous IPC since there should be a very limited number of dangerous IPCs in a OS. Therefore, we have the following tracing rules to mark entities as suspicious:

1. A process spawned by a suspicious process
2. An executable or semi executable created or modified by a suspicious process
3. A process loading an executable with a suspicious label
4. A process receiving data from a suspicious process through a dangerous IPC; and A process reading a semi executable or script file with a suspicious label.

A script file is written in interpreting language, e.g., JavaScript or VBScript, and thus needs execution engine, e.g., wscript.exe or cscript.exe, to load and run it. Accordingly, to defend against a script virus, Tracer should restrict the engine processes that are reading and interpreting a suspicious script file. On the other hand, a semi executable represents certain types of data files that might contain executable codes, which mainly involves various types of compressed files and Microsoft Office documents.

5) *Restricting Intruders*

In order to disable malware programs on a host, the restricting action monitors and blocks intruders' requests for executing critical malware behaviours listed in Table1. To follow the principle of complete mediation for building a security protection system, Tracer further restricts two extensive behaviours, called generic malware behaviours, to protect security more widely. The first one is "Steal confidential information," which represents all illegal reading of confidential information from files and registry entries. The other is "Damage system integrity," which represents all illegal modifications of the files and registry entries that require preserving integrity. In addition, other behaviours that can be used to bypass Tracer mechanism also need to be monitored and restricted, including "Change file attributes," "Change registry entry attributes," "Execute non executable files," and "Execute Tracer special system calls." The behaviour "Change file attributes" represents changing file extension names to executable or changing file DAC information. All behaviours restricted are listed on the column "restrict" in Table 2. In summary, the restricting action consists of three rules:

1. Restricting critical malware behaviours,
2. Restricting generic malware behaviours,
3. Restricting behaviours bypassing Tracer.

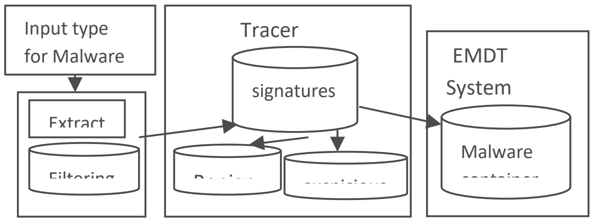


Fig.1.2. Dynamically restricting and detecting the malware behaviours using EMDT process

by mediating all these behaviours, Tracer is able to preserve system security and prevent a malware program from propagating itself in the system. To be specific, confidentiality is mainly achieved by blocking the generic behaviour "Steal confidential information;" integrity is mainly protected by blocking the generic behaviour "Damage system integrity;" availability is defended by blocking the behaviours listed in Table1 with the capital letter A attached; propagation is prevented by blocking the behaviours in Table1 with the capital letter P attached. Meanwhile, blocking these behaviours can help to defend against unknown malware programs for two reasons. First, these behaviours are extracted from thousands of malware samples and thus represent popular hacking techniques that are often used in unknown malware programs by malware authors. To efficiently restrict these malware behaviours, two issues need to be addressed. The first is how to determine the generic malware behaviours. We identify behaviours "Steal confidential information" and "Damage system integrity" by monitoring illegal reading on read-protected objects and illegal writing on write-protected objects, respectively. However, it is difficult to identify the objects that need protection among a large number of candidates in a OS in order to recognize the generic malware behaviours

```
Algorithm1: Monitoring the Application Process :

Input: File to be read , Buffer reader

Process:

If (File!= Copying Behaviour)||(Current Process==Benign)

Return Operation To Buffer

For (Node of file =Read list of Buffer)

If(File==Node)

Statement: Attach the File in the Buffer reader

Else

Statement: copy the File into Node(Stack) for Blocking

          Then

           Copy the file into buffer

Return (permit the File to monitor)
```

The algorithms may impose a relatively high overhead only on the malware processes that frequently exhibit file copying behaviours but not on benign processes and the suspected processes that are actually benign. The algorithms only need to monitor the file-copying behaviour “Copy itself” by watching the read operation on the process’ image file. However, in reality a benign process rarely tries to copy itself, thus the read-list is often empty and the algorithms do not need to do anything. It only needs to read an executable in two situations while such read operations do not need to be recorded into the read-list.

Algorithm for detection that correlate read and writes operations by comparing buffer contents are more difficult to be circumvented than other candidate algorithms, e.g., comparing buffer addresses. In the worst case that a malware program successfully circumvents the algorithms, EMDT still can tail it by monitoring related behaviours, e.g., “Create executables,” since file-copying behaviours need to create executables.

S. No	**Malware Type**	**Migrating Channels**	**Behaviour detected by Directed**	**Behaviour Detected By EMDT**
1	Worm	Website	Copy, Modify the Application content	Copy, Modify the Application content
2	Trojan	Website	Copy, Modify the Application content	Copy, Modify the Application content
3	P2P worm	Communication protocol or Channel	Damage system Integrity	Damage system Integrity
4	Root Kit	Removable Drive like pen drive or CD	Corrupt or modify driver softwares	Corrupt or modify driver softwares
5	Back Door	FTP	Driver software corruption	Driver software corruption

Table 3: EMDT detected Malware Types

6) Dynamic changes of Malware Behaviours detection Process:

If the detection is purely based on known malware characteristics and behaviours, a detector may not be able to function effectively in the long run as new malware characteristics and behaviours may emerge over the time. To address this limitation [15], a novel extensible mechanism is implemented in EMDT so it can dynamically add in new behaviours to monitor. A behaviour consists of operation, object, and parameter. For example, the operation create_ file corresponds to two system calls: NtOpenFile and NtCreateFile. In contrast, a single system call may contain more than one operation.

In each concerned system call, we set up one or more checkpoints, each of which is responsible for checking the behaviours belonging to the same operation with the support of a modifiable behaviour list in memory. The new malware behaviours are read from a configuration file and distributed to proper behaviour lists corresponding to different operations in memory. At each checkpoint, Tracer searches for the object and parameter currently requested in the corresponding list to determine whether the current access forms a malware behaviour.

7) Evaluation Results

Table 3 describes the detailed test results of 7 selected malware samples. We can see that all the malware samples are successfully disabled via the restriction of their malware behaviours. For example, the worm "Worm." downloaded from the local website has the following main steps for function: it first copies itself, i.e., regsv.exe, to hard drive in OS, then runs regsv.exe as a new process, the new process then adds a value under registry key regsv.exe so that it can be launched when the system restarts, finally listens at port 113 to accept commands from a remote attacker. On a host without EMDT enabled, all above steps are successfully executed. However, after activating the EMDT protection, the malware behaviour "Copy itself" is blocked, i.e., the malware cannot create a new copy of itself in the system folder. Consequently, the rest of the behaviours do not appear anymore because these behaviours depend on the new process launched from the malware's copy. In other words, the worm is disabled.

8) Performance Measures Detection rate using Kotlin

```
Algorithm2: Detecting the Malware Process :

Input: File to be read , Buffer writer

Process:

If (File != Copying Behaviour)||(Current Process==suspicious)

Return Operation To Buffer

For (Node of file =Read list of Buffer)

If(File==Node)

Statement: Attach the File in the Buffer writer

Else

Statement:

 Blocking file from Corruption Then

Copy the malware type into buffer writer

Return (Malware type to buffer)
```

It shows the EMDT system detection rate in percentages with directed graph technique for multiple OS systems. On an average, 85% of malware types in the host (OS) can be effectively determined by proposed system. [16] In processing of EMDT using Kotlin running in source code of the application 40% of coding is reduced and hence the system detects the malware with signature based system and machine learning based system in which it considerably reduces the false positive rate.

9) Calculation for False positive

False positive rate = 1- (No .of True Positive / (No. of True positives + No. of false Negatives))

The modules that block suspicious behaviours contribute to most of FPs of the antimalware tools. The fundamental reason is that the antimalware tools identify a suspicious behaviour only based on the behaviour itself while Tracer further considers the suspicious label of the process requesting the behaviour.

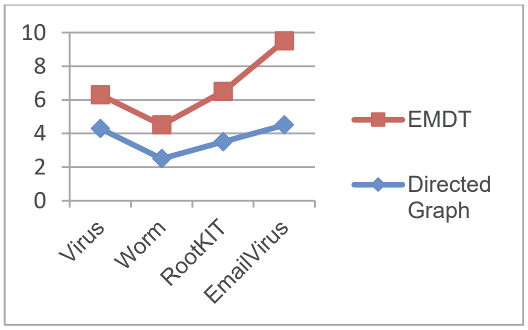


Figure 5: Detection Rate of Virus through EMDT and Directed graph

False Positive Measures

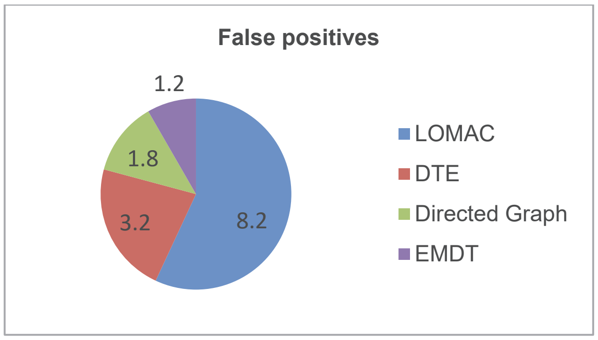


Figure.6: Comparing the false positive rate of EMDT with other existing technique

Conclusion and Future enhancement

In this paper, we propose a novel MAC enforcement approach that integrates intrusion detection and tracing to defend against malware in an android OS developed by using Kotlin as a source code. We have extracted 30 critical malware behaviours and three common malware characteristics for the incompatibility and low usability problems in MAC, which will benefit other researchers in this area. Based on these studies, we propose a novel MAC enforcement approach, called EMDT using Hidden markov model, to disable malware timely without need of malware signatures or other knowledge in advance. The novelty of Tracer design is two-fold. One is to use intrusion detection and tracing to automatically configure security labels. The other is to trace and restrict suspected intruders instead of information flows as done by traditional MAC schemes. EMDT system doesn't restrict the suspected intruders right away but allows them to run as long as possible except blocking their critical malware behaviours. This design produces a MAC system with good compatibility and usability. We have implemented Tracer in several OS and the evaluation results show that it can successfully defend against a set of real-world malware programs, including unknown malware programs, with much lower FP rate than that of commercial antimalware techniques by using Kotlin as a implementation tool which has a unique feature of reducing syntactic structure of the source code[16].In future we are going to launch this work for a large cloud server runs the application front-end logic and data are outsourced to a database or file server where there is increase in application and data complexity.

REFERENCES

[1]. N. Li, Z. Mao, and H. Chen, “Usable Mandatory Integrity Protection for Operating Systems,” Proc. IEEE Symp. Security and Privacy (SP ’07), pp. 164-178, 2007.

[2] T. Fraser, “LOMAC: Low Water-Mark Integrity Protection for COTS Environments,” Proc. IEEE Symp. Security and Privacy (SP ’00), pp. 230-245, 2000.

[3] Microsoft, Mandatory Integrity Control, http://en.wikipedia.org/ wiki/ Mandatory_Integrity_Control, 2012.

[4] X. Wang, Z. Li, J.Y. Choi, and N. Li, “PRECIP: Towards Practical and Retrofittable Confidential Information Protection,” Proc. 15th Network and Distributed System Security Symp., 2008.

[5] L. Badger, D.F. Sterne, D.L. Sherman, K.M. Walker, and S.A. Haghighat, “Practical Domain and Type Enforcement for UNIX,” Proc. IEEE Symp. Security and Privacy (S&P), pp. 66-77, 1995.

[6] X. Wang, Z. Li, J.Y. Choi, and N. Li, “PRECIP: Towards Practical and Retrofittable Confidential Information Protection,” Proc. 15th Network and Distributed System Security Symp., 2008.

[7] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R.A. Kemmerer, “Behaviour-Based Spyware Detection,” Proc. 15th Conf. USENIX Security Symp. (USENIX-SS ’06), 2006.

[8] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J.C. Mitchell, “A Layered Architecture for Detecting Malicious Behaviours,” Proc. 11th Int’l Symp. Recent Advances in Intrusion Detection, pp. 78-97, 2008.

[9] C. Nachenberg, “Behaviour Blocking: The Next Step in Anti-Virus Protection,” http://www.securityfocus.com/infocus/1557, Mar. 2002.

[10] Kaspersky Lab, http://www.kaspersky.com/, 2012.

[11] Vipre, Inc., http://www.vipre.com/vipre/, 2012.

[12] http://www.symantec.com/business/security_ response/ threat explorer/threats.jsp, 2012.

[13] S.T. King and P.M. Chen, “Backtracking Intrusions,” Proc. 19th ACM Symp. Operating Systems Principles (SOSP ’03), pp. 223-236, 2003.

[14] A. Goel, K. Po, K. Farhadi, Z. Li, and E. Lara, “The Taser Intrusion Recovery System,” Proc. 20th ACM Symp. Operating Systems Principles (SOSP ’05), pp. 163-176, 2005.

[15] Z. Shan, X. Wang, and T. Chiueh, “Tracer: Enforcing Mandatory Access Control in Commodity OS with the Support of Light- Weight Intrusion Detection and Tracing,” Proc. Sixth ACM Symp. Information, Computer and Comm. Security, pp. 135-144, Mar. 2011.

[16] LucaArdito,Riccardo Coppola, Giovanni Malnati, Marco Torchiano ,”Effectiveness of Kotlin vs. Java in android app development tasks”, Journal of Information and Software Technology (ElseVier),Nov 2020,Vol 127,106374

ACKNOWLEDGEMENT

Dr.A.EdwinRobert, MCA.,M.Phil.,Ph.D in Computer Science. He is working as an Associate professor in Computer Science at SMS College of Arts and Science, Coimbatore, Tamilnadu,India. He had 13 years of teaching experience. He has presented some papers in International Conferences. His area of research is Software Engineering and Security.