International Journal of Engineering & Technology, 7 (4.15) (2018) 113-119

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A Preemptive Behaviour-based Malware Detection through Analysis of API Calls Sequence Inspired by Human Immune System

Fadzli Marhusin[1,2]*, Christopher John Lokan[3]

[1]*Faculty of Science and Technology (FST), Universiti Sains Islam Malaysia (USIM), Bandar Baru Nilai, 71800 Nilai, Negeri Sembilan, Malaysia*

[2]*CyberSecurity and Systems Research Unit, Islamic Science Institute (ISI), Universiti Sains Islam Malaysia (USIM), Bandar Baru Nilai, 71800 Nilai, Negeri Sembilan, Malaysia*

[3]*The University of New South Wales (UNSW), Canberra, Australia*

**Corresponding author E-mail: fadzli@usim.edu.my*

Abstract

This study detects malware as it begins to execute and propose a data mining approach for malware detection using sequences of API calls in a Windows environment. We begin with some background of the study and the influence of Human Immune System in our detection mechanism, i.e. the Natural Killer (NK) and Suppressor (S) Cells. We apply the K = 10 crosses fold data validation against the dataset. We use the n-grams technique to form the data for the purpose of establishing the Knowledge Bases and for the detection stage. The detection algorithm integrates the NK and S to work in unison and statistically determine on whether a particular executable deemed as benign or malicious. The results show that we could preemptively detect malware and benign programs at the very early beginning of their execution upon inspecting the first few hundreds of the targeted API Calls. Depending on the speed of the processor and the ongoing running processes, this could just happen in a split of a second or a few. This research is as part of our initiative to build a behaviour based component of a cyber defence and this will enhance our readiness to combat zero-day attacks.

Keywords: *API Calls based Detection; K-grams; Malware; N-grams.*

1. Introduction

Malware is software code that has malicious intent, can only do harm if it is allowed to execute without being detected. In recent years, there have been huge changes in the threat landscape and it is a major cybersecurity threat. Malware writer working day and night. Every day, there is a huge volume of brand new and unique malware coming up. Among the most difficult and challenging tasks for malware detection researchers are to detect and eliminate evasive malware that use state of the art technology, i.e.; encryption, encrypted payload, ransom facilities, social engineering techniques, for exploitation objectives. Antivirus companies detect large number of them. However, some of the malware can be identified and some others unable to be detected. For many of us, we familiar with most of malware types, but there are still brand new and novel ones emerging from times to times.

In this paper, we explain the technique we use to prepare our experiment from the context of real-time detection process. This shall include on how we capture the raw data, select features, transform, establish and updates Knowledge Bases and eventually do the detection.

2. Background of the Study

Attacks may vary from the individual or organisational level, to nation-states resorting to cyber warfare to infiltrate and sabotage enemies' operation. Hence, there is an urgent need for a dependable cyber defense. The professional malware writer has every opportunity to test against all commercial based on signature based detectors. Malware detection is hard when the latest malware employs some protection and evasion techniques – thus creating a zero-day attack situation. This happens because none of the signature based detectors in the market possesses knowledge about the attack. Thus, behaviour-based detection is needed to detect zero-day malware attacks.

In API calls research, it is believed that malware generates sequences of API calls that are different from benign API calls, and involve sequences of actions such as create, read, write, delete and change files, and directories or special resources of the OS. Most malware is also capable of communicating with other hosts (although this is not always necessary) for replication attempts, accessing resources from other hosts and sending information to others. Some other types of malware such as worms, utilise a special facility in a certain OS so that, upon logon, the malware automatically run again, thereby disallowing the end-user from making any attempt to remove them, as exhibited by the *Brontok* worm [1].

A seminal work by [2] fragmented long system calls of UNIX processes into shorter system call signatures. They used a sequence length of 10 based on their empirical observation of the unique n-gram sequences. When deciding the size of the *n*-gram, they raised two issues: (1) if the size of n is large, the size of the storage database will also be large; and (2) if the size of n is too

small, it could be very difficult to discriminate between benign software and malware.
In [3] enhanced the above technique by proposing a detection using variable-length sequences of API calls as an alternative to fixed-length sequences.

3. The influence of Human Immune System (HIS)

In this section, we briefly describe the mechanisms of the HIS which have inspired many researchers to adopt similar characteristics in computer defence [2, 4-10]. The death of human cells may undergo one of these 1) Lysis 2) Apoptosis 3) Necrosis. The lysis is a stage where a cell might be just repaired. The apoptosis is a stage whereby a cell will undergo a dismantle process and the necrosis is the one where a cell is destroyed due to external factors such as pathogen.
In other context of HIS, B cells are white blood cells that play a large role in the humoral immune response, whereas T cells have roles in the cell-mediated immune response, also known as the innate immune system. The major task performed by B cells is to make antibodies. A variety of T cells, Natural Killer (NK) cells, recognises a pathogen when the cell's Major Histocompatibility Complex (MHC) shown on its surface is detected as non-self. Damaged or infected cells tend to show unusual levels of MHC. NK cells are cell killers activated when they receive one of the following signals:

- Cytokines: A stressed cell may release uric acid to inhibit a pathogen that is entering through its cell wall. NK cells detect this acid and respond against pathogens situated in the surrounding area of the cell.
- FC-Receptor: At the site of infection, a large number of white cells engulf pathogens and repair infected cells.
- Activating and inhibitory receptors: NK cells have receptors that connect to nearby cells to regulate their destructive activities.

Pioneering works by [2] used the nature of peptides to allow the recognition of self and non-self by using the input vector as analogous to the peptide. Using the negative selection algorithm, there are two stages involved: generation and detection. In the first, a normal profile is recorded with the assumption that there are no intrusive activities. Once the normal profile is sufficiently developed, a raw vector is passed to a process with the aim of matching it to the self-sample. If there are any matching signatures, the ones in the self-sample are discarded and the remaining vectors (abnormal) are passed to a detector. In the detection stage, the detector compares the recorded attack vectors with the incoming vectors and any matching pattern is considered anomalous.
There are a number of other research studies, including those of [7-8] which have attempted to explore ideas of mapping between malicious code detection and the Danger Theory [10]. The Danger Theory [11-12] involves algorithms such as the dendritic cells (DCs) [7] and Toll-like Receptor [13]. In [7], the authors adopted the functionality of DCs. Firstly, the DCs forward a collected protein (antigen) together with its environmental context to the effector's T cells. When passed to the lymph node, a DC displays an antigen with context signals, and T cells that have a complementary receptor for the antigen are activated for immunisation. If a cell is stressed because danger is present within a particular tissue, nearby DCs will produce inflammatory cytokine. Then, the cell will undergo lysis or apoptosis. Additionally, the authors included the idea of pattern recognition receptors that are available on DCs and can detect certain well-known pathogens, such as bacteria, that have particular proteins called pathogen-associated molecular patterns (PAMPs) which are learnt over a long time. Mature DCs activate the immune response and semi-mature DCs suppress it. These activation and suppression processes regulate and balance the immune response activity.
In [7], the PAMP can be assumed as a security policy violation. The Safe Signal is the same as normal behaviour whereas the Danger Signal is equivalent to a harmful symptom, such as a sharp spike in memory or Control Processing Unit (CPU) processes. Cytokine is equivalent to a system's load average that can change as a result of one or more symptoms. An antigen is regarded as an exploited system call.
In computer security, a better understanding and discovery of mechanism in HIS is important and a way forward to formulate new strategies towards a more sophisticated control of malware attacks.

4. The Architecture of the Detection System and Methodology

The method of malware detection in our study is via real-time monitoring of the API call sequences as execution begin. Our approach is inspired by the nature of Human Immune System (HIS) theories. The detection algorithm is mainly inspired by the role of the Natural Killer (NK) and Suppressor (S). The algorithm decides by relying upon Knowledge Based (KB) that evolve over time. The Negative Selection algorithm is used to produce several optimal models of KB and find the most optimal, given several constraints. The KB, built via recent based data, is a collection of the behaviour of the old malware and benign profiles. KB is used to detect zero-day malware and good programs. Sequences of API calls are analysed in n-grams which are compared with KB. A decision is made based on a statistical measure, indicating similarity represented in the n-grams against each of the profiles in the KB. Findings of the research is fundamental for a solid insight and capability to combat any zero-day attacks.

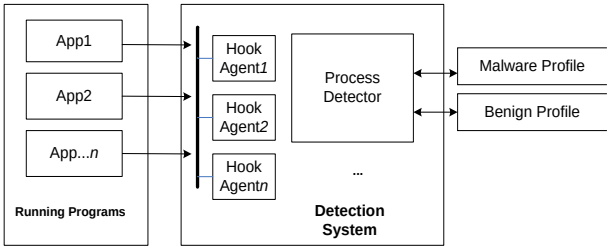

Fig. 1: Architecture of the detection system

Figure 1 illustrates the general components in our detection architecture. The detection system contains detectors and one of it is the Process Detector. The running programs are subjected to monitoring by the process detector. As a new program launched, a hook agent is assigned to the process, capturing all the relevant API Calls for the purpose of detection. The detector will hook itself with the executable with the aim of finding anomalies in the API call sequences of each program. Hooking of this API Calls in this way are possible via the support of *Deviare* framework [14] or *Detours* software package [15]. This two software can quickly assist developers and researchers to intercept API calls of Windows' OSs. This software could be used to collect and establish benign and malware profiles which will be used by the Process Detector. There is also *apimonitor* software program [16] that can be used to gather API Calls in a controlled environment.
An existing dataset is used, available at [17], to enable comparison with the results published in the work of Ahmed et al. [18] who used the same dataset. This dataset consists of sequences of API calls from 98 benign programs and 416 malware executables. The latter comprise 117 Trojans, 165 viruses and 134 worms, and include a number of malware that make use of obfuscation techniques. Based on our checking of online databases describing malware [19-21], some malware implement polymorphism or encryption engines, e.g., Virus.Win32.Alman.a, Virus.Win32.Dream.4916, Virus.Win32.Crypto, Virus.Win32.Chop.3808 and Virus.Win32.Aris, and others packing

and unpacking engines, e.g., Worm.Win32.Lioten, Trojan.Win32.AVKill.a, Trojan.Win32.AntiNOD.b, Trojan.Win32.Ajim, Mytob and Zotob. We believe that the inclusion of these malware will provide some insight into the capability of our detection system to fight evasion attacks. As described by the original authors, this malware collection was obtained from [22], and proprietary software [16] was used to record the API calls of its benign programs and malware executables. The sizes of the executables used varied: in the benign category, the minimum and maximum were 4KB and 104588KB, respectively, and the average 1263KB; whereas in the malware category, the maximums of the Trojan, virus and worm types were comparatively small, being only 9,277KB, 5,832KB and 1,301KB, respectively, with an average of around 266.7KB and a minimum of approximately 2.7KB. Apparently, benign programs generate longer API sequences than malware. Below is an example of API call sequences generated by the Trojan horse Win32.Bancos.j.apm:

"GlobalFree, RegOpenKeyExA, RegOpenKeyExW, HeapAlloc, HeapFree, RegQueryValueExW, HeapAlloc, HeapFree, RegCloseKey, GlobalSize…".

We examined the dataset and API classes on the Microsoft Developer Network (MSDN) website [23] and noted that the APIs appearing in the dataset fall into the 16 classes:

Registry, Network, Network Share Management, Windows Networking Functions, Memory Management, Windows Native System Services Routine (Windows Driver Kit), File Management, Directory Management, Volume Management,

Disk Management, Large Integer Functions, Winsock, Winsock Service Provider Interface (Winsock SPI), Process and Thread, Process Status API (PSAPI) Function and lastly Dynamic Link Libraries.

We grouped them into seven classes: 1) registry; 2) network; 3) memory; 4) file directories and special functions; 5) socket; 6) process and thread; and 7) dynamic link libraries, as listed in Table 1. Our exploratory analysis shows that there is a total of 237 unique API calls generated by benign programs and malware executables, of which the benign use only 166 and the malware 195. There are 71 API calls used only by malware executables of which 33 (46%) appear in all malware classes while 12.6%, 11.2% and 19.7% appear exclusively in Trojan, virus and worm, respectively, as listed in Table 2.

Table 1: API classes in [23] evaluated in study.

Class ID	API Function/Routine Classes
1	Registry
2	Network, Network Share Management and Windows Networking Functions
3	Memory Management
4	File Management, Directory Management, Volume Management, Disk Management and Large Integer Functions
5	Socket (Winsock and Winsock SPI)
6	Process and Thread, and Process Status API Functions
7	Dynamic Link Libraries

Table 2: Statistics of targeted API calls invoked in the dataset

Unique API Calls Category	Total
Total API Call category invoked	237
Total uniquely invoked in benign	166
Total uniquely invoked in malware	195
Shared in malware and benign	124
Exclusively invoked in benign	42
Exclusively invoked in malware	71
Mutually invoked in Trojan, virus and worm but not benign	33
Total uniquely invoked in Trojan	159
Total uniquely invoked in virus	147
Total uniquely invoked in worm	140
Exclusively invoked in Trojan	9
Exclusively invoked in virus	8
Exclusively invoked in worm	14

Table 3: Shared or exclusive API call sequences in benign and malware

Category	Benign	Trojan	Virus	Worm
Total executables	98	117	165	134
Total calls	2,210,786	635,989	612,808	43,3554
Appear only in benign	2061	N/A	N/A	N/A
Appear in benign & malware	2,208,725	599,533	603,124	42,6604
Appear in malware but not in benign	N/A	3,6456	9,684	6,950

The above-mentioned APIs are based on only unrepeated figures. A real API call sequence involves a mixed invocation of repetitive functions starting from the initial execution of the executable until it stops. Table 3 shows the total numbers of appearances of API calls in the dataset. Although there are fewer benign programs than malware executables, they carry 56.8% of the total API call invocations, followed by Trojan (16.3%), virus (15.7%) and worm (11.1%). The ratios of malware calls appearing exclusively in Trojan, virus and worm to the total malware calls are very small, being 0.03%, 0.01% and 0.02%, respectively.

The benign programs in the dataset generate relatively more API calls than malware. The APIs used by benign, malware and both programs are shown in Table 3 in which it can be seen that a small proportion of the total APIs, 0.09%, is used exclusively by benign programs' executions. Trojan, virus and worm types use APIs which do not exist in any benign applications and, regardless of whether the APIs exist exclusively in each malware category, we identify 5.7% of them in Trojan, 1.58% in virus and 1.6% in worm. In summary, to discriminate benign from malware programs is challenging because a large number of APIs are used by both, that is, 99.91% in benign, 94.27% in Trojan, 98.42% in virus and 98.4% in worm. Based on this information, we expect that viruses will contain more n-grams similar to benign, followed by worms and Trojans.

4.1. Features Selection and Data Reduction

An API call sequence contains a number of features depending on the names of its functions. A sequence of API calls captured using the *apimonitor* [16] tool can contain comprehensive information, such as the executable's profile, and the function's name and its associated parameters. Although spatial and temporal information could be retrieved from a collection of API call sequences, using too many features will usually involve more complex detection algorithms in order to associate them and produce aggregated data. Therefore, we use only function names as a feature for detection purposes.

We investigate the need to ignore certain API classes in the dataset and find that some which appear in benign, malware or both seem to have high concentrations of one type, as can be observed from the APIs in Classes 3, 4 and 7 shown in Table 4.

As too many API calls are invoked from Memory Management, with many at a high frequency and yet co-existing in high proportions in benign and malware, we propose that memory-related API calls not be used as a source of data due to their high frequency of variable declaration, invocation and re-invocation in modern programs. Based on this, and further evidence provided in the next section, we remove all memory-related API calls in the dataset.

It is noted that the benign data does not use any API calls from Class 4 which is evidence that benign API calls are only partially collected, perhaps only those of the executables captured from the initial execution up to a certain time or condition (i.e., when the GUI is ready) and does not include those generated when an end-user started interacting with the program.

Table 4: Insight into reduction process for API classes

No.	API Function/Routine Classes	Appear in Benign	Appear in Both	Appear in Malware
1	Registry	15	34	n/a
2	Network, Network	1	4	10

No.	API Function/Routine Classes	Appear in Benign	Appear in Both	Appear in Malware
	Share Management and Windows Networking Functions			
3	Memory Management	1	33	1
4	File Management, Directory Management, Volume Management, Disk Management and Large Integer Functions	n/a	n/a	50
5	Socket (Winsock and Winsock SPI)	20	16	6
6	Process and Thread, and Process Status API Functions	5	32	4
7	Dynamic Link Libraries	n/a	5	n/a

4.2. The N-Grams

An *n*-gram is a technique used in data mining [24-25] can be defined as a sub-sequence of n items from a given stream or sequence of data which can come from various sources, such as text, graphic, audio and video. Concerning the dataset used in this paper, the term sequence or stream refers to the API calls invoked by a running executable and an item refers to any API functions within chosen classes of API in Windows Platform. Hence, an *n*-gram of an API call sequence refers to a sequence of functions the size of which is subject to the value of *n*. Usually, *n*-grams are non-overlapping sequences of items but can be designed to be overlapping. We apply non-overlapping sequences of API calls; for example, a string of API call sequences generated by a Trojan horse named *Win32.Bancos.j.apm* is:

"GlobalFree, RegOpenKeyExA, RegOpenKeyExW, HeapAlloc, HeapFree, RegQueryValueExW, HeapAlloc, HeapFree, RegCloseKey, GlobalSize …".

If the size of $n = 5$, it can be transformed into:

n-gram1 = GlobalFree, RegOpenKeyExA, RegOpenKeyExW, HeapAlloc, HeapFree

n-gram 2 = RegQueryValueExW, HeapAlloc, HeapFree, RegCloseKey, GlobalSize

Further, these *n*-grams can be transformed into the simpler format of:

n-gram 1 = 1,2,3,4,5

n-gram 2 = 6,4,5,7,8

Determining actual *n*-gram sizes is very important as two issues arise, as highlighted in [2, 26]. If *n* is small, the *n*-gram sequences will find it difficult to discriminate between benign and malware. Inversely, a large *n* will create a large number of unique sequences as they form a larger combinational matrix of APIs.

Table 5: Unique *n*-grams of benign vs malware with removal of Memory Management Class APIs

n Size	Benign	Malware (Redundant)	Malware (Unique)
1	132	161	70
2	1237	1850	1188
3	4460	5173	3385
4	9477	8434	5651
5	14220	10423	7552
6	17530	11071	8621
7	19080	11116	9135
8	19408	10819	9209
9	19423	10568	9190
10	18854	10104	8929

We investigate this issue and evaluate *n* sizes of 1 to 10. As shown in Table 5 it appears that the best setting for *n* is when $n = 5$ as any *n* values greater than five will have lesser reduction rates but larger *n*-gram cardinality. Therefore, we set $n = 5$ in this experiment which means that there are five function names for every *n*-gram.

The dataset contains 48472 benign and 19732 malware *n*-grams when $n = 5$. Removing the Memory Management class greatly reduce the numbers to only 14220 benign and 7552 malware *n*-grams, a reduction of 70.66% and 61.73% respectively from their original numbers. Table 5 shows the number of *n*-grams in the benign and malware profiles based on all the 10-fold data for *n*-gram sizes of 1 to 10, without the Memory Management class APIs. The last column shows the number of n-grams in the malware profile after removing redundant *n*-grams appearing in the benign dataset.

We analyse the data which is partially displayed in Table 5 that we obtained using the paired-samples Wilcoxon test to see if there is evidence of a real difference with and without removing the Memory Management class APIs and if there are real differences between the three benign/malware columns. At the .05 significance level, we conclude that with the Memory Management class APIs, there is a significant difference in the number of n-grams in the benign and malware (redundant) categories [$V = 49$, p-value = 2.734E-02] as are those in the benign and malware (unique) categories [$V = 55$, p-value = 1.953E-03] and for malware (redundant) and malware (unique) categories [$V = 55$, p-value = 1.953E-03]. Similar results are also obtained when we remove the Memory Management class APIs for the number of *n*-grams in the benign and malware (redundant) categories [$V = 51$, p-value = 1.367E-02] as are those in the benign and malware (unique) categories [$V = 55$, p-value = 1.953E-03] and for malware (redundant) and malware (unique) categories [$V = 55$, p-value = 1.953E-03].

Table 6: Percentage of *n*-gram reduction from the removal of the Memory Management class APIs

n Size	Benign (%)	Malware (Redundant) (%)	Malware (Unique) (%)
1	20.5	17.4	1.4
2	44.2	33.7	15.9
3	59.0	51.6	38.2
4	66.1	61.4	54.6
5	70.7	66.1	61.7
6	73.4	69.1	66.3
7	75.5	70.9	68.8
8	76.9	71.7	70.2
9	77.5	72.1	71.2
10	78.0	72.6	71.9
Average	64.2	58.7	52.0
Std. Dev.	18.6	19.0	25.3

Statistical analysis shows that the removal of the Memory Management class APIs from the dataset significantly reduces the number of *n*-grams [$F = 9.934$, $p < 5.520E-03$] as are those for the malware (redundant) [$F = 11.88$, $p < 2.870E-03$] and malware (unique) [$F = 8.734$, $p < 8.470E-03$] categories. Table 6 shows that the amount of n-gram reduction is in an increase trend and proportional to the size of *n*.

4.3. The NK and S Inspired Detection Algorithm

In our study, we form an algorithm based on the Self/Non-self-Theory. We have malware and benign profiles and identify a threshold that distinguishes between them. We adapt the roles of NK, which identifies the preferred threshold, and Suppressor, which controls its setting so that it will not overkill, to dynamically modify the threshold. We explain the profiles, threshold and its self-adjusted process in the following sub-sections.

Figure 2 shows the general structure of API calls for trapping an executable. While an executable is running, the system captures its related sequences of API calls, which is processed and transformed before being passed to the decision component as a block of n-grams. Statistical data are retrieved from this block, containing the n-grams' degrees of closeness to both the malware and benign profiles. The decision component relies greatly on the in-

formation learned from past data contained in these profiles and uses a threshold to make the decision. If the executable's profile is above this threshold, it is deemed malware, otherwise benign. The process of selecting the threshold value is inspired by the roles of NK and Suppressor.

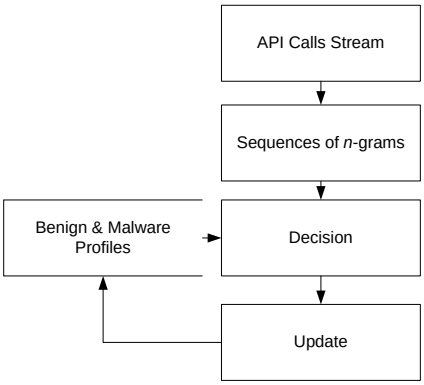

Fig. 2: General structure of API calls-trapping on single executable

Although many programs can be hooked, some mature or trusted ones can be relaxed or ignored, thereby avoiding the need to hook too many safe executables which may include protected executable and standard services of the OS.

4.4. Benign and Malware Profiles

Benign and malware profiles maintain a system's knowledge about its self and non-self. They were built based on knowledge of known benign and malware detected in the past.

In a commercial environment, an antivirus company can obtain these two required profiles by collecting API calls of common software used by users worldwide. Then, which program profiles need to be included would depend on the programs installed or available on the relevant computers. Its malware profile could be generated from its existing malware collection or from several websites that offer already-detected malware. The results are two common profiles for use in detection during execution, updates to which can be made, as necessary, under the supervision of the antivirus company. Within this framework, each machine will have a unique profile of itself, very similar to the HIS which is unique to an individual, but updates could also be standardised by the antivirus company. To determine whether this approach will work requires further research.

4.5. Detection Process

Details of the proposed algorithm are presented in this section. Throughout this paper, we use the notations listed in Table 7.

Table 7: Notations used for algorithm

Notation	Description
B	All benign files in training dataset
M	All malware files in training dataset
Bn	*n*-grams collected from all benign training files
Mn	*n*-grams collected from all malware training files in which *Mn* not subset of *Bn*
m	Mimics indicator for MHC level in cell; process for obtaining *m* value described on next page; and *m'* new value of *m* after adjustment influenced by *NK* and *S*
X	Current file to be evaluated and it is from testing data
Xn	*n*-grams of *X* testing data
y	Score value returned when *Xn* parsed into *Mn* and *Bn*
NK	Natural Killer cell
S	Suppressor cell

As an executable run, its *n*-grams are monitored. Then, to make a decision, the collection of *n*-grams seen so far (*Xn*) is compared with the total sets of n-grams from the known malware (*Mn*) and known benign programs (*Bn*). Three counts are obtained: the numbers of *n*-grams in common with malware, benign and neither. From this, the ratio *y* is computed as *y* = (*n*-grams in common with malware) / (*n*-grams in common with malware + *n*-grams in common with benign); if it is at or above a threshold (*m*), the executable is deemed to be malware.

The main question is the reliability of the decision. Another question of interest is how soon during execution can a reliable decision be made.

The initial value of *m* is found by computing ratio y for each known malware in the training set (using all its *n*-grams) and selecting the median of all of the *y* values. The value of *m* can be adjusted. Generally, the detection process undergoes the following two main stages.

4.5.1. Stage 1: Preparation

The *n*-grams of the malware (*Mn*) and benign programs (*Bn*) in the training set are obtained and gathered as two separate collections, with those that appear in both being removed from the *Mn* profile but retained in the *Bn* profile.

Mn and *Bn* are used for two purposes: as part of the process for determining the decision threshold (*m*); and for describing whether each test executable is malware or benign.

Then, for each file in M, its *n*-grams (*Xn*) are obtained and compared with those of *Mn* and *Bn*. The idea is to determine how each malware compares with other malware and with benign programs. The outcome from that comparison is a *y* value for each file in *M:* *y* = (total *n*-grams in common with *Mn*) / (total *n*-grams in *Mn* + total *n*-grams in *Bn*). These *y* values are sorted and their median is the initial value for *m*, using which benign programs will most likely be correctly detected. However, this initial value of *m* is clearly too high: by definition, many malware will fall below this threshold, and thus not be detected. Therefore, before the testing phase begins, an adjustment for *m* is made by simply following the rule that, if (*m* > *S*) *m* = *NK***S*, else *m* = *NK***m*, so that a new *m* is obtained.

4.5.2. Stage 2: Evaluation of Testing Set

For each file (*X*) in the testing set, *Xn* is parsed into *n*-gram matching *Bn* and *n*-gram matching *Mn*. From this, ratio *y* is computed as *y* = (total *n*-grams in common with *Mn*) / (total *n*-grams in *Mn* + total *n*-grams in *Bn*) and, if it is at or above the threshold (*m*), the executable is deemed malware.

4.6. K-Fold Cross-Validation

Cross-validation [27] is used to assess the results of a learning algorithm by dividing a dataset into two parts called training and testing sets. It is a common technique known as *k*-fold cross validation for which *k* is usually set to 10. Its objective is to provide each partition of a dataset with an equal chance of both validating other partitions and being validated. Thus, the *k*-fold aims to make use of all the data for both training and testing while avoiding the over-fitting that can arise if the same data is used in full for both training and testing.

The experiment at this phase is conducted in a controlled environment. We distribute the 98 benign and 416 malware files equally into 10 folds, following the standard *k*-fold (*k* = 10) cross-validation scheme. The former is sorted alphabetically, with each sequentially placed in one-fold, thereby resulting in 8 folds with 10 benign programs and 2 with 9. For the malware files, we first group them according to their malware types and sort them alphabetically. Then, those in the Trojan cluster are sequentially added to folds followed by those in the virus and worm clusters.

In each of these 10-fold data sets, one-fold is used as testing data and the other 9 form the training set. As one round of the *k*-fold begins, we pass the training set to the detection system, so it can begin learning it to find the values for *Bn*, *Mn* and *m*. Then, the testing folds are loaded for testing to begin. For each *X* file, the *y* value is obtained and compared against *m* which is moderated by *NK* and *S*. The *TP* and *FP* rates are calculated for each round of the folds and the normalised and averaged results representing the entire folds are obtained at the end of the experiment.

4.7. Performance Measures

We evaluate the performances of the detection system using:

- TP: The percentage of malware executables correctly classified as malware
- FP: The percentage of benign programs wrongly classified as malware

5. Results and Discussion

The aim of this experiment is to provide insight into the capability of this system to recognise malware and benign pre-emptively, which is useful for combatting most malware since they take the form of a single file or are embedded in a small program. However, it is not intended to detect malware which is specially crafted and embedded within a large software application and those with time bomb features, i.e.; the execution of malicious payload is scheduled to occur at a specific time later. In this pre-emptive-based detection, we attempt to determine how many n-grams are needed to reliably detect malware.

First, we test system performances on a range of numerical values, from the first to first 1000th *n*-grams, and then identify which percentage point of the malware execution, based on the *n*-gram sequences, generates the best results.

The size of each executable varies due to the operations programmed in it and its invocations of different patterns of API call sequences. While most executable in the dataset contain less than 1000 *n*-grams, some have many more. However, we assume that evaluating the files beyond the 1000th *n*-gram is unnecessary as the results are predictably much earlier, as can be seen in Figure 3 and 4.

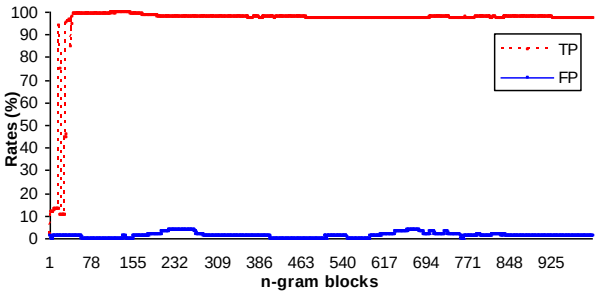

Fig. 3: Detections based on 1st to 1000th n-grams with NK and S

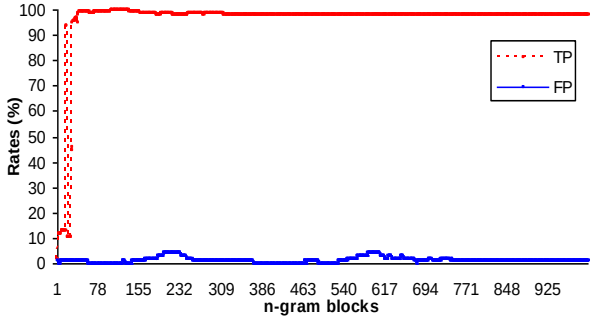

Fig. 4: Detections based on 1st to 1000th n-grams without S

It can be seen that pre-emptive detection achieves good performances at a consistent rate from the 29th *n*-gram, yielding averages of 95.19% for TP and 1.02% for FP with respective standard deviations of 3.0 and 3.2. To be more precise, most TP rates of the 10-fold tests produce at least 92.7%, and the sixth 100%, detection. The system records 0% FP rates on all folds except the ninth which is 10%.

When *NK* and *S* work in tandem, the peak performances appear in several places from the 113th to 135th *n*-grams, most of which record accuracy rates of 100% for TP and 0% for FP. This indicates that the system is capable of recognising benign programs and malware executable quite early in their execution, even after only 113 *n*-grams, with high accuracy.

We also evaluate the performance of the system with *NK* alone and find good performances still start at the 29th *n*-gram. Figure 3 shows the overall performances of the detection system with this setting. The peak performances appear in several places from the 113th to 135th *n*-grams. Within the range of these overall performances, most record accuracy rates of at least 99.76% for TP and 0% for FP, which at the 113th *n*-gram, reach 100% and 0%, respectively. Overall, the system performs slightly better with *NK* and *S* integrated rather than with *NK* alone.

How far through a malware's execution is the 113th *n*-gram? Table 8 shows details of each *k*-fold, and median and mean values of the percentage range for malware files; for example, for the malware in fold 1, when 113 *n*-grams are expressed as a percentage of malware total executions, the median value is 12.4%. This indicates that a reliable decision could be made in the very early stage of program execution.

Table 8: 113th *n*-gram as percentage of total execution

k-Fold	Median of % Range for Malware	Mean of % Range for Malware
1	12.4	28.7
2	10.0	28.8
3	5.0	30.5
4	10.3	31.9
5	5.8	32.6
6	10.3	32.3
7	14.7	34.2
8	9.2	35.7
9	8.5	37.2
10	10.6	39.4
Average	9.7	33.1
Std. Dev.	2.9	3.5

5.1. Distinguishing Benign from Malware

As can be seen in Figure 5, several benign programs are close to the decision threshold. We identify these files and find that MSN Messenger is the closest, followed by FreeCell, Word and Counter Strike. This indicates that, by following our API scheme, these executables generate *n*-gram patterns closer to malware profiles than do the rest of the benign program. Programs or executables that produce *y* values too close to the *m* line are at risk of being misclassified.

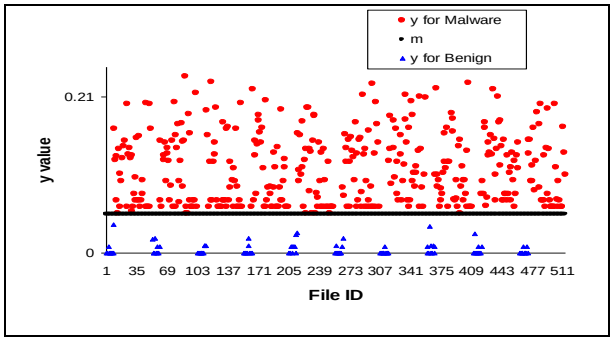

Fig. 5: Detection graph showing discrimination lines between malware and benign at 113th n-gram

5.2. Execution Speeds

The overall speed performances for pre-emptive detection as in Table 9 lists the average times taken to perform the evaluation tasks for the 10-fold data with their respective standard deviations, with and without the detection algorithm.

Table 9: Times taken to perform detection

Setting	Average/Fold (in seconds)	Std. Dev.	Differences without Detection Algorithm/Fold (in seconds)	Std. Dev.
NK+S at 29th	149.84	5.71	3.63	5.71
NK+S at 113th	150.88	5.27	1.39	6.48
NK+S on full execution	166.39	5.86	16.94	4.74

6. Conclusion

In this section, we presented a malware detection approach that can correctly distinguish between malware and benign programs. Using a data mining technique and inspired by the immune system of NK and Suppressor, the results showed that this system is robust. Also, its correct selection of API call sequences and numbers of n-grams also helped to achieve promising results.

The results suggested that we could effectively detect most of the malware executable and benign programs as early as the 29th *n*-gram. We obtained stable performances over a range of *n*-gram blocks, and the peak and perfect performances were seen soon after the first hundred, i.e., as early as the 113th *n*-gram.

Our future works include the evaluation against a bigger scale of the dataset and to include ransomware. Alongside with these experiments, we will begin the development of the malware detection software tools.

Acknowledgement

This work supported by a research project funded under The Ministry of Higher Education, Malaysia (Grant No FRGS/1/2015/ICT01/USIM/02/1).

References

[1] F-secure. Brontok.N. http://www.f-secure.com/v-descs/brontok_n.shtml.
[2] Forrest, S., Hofmeyr, S. A., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. Journal of Computer Security, 6(3), 151-180.
[3] Wespi, A., Dacier, M., & Debar, H. (2000). Intrusion detection using variable-length audit trail patterns. Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, pp. 110-129.
[4] Delves, P., Martin, S., Burton, D., & Roitt, I. (2006). Roitt's essential immunology (essentials). Wiley-Blackwell.
[5] Declercq, W., Vandenabeele, P., & Begley, T. P. (2007). Apoptosome and Caspase activation. Wiley Encyclopedia of Chemical Biology, pp. 1-12.
[6] Ismail, S. (2010). Apoptosis: Kematian terancang sel. https://www.majalahsains.com/apoptosis-kematian-terancang-sel/.
[7] Kim, J., Greensmith, J., Twycross, J., & Aickelin, U. (2005). Malicious code execution detection and response immune system inspired by the danger theory. Proceedings of the Adaptive and Resilient Computing Security Workshop, pp. 1-4.
[8] Fu, H., Yuan, X., & Hu, L. (2007). Design of a four-layer model based on danger theory and AIS for IDS. Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing, pp. 6331-6334.
[9] Dasgupta, D. (2006). Advances in artificial immune systems. IEEE Computational Intelligence Magazine, 1(4), 40-49.
[10] Matzinger, P. (1994). Tolerance, danger and the extended family. Annual Review in Immunology, 12, 991-1045.
[11] Aickelin U., Bentley P. Cayzer S, K. J., & McLeod J. (2003). Danger theory: The link between AIS and IDS. Proceedings of the 2nd International Conference on Artificial Immune Systems, pp. 147-155.
[12] Zekri, M., & Souici-Meslati, L. (2014). Immunological approach for intrusion detection. Revue Africaine de la Recherche en Informatique et Math. ematiques Appliquees, 17, 221-240.
[13] Akira, S., Takeda, K., & Kaisho, T. (2001). Toll-like receptors: Critical proteins linking innate and acquired immunity. Nature Immunology, 2(8), 675-680.
[14] Nektra Advanced Computing. Deviare API. http://www.nektra.com/products/deviare-api-hook-windows/.
[15] Microsoft. Detours - Microsoft research. http://research.microsoft.com/en-us/projects/detours/.
[16] API Monitor. (2010). http://apimonitor.com/order.html.
[17] nexginrc.org. API call dataset. (2010). http://nexginrc.org/Datasets/Default.aspx.
[18] Ahmed, F., Hameed, H., Shafiq, M. Z., & Farooq, M. (2009). Using spatio-temporal information in API Calls with machine learning algorithms for malware detection. Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, pp. 55-62.
[19] Symantec. W32.Almanahe.A. (2007). http://www.symantec.com/business/security_response/writeup.jsp?docid=2007-041317-4330-99.
[20] Kaspersky Lab. Malware profile search. (2010). http://www.kaspersky.com/find?
[21] Symantec. Malware profile search. (2010). http://searchg.symantec.com/search?
[22] VX Heavens. Virus collection. (2010). http://vx.netlux.org/faq.php#whole.
[23] Microsoft Corporation. MSDN library. (2010). http://msdn.microsoft.com/en-us/library.
[24] Witten, I. H., & Frank, E. (2005). Data mining: Practical machine learning tools and techniques. Morgan Kaufmann.
[25] Lim, H. (2016). Detecting malicious behaviors of software through analysis of API sequence k-grams. Computer Science and Information Technology, 4(3), 85-91.
[26] Forrest, S., Perelson, A .S., Allen, L., & Cherukuri, R. (1994). Self-nonself discrimination in a computer. Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 202-212.
[27] Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-validation. In L. Liu, & M. T. Özsu (Eds.), Encyclopedia of Database Systems. Massachusetts: Springer, pp. 532-538.