Joint Use of Set-Theoretic and Markov Models for Detection of Malicious Attacks

Sergey Frenkel and Victor Zakharov
Federal Research Center "Computer Science and Control " Russian Academy of Sc.,
Moscow, Russia, fsergei51@gmail.com, VZakharov@ipiran.ru

Abstract. This report considers some modelling issues to the estimation of security risks of programs due to malicious attacks, which is needed for rational choice and evaluation of the attacks protection methods. In particular, it can be the malicious codes action altering the target program's control data, say, as data that are loaded to processor program counter at some point in program execution. A basic underlying models of application program considered are calls graph and the Finite State Machine (FSM), corresponding to the algorithm implemented by this program. This FSM can be built either from a program source or from system calls traces. We will consider both the similarity metrics (for example, Jaccard similarity, the Edit (Levenshtein) distance) between the dynamic traces of system calls, and the probabilistic Markov effect attack models that are defined on the same traces.
We show some helpful relationships between the similarity distance based models and the Markov models.

Keywords: malicious attacks detection, similarity metrics, Markov chains, Finite State Machine

1. Introduction

Design of systems with security and safety deals with choice of mathematical models of the system to quickly and accurately estimate their ability to detect malicious programs. Malicious attacks detection is an essential part of this activity. The system calls (or an instruction sequences/set) automatic observation is a natural way of malicious codes detection. A widely-used approach of detection (or classification as "benign-malicious") is based on similarity measurement. Presently both the similarity metrics (for example, Jaccard similarity, the Edit (Levenshtein) distance [1]) between the dynamic traces of system calls, and the probabilistic Markov effect attack models that are defined on the same traces. The various similarity metrics were suggested [1]. One of the most discussed is the Edit Distance, namely, the minimal number of edit

operations (delete, insert and substitute of a single symbol) required to convert one sequence to the other, as it reflects the traces semantics. However, the time and space complexity of the edit distance between two strings $s1$ and $s2$ even the fastest algorithm requires quadratic complexity $O(|s|2/log(|s|))$, where $|s|$ denoted the length of the string [1]. Another aspect is the inherent imbalance between malicious and benign API traces that are harvested from the system, as most of the traces are benign. Therefore, clustering only on the malware traces where each cluster concentrates malwares with some specific common essence is more practical.

The Markov models suggested recently have more appropriate complexity, but they do not reflect the semantics of the traces, and there remains the problem of the imbalance between malicious and benign API traces.

In our approach, based on possibility to use a Markov model with two absorbing states defined on direct product of the spaces of two finite state machines (FSM), one of which is a program finite automaton model that is running under some original conditions, and second is the same FSM which at some point in time (depending on the considered temporal discreteness) underwent some malicious attack which transformed the trace of the program execution into a malicious one, do not assume that the original trace appeared as a result of the performance of a benign program, but simply analyze the possibility to detect some changes in the trace.

Using this approach we suggest in this report a framework for combining both graph-based and probabilistic modes enabling both the analysis of the system robustness to malicious attacks and malicious codes recognition and detection.

2 Problem description and related works analysis

The approaches to assessing the ability of projected programs (software systems) to detect malicious attacks (as well as their resistance to attack) can be divided into two classes: (i) based on the search for differences in the traces of normal and malicious programs, and (ii) based on the behavior of certain mathematical models representing these programs, among which, most often considered automaton models [2].

The first group of this models are the means of the traces classification over the classes “benign”, “malicious” on the base similarity measurement, which is performed through comparison of all pairs of the traces. Presently various similarity metrics were suggested, one of the most discussed is the Edit Distance, namely, *the minimal number of edit operations (delete, insert and substitute of a single symbol) required to convert one of compared sequence to the other* [1]. It can be done both for textual (e.g., in terms of Win API) traces representation (Fig.1), or as q-grams, which is all the contiguous substring of length n in sting s [1]. For example, given an integer q and a string s, the q-grams set of the string s is all the contiguous substring of length q in s, for example, 3-grams a for the string s=”peter” are “pet”,”ete”, “ter”. The size of q-grams set of $s = |s| - q + 1$,

The q-grams are collected by a process of shingling - passing a sliding window of length q on the string.

Representing strings of q-grams sets can be obtained by sorting the q-grams in the set according to, say, lexicographic ordering, and concatenating them as it is

described in [1]. In this case, the outcome is strings of n-grams (string over the alphabet of the q-grams) that are sorted and have no repetitions.

```
LoadLibrary lpFileName=VERSION.dll Return=SUCCESS
#272230000
CreateFile hName=C:\WINDOWS\System32\Wbem\wmic.exe
desiredAccess=GENERIC_READ  creationDisposition=OPEN_EXISTING
Return=SUCCESS
```

Fig 1. Fragment of systems calls trace. Here #272230000 means a *timestamp* in the trace, which also may be used sometime in the traces modeling .

The 3-gram of this trace is :
Loa oad adL dLi Lib ibr bra rar ary ry@ y@N @No Non one ne* e*C *Cr Cre rea eat ate teF eFi Fil ile le@ e@N @No Non

Along with edit distance of two strings x,y defined above, D(x,y), so called-Normalized Edit Distance of two strings (traces) x,y NED(x; y) = ED(x;y)/max(|x|,|y|) is used, in order to map the metric in [0,1] interval. Although the triangular inequity is not true for NED, it, in contrast to ED, has obvious probabilistic sense what is very useful for the detection analysis [7].

These metrics allow to clusterize the set of traces, corresponding to different programs (with different paths of execution) and detection is implemented by the distance (or similarities, e.g. Jaccard, or 1-NED(x,y)) relatively malicious codes clusters to provide the malicious codes detection by compare the distance of a suspicious trace from the clusters .

Since the system calls sequence can be represented as so-called “calls graph”, that is as a directed graph with labeled vertices, where the vertices correspond to functions and the edges to function calls [3], there is an extension of the ED notion for the graph, namely, as a directed graph in which functions are modeled as vertices, and calls between those functions as directed edges.

Graph edit distance (GED) is obtained by computing how many alterations have to be done to graph G for it to become isomorphic with graph H. In fact, the GED is size of the symmetric difference of the edge sets. The GED may be used for clustering of malicious samples in off-line training and then examine the on-line data for being benign or malicious by the distance of the data from existing clusters defined in the training phase.

The main difficulties of such approach is also its computational complexity. Therefore, the GED-based clustering problems is formulated as finding of minimal set of samples allowing an ε-approximation of the similarity metric with time polynomial complexity.

Second group of models can be defined either on traces abstraction by Markov chain generated by a finite state machine (FSM) with random input variables, or on the models of attack detection by comparing the normal and corrupted (caused by the attack) behavior of FSMs generating by the FSMs (also with random input variables) corresponding to both benign and malicious codes.

First of all, note, that many Markov models of anomaly detection are not built for a pair traces, but for individual traces only. For example, in [2], the program flow (in the traces form or not) is represented by a Markov chain in order to better fit the clustering to system call parameters (e.g., hName=C:\WINDOWS\System32\Wbem\wmic.exe of the system call CreateFile mentioned above in the Figure 1) and creates inter-relations among different parameters of a system call. The model states represent the system calls, or they represent the various clusters of each system call, in the form, in which the clustering was performed. Each transition will reflect the probability of passing from one of these groups of system calls to another through the program.

These models can be introduced basing on analysis of system calls that are collected dynamically from the system calls traces. The calls trace can be represented with a Markov chain structure in the transition matrix. The traces can be received by some tracing tools (e.g., CWSandbox, Joebox, Norman Sandbox [3]) to record the system call sequence of an application.

Considering that each distinct system call corresponds to one unique state of the Markov chain, it is possible to count the times of transition from one system call to another one to calculate the transition probability matrix.

For instance, suppose there are only four system calls, noted as SC0,SC1, *SC2*, and SC*3*, and the system call sequence as, say: SC0 → SC0 → SC1 →SC2 →SC3 → SC1 → SC2 →SC3 → SC0 → SC1 → SC0 → SC2 →SC3 → SC1 → SC2 →SC2.

The transition times from system call *i* to system call *j* (i,j*=0,1,2,3)* is noted as a_{ij}. In this case, $a_{00} = 1$, $a_{01} = 2$, $a_{02} = 1$, $a_{03} = 0$ and so on. Then we define, p_{ij}, the transition probability from system call *i* to system call *j*, in the following way: $p_{ij} = a_{ij}/\sum_j a_{ij}$

Elements of an estimated probability transition matrix $P=\{p_{ij}\}$ for the dynamic trace of each program are used as predictors to classify a program as malicious or benign.

But uncertainty in P for online classification will have a large impact on the decision until a sufficiently long trace is obtained.

In another approach the Markov chain can be used for a steady-state distribution computation which describes how likely each state, corresponding to given system call is to be visited in the long-run [4]. If a system call underlying the Markov chain to be in an unlikely state with great frequency, the sample could be considered as malicious behaviour. For example, let us a benign sequence involves the system calls RegEnumKeyExW, RegCloseKey, RegCloseKey, RegQueryValueExW, Reg-QueryValueExW, and the most-likely state (say, with the probability 0.3 which exceeds essentially the steady-state probabilities of others) _involves the system calls RegOpenKey followed by three repetitions of the seqence ReqQueryValueExW, Reg-CloseKey, and RegCloseKey. If in an unknown new trace of the system calls this call occurred with greater relative frequency than in the model, the new data could be deemed suspicious, as this calls deals with access to the user data.

Of course, the estimation of this probabilities assumes of some statistical tools, e.g., using a Kolgomorov-Smirnov non parametric test.

In view of the fact that in the automaton model we use only states, and in ED-recording of traces, it would be useful to have a model combining both types of information. The Hidden Markov Model (HMM) can be used to describe the statistical rules among the system calls [5].

For HMM model (S, O, A, B, π), the system call sequences are compared to the observed sequences O will be either normal or attack. Here are:

S = (Normal, Attack, Intrusion) is the state space, represents that the system call sequence has the following three states:

- *Normal* (N)
- *Attack* (A) (that indicates an attack activity that is setting itself up),

Intrusion (I) state indicates that an attack corrupted the normal behavior.

The maximum number of observations m is dependent on the number of system calls.

The sequences observed may be not distinct enough, what is a challenge of this model from the point of view the malicious property detection. Correspondingly, involving HMMs, we need to measure the size of uncertain disturbances in an underlying HMM. In particular, some possibilities of using relative entropy rate as a distance between normal and corrupted (perturbed) HMMs can be studied [6]. This relative entropy rate conveniently used to measure a "distance" between the two HMMs or Markov chains defined by the corresponding sets of parameters.

To illustrate an HMM, we consider an example where two hidden mechanisms, one malicious and one normal, are switched T times to generate an observation sequence O. We switch one of mechanism at a time, and we occasionally switch between the mechanisms. Suppose that the alphabet is {N; C} (which implies M = 2), where N stands for normal and C for corrupted behavior, and we observe the sequence O = {N, C, N, C, N, N}. There are two hidden states corresponding to the corrupted (by an attack) and normal traces.
Suppose that the transition probability matrix is

A =
0.9 0.1
0.20 0.80

where row (and column) 1 represents the normal coin, and row (and column) 2 represent the behavior of attacked program. Then, for example, the probability that the Markov process transitions from the normal state to the corrupted by the attack state is 0.1, since $a_{12} = 0.1$. That is, if the normal program is flipped, the probability the corrupted program is flipped next is 0.1.

The symbol distribution matrix B gives the probability distribution of a system calls SC1 and SC2 for both the normal and attacked states. Suppose that in this example we have

B =
0.5 0.5
0.7 0.3

where first row gives the probability of N and C, respectively, when the program under normal conditions normal is executed, and second row is the corresponding distribution for the attacked program.

The term b_{21} (SC1) represents the probability of SC1 when the attacked program is executed. In this example, $b_{21}(SC1) = 0.7$. There is also an initial distribution, _, which specifies the probability that the Markov process begins with the normal and corrupted states, respectively. For example, we can take $\pi = (0.5, 0.5)$

But the principal difficult of all well-known statistical models based approaches is that benign applications ("labels", from the machine learning viewpoint) are usually more imprecise than malware labels, in the sense that new applications that are originally labeled benign might later have its label changed to malware.

That is there is the inherent imbalance between malicious and benign API traces that are harvested from the system.

The result is that while we can be confident that an application is malicious, you can never be certain that a benign application is really benign or just an undetected malware.

This leads to essential difficulties with the statistic estimation.

It is important that these approaches were not investigated in conjunction with the handling of parameters and with a clustering of system calls based on such parameters.

Also, since in this Markov approaches deal with the state transition over all trace but not with specific system calls, there is not explicit relation with similarity metrics which are used widely for semantic analysis.

Therefore, let us consider the approach what we suggested recently for program robustness verification to attacks [8].

3. Markov chains product based Model of robustness to attacks estimation

We consider a trace of system calls (with their parameters - see Figure1) as a trajectory of transition of an FSM working with the alphabet used in the trace representation. This FSM can be built either from a program source or from system calls sequences (see below). We consider the attacks effect (that is the malicious codes action) as the *control-data attack which* alter the target program's control data.

The automation model corresponding to the automaton–based representation of the system calls sequences mentioned in the Section 2, where each vertex of the system call graph representing the sequence corresponding to a state of the automaton, and edges are transitions from one state to another. Then we consider a result of the attack as a replace of a system call SC during given timestamp in the program execution trace by another system call.

In terms of this automaton model, let us define a malicious behaviour model which would permit to coordinate such transitions of the system with an automaton behavior, that is the *Malicious behavior model is* $\{(a_i,a_j) \rightarrow (a_i, a_k)\}$, where (a_i,a_j) is an inter-state transition in the FSM, represented the program with normal behavior,

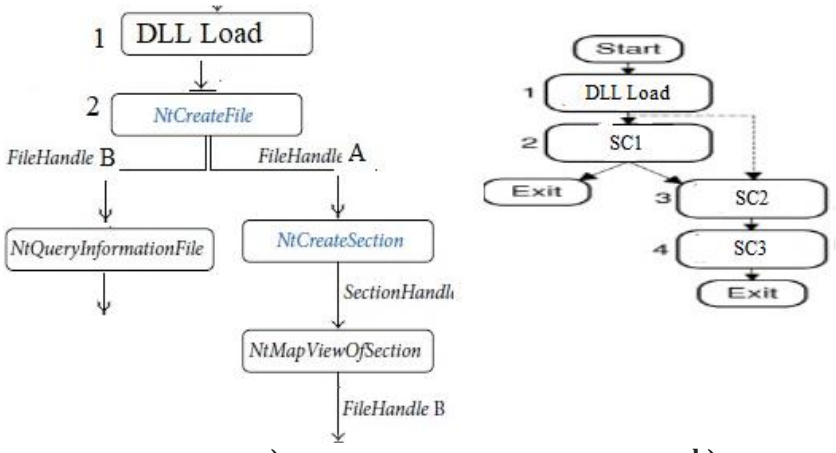
which was changed to the transition in the state a_k due to an attack. No new states arise.

Let us a program which may be subjected to an attack is represented by Mealy automaton (FSM) $s_{t+1} = \delta (x_{t+1},s_t)$, or, we consider the program on the system calls (parametrized, in general)) level as [9]), where δ is a transition system, x and s are input and state vectors correspondingly. The automaton clock *t* corresponds to execution of each of the program operator, or each of system call execution. A way of the FSM building from a system calls sequences see, e.g., in [10].

Let us assume, that all components of the input vector x are independent random variables. This independence can be provided by a specific choice of the FSM inputs (see, for example, in [11]).

Let $\{M_t, t \geq 0\}$ is Markov chain (MC) describing the target behavior of target FSM with *n* states under random input, that is, functioning without effect of any faults caused by an attack (altering the flow graph, corresponding to the transition function δ) and $\{F_t , t \geq 0\}$ is the MC based on the same FSM but exposed by some altering transition. Let $Z_t = \{(M_t, F_t, t \geq 0\}$ corresponding to behavior of the MCs pairs that is MC with state space $S^2 = S \times S$ of pairs (a_i, a_j), $a_i, a_j \in S$. The size of the MC will be n(n-1) + 2. The matrix of transition probabilities of these MCs are calculated from the given FSM transitions table and the probabilities of Boolean input binary variables of the FSM as well. Along with the states, Z_t has two absorbing states A_0 and A_1, where A_0 is the event "by the moment the FSM's trajectory has been restored and the output is not distorted", A_1 is “malfunctioning has already manifested itself in the output signal”. The pairs of (a_i, a_j) states enables representation of any transient faults as "the FSM instead of the state a_i, in which it should be on this clock after the transition at the previous time cycle, as a result of the malfunction was in the state a_j".

We characterize the security regarding a malicious attack as the probability of event that the trajectories (states, transitions and outputs) of M_t and F_t will be coincided after the termination the attack causing a flow graph deviation, before than outputs of both FSMs (underlying these MCs) become mismatched. This probability that the FSM returns to correct functioning after some number *t* of time slots can be computed as probability to get in one an absorbing state, using Chapman-Kolmogorov equation expressed the probability vector $\vec{p}(t)$ of the states into which the falls the Z_t (and corresponding FSM as well, which is the product of these two FSMs) after *t* transitions in terms of initial distribution $\vec{p}(0)$ of the MC states. $\vec{p}(0)$ is determined by the initial states of the fault-free and faulty FSMs, and the state transition probability matrix of this two-states Markov chain. The components of the vector $\vec{p}(t)$ are the probabilities $p_0(t)$, $p_1(t)$ of getting into the absorbing state A_0 and A_1 mentioned above, and the probability of transitions to the rest (transient) states of the MC, in the sum equal to $1 - p_0(t) - p_1(t)$. If the fault-free FSM at the initial moment 0 is in the state i_0, and the faulty state (say, due to an attack effect) is in the state $j_0 \neq i_0$, then $p_{i_0 j_0}(0) = 1$, and the remaining coordinates of the vector $\vec{p}(0)$ are zero.



a) **b)**

Fig. 2. System call graph of the trace fragment of the Fig. 3a) is a graph obtained from the sequence, Fig.3 b) is its abstraction.

3.1 The example of the model application

Let us consider a segment of a system calls trace [12]:

Loaded DLL at 77250000 ntdll.dll
NtCreateFile(FileHandle = A, . . . , ObjectAttributes =“Sample.exe”)
NtCreateFile(FileHandle = B, . . . , ObjectAttributes = “1111.exe”)
NtCreateSection(SectionHandle = C, . . . , FileHandle = B)
. . .
13 NtQueryInformationFile (FileHandle = A, IoStatusBlock = · · ·)
11 NtWriteFile (FileHandle = B, · · ·)

Fig. 3 A fragment of system calls trace

is represented as graph in Fig. 2 a), or in more abstract, as Figure 2b), where the nodes represent the system calls executed, and a directed edge indicates a data between two nodes, an attack can, say, provoke execution NtCreateSection(SectionHandle = C, . . . , FileHandle = B) instead of NtCreateFile(FileHandle = A, . . . , ObjectAttributes =“Sample.exe”),

We assume in this example that in the fragment in question the "Exit" state is formed normally or "NtQueryInformationFile" is executed (assuming that after receiving information about the file object we finish the sub-task in question), or *NtMapViewofSection, (which* specified part of Section Object into process memory, *and* play a role in bridging between source and destination system calls).

For example, in order to open a specific file and then write to it, a program successively invokes *NtCreateFile*, *NtCreateSection*, *NtMapViewofSection*, and *NtWriteFile*.

Intermediate system calls, namely, *NtMapViewofSection*, play a role in bridging between source and destination system calls. In other words, system calls interact with other system calls, whereas intermediate system calls pass information to destination system calls (in particular, pass a file handle as a reference between modules and subs).

Function *NtMapViewOfSection* maps specified part of Section Object into process memory.

Using a methodology described in [11] we can receive a FSM, represented the transitions of the system calls. Namely, in order to build the Mealy automaton, this graph (considering each of vertexes as a state of an automaton) is rewritten as "Algorithmic State Machine" [13] in Fig. 4, where each vertex Y1,..Y5 are some abstractions of the operations which corresponding blocks of the program Fig.2 executing, the results of which are represented by output variables y1,..y5. Note, that function of vertex Y5 in this representation is to synchronize the condition checking (x1, corresponding to the conditional vertex 2 in the program (Fig.4)) only and the result to form. The (a_i,y_i) pairs are the states and the automaton output variables of the FSM (Fig.2(a)), and x1", is the input variable of the automaton.

Then, in accordance with definition of malicious behavior by an attack mentioned above, this attack altering the program flow graph (Fig.2 b)) is described in terms of this automaton transitions as {(4,3)→(4,2)}.

We think in this example that in the fragment of a normally functioning program, the state "Exit" is formed when or "*NtQueryInformationFile*" is executed (assuming that we are completing the task on obtaining the information about the file object), or *NtMapViewofSection*, (which specified part of Section Object in process memory, for interacting source and destination system calls).

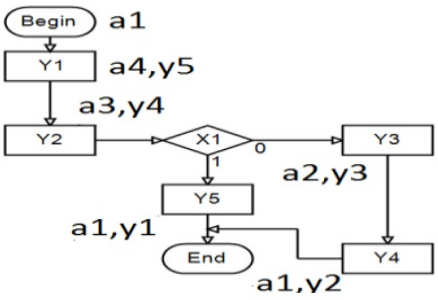
In this case, we may represent this fragment as ASM-diagram corresponding to the FSM.

Then, we could characterize the possibility to detect the attack using this path by the probability of its activation. Obviously, this probability depends on the probability of variable x1, which depends on whether it is produced in the normal functioning "FileHandle = B" or "FileHandle = A". That is the program's input data play a role in the abstraction of the program behavior by affecting the branching choice probability, that is the probability that input data provide the choice just a given branch.

Note, that this altering of the target program's control data, can be appeared, say, as data that are loaded to processor program counter at some point in program execution, or because of some reason the reference FileHandle B will be changed by FileHandle A.

A_n	A_n	X	Y
a1	a4	1	y5
a2	a1	1	y2
a3	a2	x1	y3
a3	a1	~x1	y1
a4	a3	1	y4

a)



b)

Fig. 4. States transition table ((a)) and "Algorithmic State Machine" ((b)) of the program Fig.1 corresponding to the FSM.

For example, let's Prob (x1=1)=0.9, which is the probability that result of block 2 Fig. 2 b) activates the exit from the module. Then, probabilities that the output values (say, y2 in Fig. 4b) of the program has already manifested itself to the given clock as corrupted, what means the attack detection, can be obtained by the solution of the above Markov chain is the following vector: P_D = (0,0, 0.09, 0.091, 0.093, 0.1629, 0.1638), where the number of each position corresponds to the clock number minus 1 (the first component of the vector corresponds to the moment of failure and it is considered as a zero index).

It means, that to the fourth clock of the automaton work, when the Exit will be achieved the attack can be detected with probability about 0.09 only, that may be turned out rather small, from the point of view security requirements to this program.
But If the probability of the condition Prob(x1=1) = 0.4. this probability is about 0.3, that has essentially more chances to be detected.
Thus, the model reflects the dynamic of the program behavior.

However, in contrast to the Markov chain based model considered above, we can indicate explicitly the specific transformation of the system calls sequence, and its connection with change of a similarity metric (Jaccard, NED) as well.
Namely, basing on the considered probabilistic model of manifestation of the attack effect, we can consider the following two aspects of the evaluation of the effectiveness of malicious code detection.

1. We interpret the confidence in the successful detection of a program trace damaged by this malicious attack, as the probability of its manifestation at the output of the described virtual (hypothetical) machine. At the same time, a meaningful interpretation of such an automatic representation is that changes in system calls (which do not output the traceable code from an acceptable set of codes) will appear in the trace at the moment corresponding to some timestamp (see Fig.1) of the trace.

Then, if in the normally functioning program takes place the condition "Go To Exit" and if the program variable that controls this "Go To" act is checkable by an

built-in checker (which, suppose, convoys the program execution), the attack can be detected in a point corresponding to the node “Exit” (Fig.2 b), e,g., if the program’s code contains ”return 1”.

2. We consider the detection of malicious code, as the process of checking for belonging to a particular cluster of malicious traces. Since the probability considered above determines the frequency of possible appearances of traces corruption as a result of the attacks, in order to fully characterize the security of the projected software and hardware environment from malicious codes, it is interesting to relate these probabilities to the probabilities of correct traces classification as malicious, which is determined by distance (in a given metric) between the considered (suspicious) trace and corresponding clusters of malicious program traces.

Since the probability of a correct trace classification as benign / malicious (detection) is determined by the distance (in one or another metric) between original and corrupted (= malicious) traces, we estimate this distance.

For example, let we deal with system calls without parameters (what is used rather often), that is we consider a distance between the strings:

X=LoadLibraryNtCreateFileNtQueryInformationFile (the left branch of the Figure 2a), and
Y=LoadLibrary NtCreateSection NtmapViewOfSystem (the right branch).

Let us compute the Jaccard similarity, of their 3-gram representations:

The 3-gram representation $X_{3\text{-gram}}$ of X:

Loa | oad | adL | dLi | Lib | ibr | bra | rar | ary | ryN | yNt | NtC | tCr | Cre | rea | eat | ate | teF | eFi | Fil | ile | leN | eNt | NtQ | tQu | Que | uer | ery | ryI | yIn | Inf | nfo | for | orm | rma | mat | ati | tio | ion | onF | nFi | Fil | ile | le | e

while $Y_{3\text{-gram}}$

Loa | oad | adL | dLi | Lib | ibr | bra | rar | ary | ry | y N | Nt | NtC | tCr | Cre | rea | eat | ate | teS | eSe | Sec | ect | cti | tio | ion | on | n N | Nt | Ntm | tma | map | apV | pVi | Vie | iew | ewO | wOf | OfS | fSy | Sys | yst | ste | tem | em | m

Jaccard similarity of these 3-grams-representation:

$$J\_sim\ (X_{3\text{-gram}}, Y_{3\text{-gram}}) = 17/50$$

With clustering based on hashing (Minhashing [1]), this metric represents the probability of getting into the "own" cluster, which can be significantly increased due to the proper choice of hash functions:

$$p=1-(1-\alpha^{r})^{b}$$

where α is the similarity value (interpreted as a probability to get in the right cluster), parameter of Min-hashing function ("signatures") *r* control the filtering effectiveness and *b* controls the approximation factor. i.e., the bigger the parameter *r* is, the more non-similar strings are filtered, the bigger of *b*, the better approximation to the real result.

Therefore, we should consider the probability of the result of the action of the attack specified by the vector P considered above when specifying the requirements to the specified hashing mechanism. At low detection probabilities in the hardware-software environment (0.09, in the example), performing the attacked program, it is necessary to perform the appropriate selection of the parameters r and b, namely, to strive to provide more accuracy, by increasing the number of calculations (reducing the filtering ability, checks of similarity of the tested trace with clusters, including obviously non-similar traces) with $\alpha = 17/50$.

Accordingly, even in the second case, when the likelihood of the result of a malicious attack is relatively high (0.4), the system developer may not look for additional solutions for the attack detection, because in this case the probability of detection is determined by probability only.

Otherwise, in the case of rather high similarity between original and corrupted traces (say, for $J_{\_SIM} = 0.8$), the system developer could use the model information to reduce the requirements to the clustering.

We consider the attacks effect (that is the malicious codes action) as the *control-data attack* which alter the target program's control data, say, as data that are loaded to processor program counter at some point in program execution.

We note that, as shown in our paper [7], that knowing Jaccard similarity, one can estimate the boundaries for the NED values, which in many practical cases turn out to be rather narrow.

4 Conclusion

This paper analyzes some modelling issues to the estimation of security risks of programs due to malicious attacks, which is needed for rational choice and evaluation of the attacks protection methods.

We consider both various probabilistic models and strings (set, graph) similarity-based approaches to detect a malicious code. The structural model of the software analyzed is a graph representation of the program execution traces, and therefore we suggest and analyze an approach to estimation of security risks of the programs due to malicious attacks which try to change the control flow of the program to corrupt the program behavior.

It is shown the possibility to use a Markov model with two absorbing states defined on direct product of the spaces of two finite state machines (FSM), one of which is a program finite automaton model that is running under normal conditions, and second is the same FSM in which at some point in time (depending on the considered temporal discreteness) there was a failure due to external attacks (e.g., within the time of a single operation, or a program's block execution).

A way of combination of both the similarity metrics (for example, the Edit (Levenshtein) distance) between the dynamic traces of system calls, and the probabilistic Markov effect attack models that are defined on the same traces for the estimation of detection ability was considered.

Because of well-known problem of imbalance between malicious and benign system calls traces, in our approach we do not assume that the given trace appeared as a result of the performance of a benign program, but simply analyze the possibility to detect some changes in the trace. In doing so, we assess the possibility of detecting these distortions in the trace by comparing them with clusters into which the traces of accumulated malicious programs are divided. And if in none of the classifications the estimated trace does not fall, we deduce it from the category of suspicious.

References

1. Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman, Mining of massive datasets, Cambridge university press, 2014
2. Federico Maggi, Matteo Matteucci, and Stefano Zanero, Detecting Intrusions through System Call Sequence and Argument Analysis, In Trans on Dependable and secure computing, Volume: 7, Issue: 4, Oct.-Dec. 2010,pp.381-396
3. Malheur: *Automatic Analysis of Malware Behavior*, http://www.mlsec.org/malheur
4. Geoff Mazeroff, Victor De Cerqueira Jens Gregor Michael G. Thomason Probabilistic Trees and Automata for Application Behavior Modeling, 41st ACM Southeast Regional Conference Proceedings,pp.435-440
5. Feng Zhao, Hai Jin, Automated approach to instruction detection in VN-based Dynamic execution environment, In Computing and Informatics, Vol. 31, 2012, 271–297.
6. Li Xie, Valery A. Ugrinovskii, and Ian R. Petersen, Probabilistic Distances Between Finite-State Finite-Alphabet Hidden Markov Models, IEEE transaction on automatic control, Vol. 50, No. 4, April 2005.
7. Dolev S., Ghanayim M., Binun A., Frenkel S., Sun Y.S. Relationship of Jaccard and Edit Distance in Malware Clustering and Online Identification // Proceedings of 16th IEEE International Symposium on Network Computing and Applications, 2017. P. 369–373.
8. Sergey Frenkel, Victor N. Zakharov: Brief Announcement: A Technique for Software Robustness Analysis in Systems Exposed to Transient Faults and Attacks, in CSCML 2017: 196-199, //Cyber Security Cryptography and Machine Learning, First International Conference, CSCML 2017, Beer-Sheva, Israel, June 29-30, 2017**,** Proceedings**.** Lecture Notes in Computer Science 10332, Springer 2017.
9. Jacob, G., Debar, H., Eric Filiol, Behavioral detection of malware: from a survey towards an established taxonomy, J Comput Virol (2008) 4:251–266
10. Seeger, M. M., Using control-flow techniques in a security context: A survey on common prototypes and their common weakness. In *Network Computing and Information Security (NCIS), 2011 International Conference on*, volume 2, pages 133–137, May 2011.

11. Frenkel, S., Zakharov, V., Basok, B.: Technical report of FRC “Computer Science and Control” of RAS, Moscow, Russia (2017), http://www.ipiran.ru/publications/Tech_report.pdf

12. Jae-wook Jang, Jiyoung Woo, Aziz Mohaisen, Jaesung Yun, and Huy Kang Kim, Mal-Netminer: Malware classification approach based on social network analysis of system call graph, Mathematical Problems in Engineering Volume 2015 (2015), Article ID 769624, 20 pages.

13. 3. Baranov, S.: ASMs in high level synthesis of EDA tool Abelite. In: DESDes’09 Int. IFAC Workshop Proceedings. – Valensia, Spain, pp. 195–200 (2009)