A Monte Carlo Tree Search approach to Active Malware Analysis

Riccardo Sartea
University of Verona
Department of Computer Science
riccardo.sartea@univr.it

Alessandro Farinelli
University of Verona
Department of Computer Science
alessandro.farinelli@univr.it

Abstract

Active Malware Analysis (AMA) focuses on acquiring knowledge about dangerous software by executing actions that trigger a response in the malware. A key problem for AMA is to design strategies that select most informative actions for the analysis. To devise such actions, we model AMA as a stochastic game between an analyzer agent and a malware sample, and we propose a reinforcement learning algorithm based on Monte Carlo Tree Search. Crucially, our approach does not require a pre-specified malware model but, in contrast to most existing analysis techniques, we generate such model while interacting with the malware. We evaluate our solution using clustering techniques on models generated by analyzing real malware samples. Results show that our approach learns faster than existing techniques even without any prior information on the samples.

1 Introduction

Malware are one of the biggest threats in IT security, with millions of malicious applications released every year at an ever growing rate. For this reason, automated techniques based on machine learning are fundamental tools for helping security experts in analyzing and classifying dangerous software. Common approaches can be broadly categorized as *static*, where the binary code of the malicious program is inspected but not actually executed [Sharif *et al.*, 2008; Lakhotia *et al.*, 2013; Yang *et al.*, 2014], as *dynamic*, that involves the program execution inside a safe environment to observe its behavior [Meng *et al.*, 2016; Gascon *et al.*, 2013; Zhang *et al.*, 2014; Shin *et al.*, 2011]. A limitation of all the mentioned techniques is that they are *passive*, meaning that they do not interact with malware during execution.

Dynamic analysis can also be conducted *actively*, using a methodology in which the analyzer interacts with the infected system in order to trigger malicious behaviors that would otherwise remain invisible. Active Malware Analysis (AMA) gained significant attention in the last years, and the first steps towards this approach were presented in [Moser *et al.*, 2007], where the authors discussed the existence of malware requiring specific inputs to show their malicious behaviors. Recently, AMA has been applied to various scenarios, and specifically to smartphones [Suarez-Tangil *et al.*, 2014]. Such approaches focus mainly on the Android system, which is nowadays one of the most important target of malware infection.

An interesting branch of work addresses AMA by using game-theoretic approaches, and specifically stochastic games [Williamson *et al.*, 2012]. A key element for such formalization is the availability of a model for the malware that must be manually designed by a security expert. Given this model, the procedure can then devise the most informative actions for the analyzer agent. To partially overcome this limitation, [Sartea *et al.*, 2016] propose an automated algorithm for generating the model based on an analysis of malware execution traces. A limitation of such previous work is that the malware model to be used is static, meaning that it is fixed before starting the analysis and cannot be changed during it. Another limitation is that a prior knowledge of the malware to analyze is required in order to generate the model.

In this work we design a learning approach that can be used in domains involving learning behavioral models with interacting agents. Examples include multi-agent learning such as interactive apprenticeship learning, where a learner agent queries a teacher agent for specific example traces. We propose a reinforcement learning algorithm based on Monte Carlo Tree Search (MCTS) and stochastic games, applying it to the interesting context of AMA. Our approach aims at selecting the most informative action for the analyzer agent, generating the malware model during the analysis. This removes the need to provide a static malware model for the analyzer agent. A key element to select the most informative action is the ability to represent the dynamics of the malware, i.e. the probability that a malware sample will execute a sequence of actions in response to a specific action of the analyzer (usually called a trigger). In our work, we use Markov chains to represent the malware dynamics. This is a natural choice, given that we use stochastic games to represent the AMA process, and allows us to efficiently compute the probability distribution over the malware actions. We tested our approach on real Android malware, with results showing that our solution has superior performance with respect to previous techniques for AMA [Sartea *et al.*, 2016].

Our contributions to the state of the art can be summarized as follows:

1. We remove the need of a static, pre-specified model for AMA. Specifically, we model AMA as a stochastic game and we develop a reinforcement learning approach based on MCTS that can generate the malware model at runtime, i.e. while interacting with the malware.
2. We represent the dynamics of the malware by using Markov chains. This allows us to efficiently compute the probability distribution over the possible malware responses to the analyzer's actions.
3. We empirically evaluate our approach by running AMA on a dataset of real Android malware [Sartea *et al.*, 2016]. We analyze the results of the AMA by grouping malware behaviors, i.e. the transition function of the stochastic game learned by the analyzer, through standard clustering techniques (K-Means clustering and Hierarchical Agglomerative Clustering). Results show that our approach correctly groups malware samples using fewer actions than the state-of-the-art approach [Williamson *et al.*, 2012; Sartea *et al.*, 2016], and that it is able to identify the possible existence of malware subfamilies inside the main families, i.e. variants of given malware.

2 Preliminaries and Related Work

In this section we provide the necessary background on stochastic games, MCTS and related work on AMA.

2.1 Stochastic Games

Definition 1 (Stochastic Game)**.** *A stochastic game G is a tuple* $G = (S, N, A, T, R_i)$ *where:*

- S *is a set of stage games called states*
- $N = \{i \mid 1 \leq i \leq n\}$ *is a finite set of players*
- $A = A_1 \times ... \times A_n$ *is an action profile, where* A_i *is a finite set of actions for player* i
- $T : S \times A \times S' \rightarrow \mathbb{R}_{[0,1]}$ *is a probabilistic transition function*
- $R_i : S \times A \rightarrow \mathbb{R}$ *is the reward function for player i*

The game starts from an initial state, and the joint actions of players lead from one state to another with a probability given by the transition function. All players are assigned a reward depending on the choices made by all of them.

Markov chains describe the dynamics of the states of a stochastic game where each player has a single action in each state. Similarly, if players' strategies are stationary, the dynamics of the states of a stochastic game form a Markov chain [Neyman, 2003].

Definition 2 (Markov chain)**.** *A Markov chain M is a tuple* $M = (S, \boldsymbol{u}, \boldsymbol{P})$ *where:*

- $S = \{s_1, ..., s_n\}$ *is a set of states*
- $\boldsymbol{u}$ *is a* n*-length vector of the starting distribution*
- $\boldsymbol{P}$ *is an* $n \times n$ *transition probability matrix*

The dynamics of a Markov chain are completely defined by a transition probability matrix expressing the probability of moving from one state to another. Using theorem 1, Markov chains allow to efficiently compute the probability of reaching a specific state from another one in a stochastic game with stationary strategies, focusing on a single specific action [Grinstead and Snell, 2003].

Theorem 1. *Let* $\boldsymbol{P}$ *be the transition matrix of a Markov chain, and let* $\boldsymbol{u}$ *be the probability vector which represents the starting distribution. Then the probability that the chain is in state* s_i *after* n *steps is the* i^{th} *entry of the vector*

$$\boldsymbol{u}^{(n)} = \boldsymbol{u}\boldsymbol{P}^n$$

2.2 Monte Carlo Tree Search

MCTS is a method for choosing best actions in a given domain, random sampling the action space and building a search tree. The tree is built progressively, descending from the root guided by the results of previous descents. An action reward is estimated basing on the subtree built starting from an action node, with the estimate becoming more accurate after every descent. Consequently, the tree grows in an unbalanced way, favoring the expansion of most promising subtrees. The MCTS algorithm iteratively performs a sequence of four steps [Browne *et al.*, 2012]:

1. *Selection:* from the root node, a *tree policy* is recursively applied to descend the tree until the most promising node to expand is found. How the tree is built depends on how nodes are selected in this step (figure 1a)
2. *Expansion:* a child node is added to the selected one, according to the available actions (figure 1b)
3. *Simulation:* a simulation is run from the expanded node following a *default policy*, producing a reward (figure 1c)
4. *Backpropagation:* the reward is propagated up to the root, updating the statistics of the nodes encountered (figure 1d)

After a predefined computational limit has been reached, the action corresponding to a child of the root node is returned as result. MCTS is often used in domains where a standard tree search is unfeasible. In our approach, given the elevated number of possible analyzer actions, combined with the analysis game length and the uncertainty on malware responses, building a complete search tree would require too much computational effort.

2.3 Active Malware Analysis

Static analysis techniques include control flow graphs comparison [Sharif *et al.*, 2008], similarity detection [Lakhotia *et al.*, 2013], or malicious program logic extraction [Yang *et al.*, 2014]. For dynamic analysis instead, typical methods are based on API call sequences [Meng *et al.*, 2016], call graphs [Gascon *et al.*, 2013], call dependency graphs [Zhang *et al.*, 2014], or control flow graphs [Shin *et al.*, 2011]. These techniques though, are passive, hence they can miss important malicious behaviors visible only if triggered by specific actions on the infected system.

The work of [Williamson *et al.*, 2012] introduces AMA as a stochastic game between an analyzer agent and a malware sample, where the former tries to acquire information about

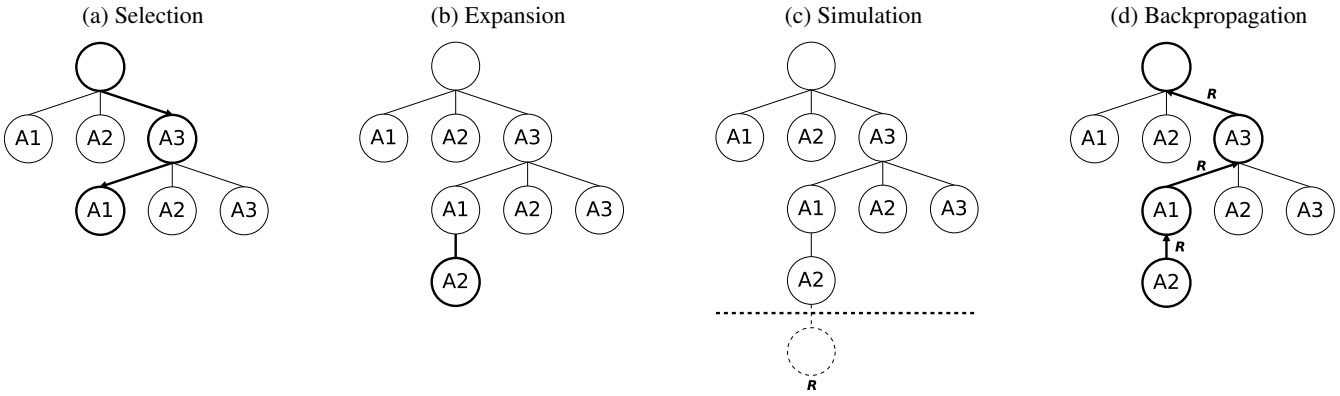
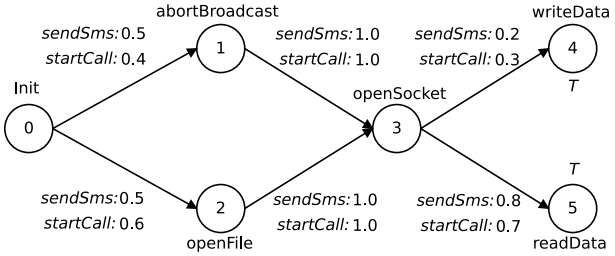
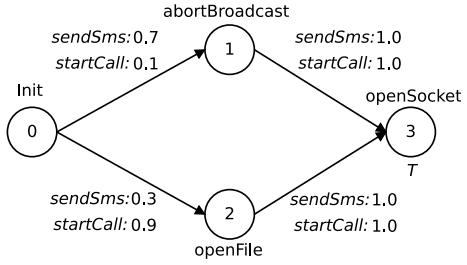


Figure 1: Monte Carlo Tree Search steps



(a) Malware model example



(b) Malware model example

Figure 2: Example of two malware models

the latter. Authors propose MYOPIC, an algorithm that tries to extract the policy of a malware sample stimulating it with the most informative action based on entropy, so as to acquire information about the malware behaviors. The strategy of the analyzer is stationary, and it selects the action with highest entropy in the model. The strategy of the malware sample is supposed to be stationary too and embedded in its code.

AMA requires a model as analysis input that is manually designed basing on the system on which the analysis is going to be performed. Recent work of [Sartea *et al.*, 2016] proposes an automated algorithm to generate the malware model to be later used as input for the analysis conducted on the execution traces of the malware. The analyzer action set is composed by all the possible triggering actions[1], whereas the malware action set includes all the possible API calls that can be executed on the system. A state is labeled with an API call (action) executed by the malware to transition from the preceding state to the current one. Joint actions of the two agents lead from one state to another with a probability given by the transition function. The model is similar to that of figure 2a, where one of the transitions goes with probability *0.2* from state *openSocket* to state *writeData* as a consequence of analyzer *sendSms* and malware *writeData* joint actions.

3 Analysis Process

The aim is to learn the transition function of a malware sample, i.e. the behavior, minimizing the number of analyzer actions to perform. Following [Williamson *et al.*, 2012] we model AMA as a stochastic game between the analyzer and a malware sample, where the analyzer chooses a triggering action and the malware sample responds with an execution trace as a sequence of API calls [Sartea *et al.*, 2016].

However, in contrast to [Williamson *et al.*, 2012; Sartea *et al.*, 2016], our approach generates the malware model at runtime using the information extracted from malware responses to the analyzer actions. The choice of which analyzer action to execute is made by using the model generated so far. These two steps are iterated multiple times during the analysis (algorithm 1). In particular, the algorithm starts with an empty model containing only the *Init* vertex, therefore with no in-

Algorithm 1 Monte Carlo Analysis

Input:
n - game length
Output: Malware model

1: $model \leftarrow \emptyset$ ▷ Start with empty model
2: **for** n times **do**
3: $tmpmodel \leftarrow model.Copy()$
4: $a \leftarrow MCTS(tmpmodel)$ ▷ Choose next action
5: $trace \leftarrow Execute(a)$ ▷ Observe malware reaction
6: $model.Update(trace, a)$
7: **return** $model$

[1]The triggering action set for the analyzer used in the experiments comprises 30 different actions

formation about the malware sample that is going to be processed. The decision of which analyzer action to perform is taken running a MCTS based on a copy of the current model being generated (copying the model is important because the simulation step modifies it). The chosen action is then executed on the system and the malware sample reaction is read as a sequence of API calls. The trace is converted into a path, starting always at *Init*, and used to update the model graph and statistics, i.e. the transition probabilities between API calls. The analysis game ends after the analyzer has performed a fixed number n of triggering actions[2], retrieving malware response execution traces. Finally, the model generated is returned as output.

In previous work, the model must be (manually [Williamson *et al.*, 2012] or automatically [Sartea *et al.*, 2016]) generated before starting the analysis, knowing exactly which malware families are going to be analyzed, or the structure of the system used to conduct such analysis. Instead, with our approach, models are automatically generated during the analysis without any prior knowledge, and can be compared anytime, by comparing their transition function vectors (see section 4.1).

4 Malware Model

The malware model contains all the information acquired during the interaction between the analyzer and the malware sample. The model is based on the dynamics of the stochastic game played during the analysis phase, such as those shown in figure 2. Vertices represent the states of the game and are labeled with malware API calls. Edges connect two consecutive API calls of an execution trace, and are labeled with transition probabilities conditioned by the actions executed by the analyzer. If a vertex is labeled with an API call that terminates one or more malware execution traces, such vertex is marked as terminal (T)[3]. A path on the model graph is a possible execution trace of the malware, and from the values on the edges we can compute the probability of reaching a terminal state from the initial one, i.e. the probability of an execution trace for a specific triggering action of the analyzer.

Probability values are assigned basing on the historical frequency extracted from the execution traces, conditioned by the specific analyzer action that triggered that response. For each analyzer action a we keep track of the number of times an edge has been traversed $e_{t,a}$, and the number of times a vertex has been reached $v_{r,a}$. Given a graph model $G(V, E)$ we can reconstruct the transition probability for every analyzer action a and outgoing edge $e \in E_v$ of vertex $v \in V$ with equation 1 as follows:

$$P(e \mid a) = \frac{1 + e_{t,a}}{|E_v| + v_{r,a}} \tag{1}$$

Notice that if statistics for a particular analyzer action regarding a vertex or an edge are missing ($v_{r,a} = 0$ or $e_{t,a} = 0$), the resulting probability value will be uninformative since we have no knowledge of the behavior in that specific case.

[2]We tested different game lengths from 1 to 10 (figure 3)

[3]An API call is considered terminal if no additional API call is performed after that one for a given number of seconds. This is not related to the game termination or length

4.1 Malware Comparison

Malware have distinctive features, such as the payload they carry or how they infect a system, and can be grouped basing on one or more of these traits, forming the so-called families. Moreover, within the same family there can be variations of malware with similar features, but behaving differently, e.g. how they release the same kind of payload.

The end goal of AMA is to infer whether malware are related to each other, sharing common behaviors. To be able to do this, we need a method for comparing models obtained after the analysis, so as to quantify their (possible) similarity. Our model formalization allows us to fuse the graphs, merging vertices basing on API call labels, and to extract the transition function (the T of the stochastic game definition 1) of each malware model projecting its statistics on the merged graph using equation 1.

The vectorial representation of the transition functions extracted by the comparison of models in figure 2 is illustrated below. The resulting merged graph shape is the same of figure 2a since it includes the one of figure 2b.

$$\begin{array}{rc} & \quad sendSms \qquad\qquad\quad startCall \\ a = & \big[0.5\ 0.5\ 1.0\ 1.0\ 0.2\ 0.8 \mid 0.4\ 0.6\ 1.0\ 1.0\ 0.3\ 0.7\big] \\ b = & \big[0.7\ 0.3\ 1.0\ 1.0\ 0.5\ 0.5 \mid 0.1\ 0.9\ 1.0\ 1.0\ 0.5\ 0.5\big] \end{array}$$

Transition functions of single models projected on the same merged graph are comparable, since each position represents the probability value on the same edge under a specific analyzer action. A dataset of transition functions may then be analyzed with clustering, classification or other techniques based on feature vectors (see section 6).

5 Monte Carlo Tree Search Policies

MCTS algorithm can be applied to a wide range of domains and performed in many different ways, according to the policies implemented. In this section we explain the choices we made for our application of MCTS.

5.1 Tree Policy

In the selection step of the MCTS (figure 1a) we decided to model the choice of which child to descend at each level as the exploitation-exploration dilemma typical of the multi-armed bandit problem [Browne *et al.*, 2012]. Indeed, there is the need of balancing the exploitation of an action currently believed to be the best, with the exploration of other actions that may turn out to be the best in the long term.

Upper Confidence Bound

The Upper Confidence Bound (UCB) is a value associated to any arm of the multi-armed bandit problem, representing the confidence we have in that arm to be the optimal [Auer *et al.*, 2002]. Its first application as a tree policy is due to [Kocsis and Szepesvári, 2006] in the form of Upper Confidence Bound for Trees (UCT)

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

where n is the number of times the current node has been visited, $\overline{X}_j$ is the average reward of the child node j, n_j is

the number of times the child node j has been visited, and C_p is a constant[4]. The left-hand term encourages exploitation of nodes with higher reward, whereas the right-hand term encourages exploration of less visited nodes. The expected value of a node is the reward approximated by the simulations run from that node. The decision of which node to choose at each level during the descent in the selection step is taken basing on the highest value of UCT.

5.2 Default Policy

In the expansion step (figure 1b) we randomly append a new child node to the selected one (figure 1a) choosing from the actions yet to be added to such selected node. Our *default policy* simulates the sequence of analyzer actions and malware responses (figure 1c) from the expanded node to the end of the game. After each analyzer action, the malware response is used to update a temporary model, and the process is repeated until the end of the simulation, where a reward is computed. Algorithm 2 illustrates the procedure, that is the simulation step of the MCTS called by algorithm 1 at line 4.

Algorithm 2 Default Policy

Input:
n - expanded node
$tmpmodel$ - temporary model
Output: Resulting model of the simulation

1: $a \leftarrow n.action$
2: **repeat**
3: $\quad trace \leftarrow Simulate(a)$
4: $\quad tmpmodel.Update(trace, a)$
5: $\quad a \leftarrow ChooseAction(tmpmodel)$
6: **until** end of game
7: **return** $tmpmodel$

Trace Simulation

Malware response actions are simulated (line 3) using past information on the analysis: if an analyzer action has never been seen, a random sequence of API calls is generated. Otherwise, a past execution trace is returned with the same historical probability associated to the observation of that trace in response to that analyzer action.

Action Choice

We tested two different strategies for choosing the analyzer actions during the simulation (line 5): the first strategy chooses randomly (a wide used policy for MCTS [Browne *et al.*, 2012]), whereas the second is guided by an entropy-minimization heuristic, which chooses the action with highest entropy in the current temporary model. The rationale behind this is that having a higher entropy usually indicates having a more informative action.

[4]We used $C_p = 1/\sqrt{2}$, obtaining a good balance between exploitation of actions that are known to trigger malware responses, and exploration of actions that have unknown outcome

Reward

Following [Williamson *et al.*, 2012] we decided to employ an information-centric reward based on entropy. The process uses equation 2 to compute the entropy gain $H_m(a) - H_{m'}(a)$ for the analyzer action a, corresponding to the node of the search tree encountered during the backpropagation step (figure 1d), between the model m built so far and the model m' estimated at the end of the simulation.

$$H_m(a) = -\sum_i D_m(a)_i \ln D_m(a)_i \tag{2}$$

$D_m(a)$ is the probability distribution for the analyzer action a over every vertex in the model m labeled as terminal (T). For the model depicted in figure 2a, the computation uses theorem 1 as follows for $n = 3$ steps:

$$D_m(sendSms) = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}0 & 0.5 & 0.5 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0.2 & 0.8\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}^3$$
$$= \begin{bmatrix}0 & 0 & 0 & 0 & 0.2 & 0.8\end{bmatrix} \underset{4,5}{\Longrightarrow} \begin{bmatrix}0.2 & 0.8\end{bmatrix}$$

The probability distribution is restricted to terminal vertices (T) 4 and 5. Using Markov chains we can efficiently compute the reward function instead of visiting the graph searching for every possible path [Williamson *et al.*, 2012; Sartea *et al.*, 2016].

6 Empirical Evaluation

The main goal is to prove the efficacy of our analysis algorithm in generating the model at runtime without any prior knowledge. Moreover, we want to attest that the models generated can be successfully compared and grouped in relation to malware families, highlighting even the possible existence of subfamilies composed by variations of similar malware.

6.1 Methodology

We compare our approach to the results obtained by [Sartea *et al.*, 2016], using the same dataset, with the application of MYOPIC algorithm [Williamson *et al.*, 2012] to automatically generated models. AMA is most effective in analyzing malware reacting to triggers, so the dataset is composed by four existing Android malware families of spyware and bots: ZSone, GoldDream, SMSReplicator, and TigerBot. The malware samples have been downloaded from [Xi'an Jiaotong University, 2011], for a total of 40 samples, 10 for each family. Furthermore, ZSone family is composed by two subfamilies of 5 samples each.

The aim is to group the transition function vectors extracted by our approach, obtaining a composition as similar as possible to the malware families of the dataset. For this purpose we applied K-Means clustering, repeating analysis and clustering 10 times, and computing results as the average in terms of purity, inverse purity and f-score w.r.t. our

	A.u.C. Purity	A.u.C. Inverse Purity	A.u.C. F-Score
MCA Entropy-Min	**889.69**	**929.91**	**890.36**
MCA Random	882.47	909.11	882.63
MYOPIC	794.84	840.13	811.53

Table 1: Comparison of Area Under the Curve values

ground truth. Furthermore, we made use of Hierarchical Agglomerative Clustering to dissect the composition of families, identifying the possible existence of subfamilies.

The analysis environment is based on the Cuckoo sandbox [Cuckoo Foundation, 2016], specifically modified to meet the requirements of AMA. Indeed, a standard dynamic analysis is not enough for our purpose because we need to use the retrieved execution trace to update the model and to choose the next analyzer action to perform, repeating the process multiple times.

6.2 Results

Table 1 reports the area under the curve values for purity, inverse purity and f-score. These values are obtained by the application of our proposed MCA algorithm in the two variants for the default policy (random and entropy-minimization). Results clearly show that both versions of our approach are better than MYOPIC. The difference is statistically significant according to a Student's paired two-tailed t-test with $p < 0.05$.

Even though the random default policy gives good results, the best is obtained using a default policy based on the entropy-minimization heuristic to choose analyzer actions in the simulation step. This principle helps in choosing actions generating more informative models, obtaining better clustering results with fewer actions. Figure 3 reports the f-score (y-axis), with standard error of the mean vertical bars, against the number of actions played (x-axis). The graph shows that our approach learns faster than MYOPIC, even without any prior knowledge of the malware model. This advantage is gained thanks to the MCTS, and more specifically, to the simulation of possible future malware reactions triggered by the analyzer. The capability of correctly grouping malware belonging to the same family increases with the number of triggering actions executed by the analyzer, since with more actions it is possible to better discriminate or associate behaviors.

Figure 4 depicts the dendogram of the Hierarchical Agglomerative clustering, restricted to samples of ZSone. This family is composed by two subfamilies of 5 samples each. This composition is reflected in the dendogram: datapoints are grouped into two clusters $\{zs_4, zs_5, zs_7, zs_2, zs_{10}\}$ and $\{zs_1, zs_6, zs_8, zs_3, zs_9\}$ respectively, representing the two subfamilies, before being merged into the complete family.

In our experiments we set the computational limit of MCTS to fit the Android emulator boot time, plus the time for the installation of the malware sample on the guest machine (about 30s in total for each analyzer action). With this solution we add no extra time to the analysis compared to MYOPIC, even though we reach better results in terms of quality. It is possible to increase the computational limit of MCTS if more complex simulations are required. Finally, it is worth noting that MYOPIC requires additional time for generating the model before starting the analysis, whereas our algorithm generates it at runtime.

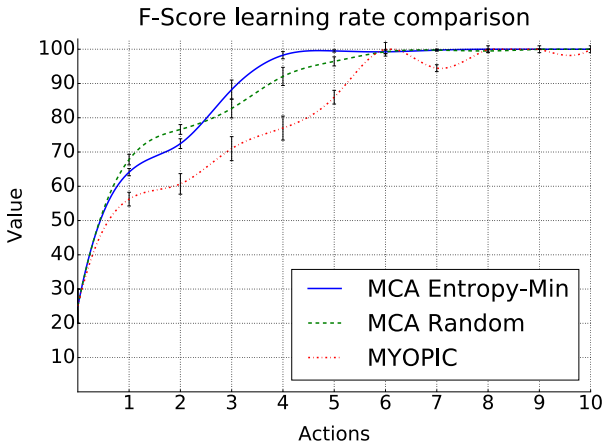


Figure 3: F-Score learning rate comparison

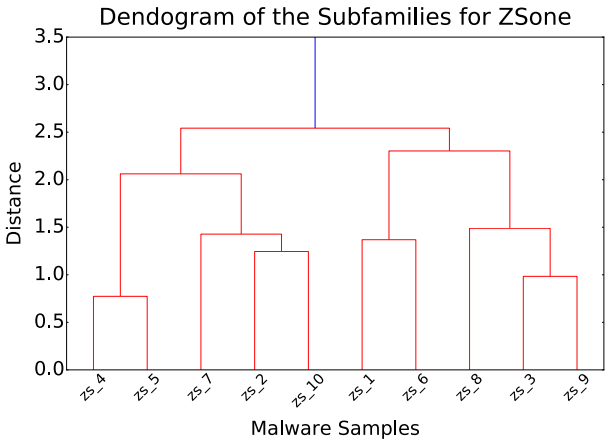


Figure 4: Dendogram of the subfamilies for ZSone

7 Conclusions

We proposed a reinforcement learning approach based on MCTS for AMA that, in contrast to existing analysis techniques, requires no pre-specified model as input. Models instead, are generated at runtime and can be compared to one another by extracting a compatible vectorial representation of the transition functions. We have been able to group malware in families using clustering techniques, basing on similar malware behaviors, highlighting the existence of possible subfamilies within groups. Results show that our approach learns faster than existing techniques even without any prior information on the malware to analyze.

References

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002.

[Browne *et al.*, 2012] Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[Cuckoo Foundation, 2016] Cuckoo Foundation. Cuckoo sandbox, 2016. `https://cuckoosandbox.org`.

[Gascon *et al.*, 2013] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 45–54, New York, NY, USA, 2013. ACM.

[Grinstead and Snell, 2003] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. AMS, 2003.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[Lakhotia *et al.*, 2013] Arun Lakhotia, Mila Dalla Preda, and Roberto Giacobazzi. Fast location of similar code fragments using semantic 'juice'. In *Proceedings of the 2Nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop*, PPREW '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[Meng *et al.*, 2016] Guozhu Meng, Yinxing Xue, Zhengzi Xu, Yang Liu, Jie Zhang, and Annamalai Narayanan. Semantic modelling of android malware for effective malware comprehension, detection, and classification. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 306–317, New York, NY, USA, 2016. ACM.

[Moser *et al.*, 2007] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 231–245, Washington, DC, USA, 2007. IEEE Computer Society.

[Neyman, 2003] Abraham Neyman. *From Markov Chains to Stochastic Games*, pages 9–25. Springer Netherlands, Dordrecht, 2003.

[Sartea *et al.*, 2016] Riccardo Sartea, Mila Dalla Preda, Alessandro Farinelli, Roberto Giacobazzi, and Isabella Mastroeni. Active android malware analysis: An approach based on stochastic games. In *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering*, SSPREW '16, pages 5:1–5:10, New York, NY, USA, 2016. ACM.

[Sharif *et al.*, 2008] Monirul Sharif, Vinod Yegneswaran, Hassen Saidi, Phillip Porras, and Wenke Lee. *Eureka: A Framework for Enabling Static Malware Analysis*, pages 481–500. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Shin *et al.*, 2011] Donghwi Shin, Kwangwoo Lee, and Dongho Won. *Malware Variant Detection and Classification Using Control Flow Graph*, pages 174–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Suarez-Tangil *et al.*, 2014] Guillermo Suarez-Tangil, Mauro Conti, Juan E. Tapiador, and Pedro Peris-Lopez. *Detecting Targeted Smartphone Malware with Behavior-Triggering Stochastic Models*, pages 183–201. Springer International Publishing, Cham, 2014.

[Williamson *et al.*, 2012] Simon A. Williamson, Pradeep Varakantham, Ong Chen Hui, and Debin Gao. Active malware analysis using stochastic games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 29–36. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[Xi'an Jiaotong University, 2011] Xi'an Jiaotong University. Androidmalshare, 2011. `http://sanddroid.xjtu.edu.cn:8080`.

[Yang *et al.*, 2014] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. *DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications*, pages 163–182. Springer International Publishing, Cham, 2014.

[Zhang *et al.*, 2014] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1105–1116, New York, NY, USA, 2014. ACM.