Detection of Intelligent Agent Behaviors Using Markov Chains

Extended Abstract

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

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

ABSTRACT

We consider the problem of detecting the behavior of intelligent agents operating in stochastic environments. In particular, we focus on a scenario where we are given two models for agent behaviors and we are interested in detecting whether one model appears within the other model. We use Markov chains to represent the behavioral models of the agents and we propose to extract the long-run probabilities as features that can be used to detect if one model is contained in the other. Results show that our approach is capable of detecting known strategies for agents interacting within classical games and to categorize malware behaviors.

ACM Reference Format:
Riccardo Sartea and Alessandro Farinelli. 2018. Detection of Intelligent Agent Behaviors Using Markov Chains. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 3 pages.

1 INTRODUCTION

Markov models are a powerful tool to represent probability distribution in stochastic processes, such as how an intelligent agent makes decisions in face of uncertainty. An interesting problem then is to detect whether a known strategy for a player appears in a given game evolution. This kind of problem has many practical applications for real multi-agent systems, and malware defense is one of the most interesting and significant.

In more detail, malware analysis is a crucial topic for cybersecurity with much ongoing research. The challenge is to be able to group similar malware in order to employ known countermeasures. Offline analysis and detection techniques allow to extract valuable information about threats that can be used to empower online detection tools, e.g., antivirus or intrusion detection systems. The most studied approaches can be divided in static [4, 8, 13] and dynamic [3, 5, 9, 14], along with some hybrid solutions. Static techniques analyze the binary code of a program without actually executing it, whereas dynamic techniques execute a program in a controlled environment studying the observed behavior. Interesting works in the context of multi-agent applications are [6, 12], where the analysis is formalized as a stochastic game between an analyzer agent and a malware agent, i.e. Active Malware Analysis (AMA). The result is a malware behavioral model, based on Markov chains, representing the dynamics of the analysis game. Malware models can then be compared and grouped in order to identify similar malware on the basis of their observed behavior.

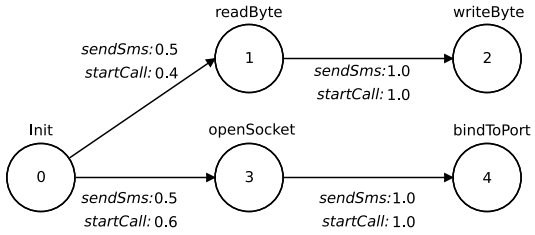

Figure 1: Behavioral malware model example

In this work we design a prototype framework aimed at detecting the presence of a given behavior inside the behavioral model of an intelligent agent. A behavior is seen as the specification of how an agent moves from one state to another of a Markov chain, expressed by the chain's transition function. We empirically evaluate our approach in two scenarios: first, we consider the problem of detecting known strategies in classical games, and in particular we consider the well known iterated Rock Paper Scissors (RPS). Then, we focus on a concrete cybersecurity scenario by analyzing real malware samples. Results show that our approach is able to successfully detect given behaviors in both scenarios.

2 EXTRACTION OF FEATURES

The behavioral models extracted by AMA are a composition of Markov chains, one for each analyzer action appearing in the model [6]. An example is visible in Figure 1, where the chains correspond to *sendSms* and *startCall*. Vertices represent the states of the interaction 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.

We are interested in comparing models based on Markov chains, especially in cases where a model may be contained within another. There is a lot of work on the study and comparison of Markov chain properties, such as the parameters of long-run distributions [1, 10, 11] or properties of absorbing Markov chains [2]. The structure of the models we take into account though, are a composition of multiple Markov chains, and all of them have to be considered at the same time. Moreover, they are not guaranteed to have at least one stationary distribution or to be absorbing for example.

The procedure we propose makes use of Theorem 2.1 that allows to compute the probability of reaching every state of a Markov chain starting from every possible other one.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

Theorem 2.1. *Let $\mathbf{P}$ be the transition matrix of a Markov chain, and let $\mathbf{u}$ 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*

$$\mathbf{u}^n = \mathbf{u}\mathbf{P}^n$$

Given a behavior D to detect and a behavioral model M, we compute the long-run probabilities of going from each state $s_i \in S_M$ to each other state $s_j \in S_M$. These values are then projected on the edges of model D if $(s_i, s_j) \in E_D$. The result is a model D' where the probability distribution on the outgoing edges between a state and its neighbors in D has been extracted from the paths connecting such states in the model M. The probability values labeling the edges of D' are then used as features to train a classifier capable of detecting if the behavior D appears in other models. Algorithm 1 details the described procedure. Notice that in line 3 we use the sum operator with Theorem 2.1 in order to consider the probability of reaching a state in $1, \ldots, T$ steps, and not exactly T.

Algorithm 1 Extract Features

Input:
D - $n \times n$ model to detect
M - $k \times k$ model to search for the presence of D
Output:
Detection features F

1: $F \leftarrow []$ ▷ Empty array
2: **for each** chain $c \in D$ **do**
3: $P = \sum_{n=1}^{T} M_c^n$ ▷ Compute the long-run behavior
4: $D' \leftarrow$ null $n \times n$ matrix
5: **for** $i \leftarrow 1$ **to** k **do** ▷ Project over D
6: **for** $j \leftarrow 1$ **to** k **do**
7: **if** s_i, s_j exist in D as $s_l, s_m \wedge D_{l,m} \neq 0$ **then**
8: $D'_{l,m} \leftarrow P_{i,j}$
9: $D' \leftarrow$ Normalize(D') ▷ Normalize rows
10: $F \leftarrow F +$ Flatten(D') ▷ Concatenate D' to F
11: **return** F

3 EMPIRICAL EVALUATION

The main goal of the empirical evaluation is to prove the effectiveness of our approach in extracting the long-run probabilities as features useful to detect given behaviors. We divide the empirical analysis in two types of experiments: in the first one we focus on players interacting within the iterated RPS, whether in the second experiment we analyze real Android malware families trying to identify malicious behaviors. Algorithm 1 has been used to extract the features from all the models generated, i.e. every model has been used alternately as D while using all the others as M, and to train a classifier. For all the experiments the classifier used is a Support Vector Machine (SVM) based on a Radial Basis Function (RBF) kernel. The quality of the classification has been evaluated with a stratified k-fold cross validation with $k = 5$ and repeated 10 times. Results are reported by average accuracy, precision, recall, and F_1-score along with the standard error of the mean.

As first testing setting we simulated the game of the iterated RPS between two players A and B. Hence, we design 4 possible behaviors for one of the players, i.e., player B, while player A chooses its actions following a randomly generated probability distribution. During the game simulation, the behavioral model of player B is observed and represented with Markov chains. Every strategy has been played 10 times in an iterated RPS of length 100, obtaining 40 behavioral models for player B. Results in Table 1 show that the classifiers trained with features extracted by observing the simulations reach a reasonable accuracy. This allows to effectively detect given known strategies of players interacting in a multi-agent context such as classical games.

Table 1: Player's strategy detection for the iterated RPS

	Accuracy	Precision	Recall	F_1-score
Tit-for-tat	1.00±0.00	1.00±0.00	1.00±0.00	1.00±0.00
Counter	0.92±0.01	0.78±0.03	0.98±0.01	0.86±0.02
Mirror	0.93±0.01	0.79±0.03	0.99±0.01	0.87±0.02
Random	0.93±0.01	0.79±0.03	0.99±0.01	0.88±0.02

Table 2: Malware family identification

	Accuracy	Precision	Recall	F_1-score
ZSone	0.92±0.01	0.93±0.02	0.94±0.02	0.92±0.01
GoldDream	0.97±0.01	1.00±0.00	0.89±0.03	0.93±0.02
SMSRep.	0.92±0.01	0.90±0.02	0.97±0.01	0.93±0.01
TigerBot	0.97±0.01	1.00±0.00	0.90±0.03	0.93±0.02

In our second experimental setting we analyzed a dataset composed by four existing Android malware families of spyware and bots, for a total of 40 samples, 10 for each family: ZSone, GoldDream, SMSReplicator, and TigerBot [7]. It is worth mentioning that the ZSone family is composed by two subfamilies of 5 samples each, meaning that the malicious behavior is slightly different between the two subgroups. The aim is to use our proposed approach in order to be able to identify the correct family of the malware samples analyzed. Results reported in table 2 show that ZSone and SMSReplicator families have lower accuracy compared to the others. This fact is a consequence of the ZSone composition in two subfamilies: the behavioral differences within this class are greater w.r.t. the others, making it harder to classify correctly and confusing the samples with SMSReplicator, the most similar other class. However, the overall results are promising.

4 CONCLUSIONS

We proposed a technique to detect given behaviors based on Markov chains. The procedure extracts the long-run probabilities of the states to be used as features for training a classifier to detect if a given behavior is contained in a target model. The empirical evaluation shows that our solution allows to search for the existence of known strategies for players interacting within a game such as the iterated RPS, and to categorize malware.

ACKNOWLEDGMENTS

Research reported in this publication was partially supported by University of Verona and Cythereal Inc. under Joint Projects 2017 Initiative (JPVR17ZMAL).

REFERENCES

[1] Persi Diaconis and Laurent Saloff-Coste. 1993. Comparison Theorems for Reversible Markov Chains. *Ann. Appl. Probab.* 3, 3 (08 1993), 696–730. https://doi.org/10.1214/aoap/1177005359

[2] Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2014. Designing Fast Absorbing Markov Chains. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)*. AAAI Press, 849–855. http://dl.acm.org/citation.cfm?id=2893873.2894005

[3] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. 2013. Structural Detection of Android Malware Using Embedded Call Graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AISec '13)*. ACM, New York, NY, USA, 45–54. https://doi.org/10.1145/2517312.2517315

[4] Arun Lakhotia, Mila Dalla Preda, and Roberto Giacobazzi. 2013. Fast Location of Similar Code Fragments Using Semantic 'Juice'. In *Proceedings of the 2Nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW '13)*. ACM, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/2430553.2430558

[5] Guozhu Meng, Yinxing Xue, Zhengzi Xu, Yang Liu, Jie Zhang, and Annamalai Narayanan. 2016. 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)*. ACM, New York, NY, USA, 306–317. https://doi.org/10.1145/2931037.2931043

[6] Riccardo Sartea and Alessandro Farinelli. 2017. A Monte Carlo Tree Search approach to Active Malware Analysis. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 3831–3837. https://doi.org/10.24963/ijcai.2017/535

[7] Riccardo Sartea, Mila Dalla Preda, Alessandro Farinelli, Roberto Giacobazzi, and Isabella Mastroeni. 2016. 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)*. ACM, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/3015135.3015140

[8] Monirul Sharif, Vinod Yegneswaran, Hassen Saidi, Phillip Porras, and Wenke Lee. 2008. *Eureka: A Framework for Enabling Static Malware Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 481–500. https://doi.org/10.1007/978-3-540-88313-5_31

[9] Donghwi Shin, Kwangwoo Lee, and Dongho Won. 2011. *Malware Variant Detection and Classification Using Control Flow Graph*. Springer Berlin Heidelberg, Berlin, Heidelberg, 174–181. https://doi.org/10.1007/978-3-642-24106-2_23

[10] Christopher C. Strelioff, James P. Crutchfield, and Alfred W. Hübler. 2007. Inferring Markov chains: Bayesian estimation, model comparison, entropy rate, and out-of-class modeling. *Phys. Rev. E* 76 (Jul 2007), 011106. Issue 1. https://doi.org/10.1103/PhysRevE.76.011106

[11] Luke Tierney. 1994. Markov Chains for Exploring Posterior Distributions. *The Annals of Statistics* 22, 4 (1994), 1701–1728. http://www.jstor.org/stable/2242477

[12] Simon A. Williamson, Pradeep Varakantham, Ong Chen Hui, and Debin Gao. 2012. Active Malware Analysis Using Stochastic Games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS '12)*. International Foundation for Autonomous Agents and Multiagent Systems, 29–36. http://dl.acm.org/citation.cfm?id=2343576.2343580

[13] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. 2014. *DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications*. Springer International Publishing, Cham, 163–182. https://doi.org/10.1007/978-3-319-11203-9_10

[14] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. 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)*. ACM, New York, NY, USA, 1105–1116. https://doi.org/10.1145/2660267.2660359