Bayesian Active Malware Analysis

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

Georgios Chalkiadakis
Technical University of Crete
School of Electrical and Computer Engineering
Chania, Greece
gehalk@intelligence.tuc.gr

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

Matteo Murari
University of Verona
Department of Computer Science
Verona, Italy
matteo.murari@univr.it

ABSTRACT

We propose a novel technique for Active Malware Analysis (AMA) formalized as a Bayesian game between an analyzer agent and a malware agent, focusing on the decision making strategy for the analyzer. In our model, the analyzer performs an action on the system to trigger the malware into showing a malicious behavior, i.e., by activating its payload. The formalization is built upon the link between malware families and the notion of types in Bayesian games. A key point is the design of the utility function, which reflects the amount of uncertainty on the type of the adversary after the execution of an analyzer action. This allows us to devise an algorithm to play the game with the aim of minimizing the entropy of the analyzer's belief at every stage of the game in a myopic fashion. Empirical evaluation indicates that our approach results in a significant improvement both in terms of learning speed and classification score when compared to other state-of-the-art AMA techniques.

ACM Reference Format:
Riccardo Sartea, Georgios Chalkiadakis, Alessandro Farinelli, and Matteo Murari. 2020. Bayesian Active Malware Analysis. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

In recent years the number of security flaws exploited by cyber-criminals has grown at an always increasing rate [25]. Among the most common infection vectors there are malicious software, often installed unknowingly by a legitimate user with the consequence of exposing computer systems to external threats. Furthermore, the increasing reliance on autonomous systems, e.g., cars, boats, as well as the spreading of smart and tiny embedded systems as part of the Internet of Things (IoT), resulted in billions of interconnected devices that are potential targets of malicious attacks [25]. In particular, Android is one of the most diffused operating systems employed not only in smartphones, but also in IoT, making it the preferred platform for cyber-criminals due to its huge market share [4]. In order to analyze the great amount of applications released every year, human cyber-security experts must rely on automated techniques to keep pace with the new threats discovered daily, since a manual analysis of every single one would be impossible.

A widely adopted methodology is dynamic analysis, in which an unknown application is safely executed within a sandbox and its runtime behavior is observed. In this context, much focus is on techniques that attempt to trigger a malicious application, i.e., a "malware", into showing core behaviors that would otherwise remain invisible without an interaction [16]. This kind of *active* dynamic malware analysis, known as AMA, has been shown to be more informative than the classical passive dynamic analysis [30], especially for systems that heavily rely on user input, such as smartphones [15, 19, 23]. Usually, the goal is to group unknown applications with respect to common behaviors or to a predefined set of classes. Indeed, malware can be grouped in families (or types), that are behavioral categories in which malicious applications fall into [6]. This grouping applies also to actions that trigger malicious behaviors: every family responds to a certain set of triggers, many of which are shared across different families. For example, both a spyware and a ransomware may react to an incoming sms: the first forwarding it to a third party, whereas the second encrypting its content. Previous works however do not use the concept of families to improve the analysis process adapting the analyzer strategy at runtime. By contrast, we use the available information on the families and on the characteristics of behaviors they contain, i.e., triggers, to guide the analyzer in selecting triggering actions that reflect the current belief regarding which family the malware being analyzed belongs to.

There exists an extensive literature that takes into account the type of the other agents to perform inference, with one key framework being that of Bayesian games [8]. These are used to model many domains, such as security games [9, 24, 31], coalitional games [2, 3], or network security solutions [10, 12]. In our proposed *Bayesian Active Malware Analysis (BAMA)* approach we build the formalization upon the link between malware families and the notion of types in Bayesian games. In particular, we formalize the analysis as a Bayesian game between an analyzer agent and a malware agent, focusing on the decision making strategy for the analyzer. Such strategy is guided by a utility function specifically designed to reflect the amount of uncertainty on the type of the adversary, according to the analyzer's belief. The aim is to be able to select

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

triggering actions that allow one to infer the type of malware with increasing accuracy at every stage of the game while it progresses.

We test our approach on a public dataset of real Android malware [29] comparing with other state-of-the-art AMA techniques [15, 19, 30] geared to solve the same problem of malware analysis and family matching that we are considering. Empirical evaluation shows that BAMA favourably compares to the other methodologies both in terms of learning speed and classification score.

In summary, our contributions are the following:

(1) We propose BAMA, a novel technique for dynamic active malware analysis, formalized as a Bayesian game between an analyzer agent and a malware agent, focusing on the decision making strategy for the analyzer. A key point is the design of the utility function, which reflects the amount of uncertainty on the type of the adversary after the execution of an analyzer action.
(2) The algorithm we devise to exploit the formalization is based on an entropy minimization principle applied in a myopic fashion to the Dirichlet prior that corresponds to the analyzer's belief. This allows us to successfully reduce her uncertainty on the type of the adversary at every stage of the game.
(3) We empirically evaluate BAMA on a public dataset of real Android malware. Results show a significant improvement in *(a)* the learning speed in terms of the number of analyzer actions required to reach the best classification score, as well as in *(b)* the best classification score, when compared to state-of-the-art AMA techniques.

2 RELATED WORK

AMA has attracted significant attention in recent years. For example in the work by [23], authors build an analyzer that tries to reproduce very specific activation conditions to trigger malicious payloads relying on stochastic models extracted by past samples of user execution behaviors. The analyzer described in the work of [15] instead, randomly interacts with the Graphical User Interface (GUI) performing thousands of actions in order to trigger malicious behaviors. The main limitation of such approaches is that the strategy of the analyzer does not adapt to what is observed during the analysis as it either reproduces past user execution traces or randomly selects the next analyzer action. A first step toward adapting the analyzer strategy to the malware reactions is taken in [30], where authors propose a game-theoretic framework for malware analysis. In particular, the analysis is formalized as a stochastic game between two players: an analyzer and a malware. The goal of the analyzer is to select the best action to perform on the system in order to trigger a reaction of the malware by using the information gathered during the analysis so far. In order to achieve such objective, the analyzer is given a fixed and manually pre-specified model of the behavioral patterns she aims to capture (based on the system on which the analysis is going to be performed). The model, that requires expert's knowledge to be created, is then used at runtime to store the observed transition probabilities between states that describe the behaviors of the malware, i.e., its policy extracted via interaction.

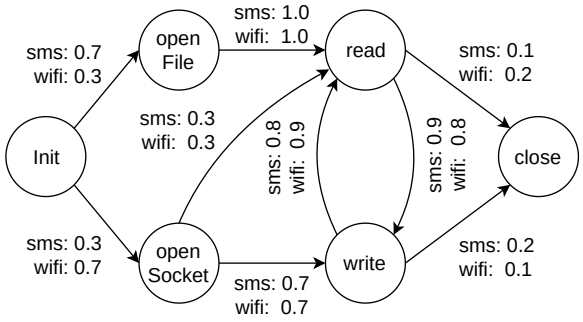

Figure 1: Example of malware model

To remove the requirement of a pre-specified model, authors of [19] propose Monte Carlo Analysis (MCA), an improved version of AMA in which the malware model is generated at runtime, and does not depend on the underlying system anymore. In particular, the model is formalized as a composition of multiple Markov chains, where each one represents the behavior of the malware in response to a specific action executed by the analyzer. In more detail, states are Application Programming Interface (API) calls extracted by the execution trace of the malware, and edges are labeled with transition probabilities conditioned by the analyzer action that triggered such transition. An example is visible in Figure 1 where the malware represented by that model transitions from state $write$ to $read$ with probability value 0.8 in response to the action sms of the analyzer. The same transition in response to $wifi$ instead has probability value 0.9. The analyzer action selection is performed with Monte Carlo Tree Search (MCTS), since without a pre-specified model the analyzer has to consider a huge number of possible malware responses in terms of execution traces. The reward of the game is designed to be the entropy gain between the current model and the one obtained at the end of the simulation step of the MCTS. When the analysis terminates, the transition matrices of the Markov chains representing the generated model are used as features for standard classification or clustering techniques, in order to group malware into their respective families.

Both works [19, 30] use an intelligent action selection strategy for the analyzer that improves over a random or sequential strategy. Nevertheless, if the number of possible analyzer actions is high, many analysis steps may be required to converge to the best classification score. In fact, authors employ exploration methodologies based on the information gathered so far, that however do not take in consideration the type of the adversary. By contrast, our proposed approach aims to exploit the intrinsic characteristics of the malware families, where each one is known to respond to specific stimuli depending on the malicious payload. With such information at hand, we model the analysis in order to reason about the malware family (type) at runtime, and to adapt the analyzer action selection strategy accordingly.

3 BACKGROUND

A Bayesian game is a game of incomplete information where each player can be of several types, each one corresponding to a possible payoff function for that player.

Definition 3.1 (Bayesian game). A Bayesian game with n players is a tuple
$G = (N, \boldsymbol{A}, \boldsymbol{\Theta}, \boldsymbol{u}, p)$ where

- N is the finite set of n players
- $\boldsymbol{A} = A_1 \times \cdots \times A_n$ where A_i represents the actions available for player i
- $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$ where Θ_i represents the types available for player i
- $\boldsymbol{u} = (u_1, ..., u_n)$ is a profile of utility functions with $u_i : \boldsymbol{A} \times \boldsymbol{\Theta} \rightarrow \mathbb{R}$ utility function for player i
- p is the common prior over $\boldsymbol{\Theta}$

A player's type θ_i is only observed by player i and encodes all relevant information about some important private characteristics of such player. Utility functions, and consequently strategy construction, take into account not only player's actions but also their types. To model a Bayesian game it is useful to introduce a special player called *nature* that randomly chooses a type for each player according to the prior probability distribution p that is assumed to be known by every player [8].

In our domain, a natural choice for prior is the Dirichlet [17] since it is the conjugate prior of the multinomial distribution that we employ as uncertainty measure over the possible malware families.

Definition 3.2 (Dirichlet distribution). A Dirichlet distribution of order $k \geq 2$ with parameters $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k) \in (\mathbb{R}_{>0})^k$ is denoted as $Dir(\boldsymbol{\alpha})$. A k-dimensional Dirichlet random variable $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$ with $\theta_j \geq 0$ for $j = 1, \ldots, k$ and $\sum_{j=1}^{k} \theta_j = 1$ has probability density[1]

$$Dir(\boldsymbol{\theta} \mid \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$$

where

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma(\alpha_0)} \quad \text{and} \quad \alpha_0 = \sum_{i=1}^{k} \alpha_i$$

Γ is the Gamma function

Our proposed algorithm uses the entropy of a Dirichlet distribution, which can be computed as follows [5]:

Definition 3.3 (Entropy of a Dirichlet distribution). The (differential) entropy of a Dirichlet distribution $Dir(\boldsymbol{\alpha})$ of order k is

$$H_D(\boldsymbol{\alpha}) = \log B(\boldsymbol{\alpha}) + (\alpha_0 - k)\psi(\alpha_0) - \sum_{i=1}^{k} (\alpha_i - 1)\psi(\alpha_i)$$

where ψ is the Digamma function

There are some differences in the entropy between a discrete or a continuous random variable (differential entropy). First of all, differential entropy can assume negative values, and this is the case for the Dirichlet distribution (visible also in Figure 3 of Section 5). Secondly, in the discrete case, entropy quantifies randomness of a system in an absolute way, whereas in the continuous case this quantification has only a relative meaning. Consequently, differential entropy cannot represent the absolute amount of information carried by a system, unless carefully interpreted. In this work, we use the entropy value of a Dirichlet distribution to compare multiple actions (therefore in a relative way) in order to select the most promising one [22]. In particular, the entropy of the Dirichlet distribution is reflected in the entropy of the multinomial distribution that is sampled: the lower (higher) the entropy of the Dirichlet distribution, the lower (higher) the entropy of its expectation (the multinomial distribution result of the average sampling) [7]. When $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ with $\alpha_i = \alpha_j$ for all $i, j \in [1, k]$, the average multinomial distribution obtained by sampling $\boldsymbol{\theta} \sim Dir(\boldsymbol{\alpha})$ is uniform, therefore carrying the maximum entropy value (and consequently uncertainty).

To distinguish between a *passive* and a *reactive* execution trace, hence to decide if a malware execution trace is actually a reactive response to an analyzer action, we employ the Kullback-Leibler divergence D_{KL} [11].

Definition 3.4 (Kullback-Leibler divergence). The Kullback-Leibler divergence is a measure of how one probability distribution is different from a second reference probability distribution

$$D_{KL}(P \parallel Q) = -\sum_{x \in \mathcal{X}} P(x) \log \left(\frac{Q(x)}{P(x)} \right)$$

The Kullback-Leibler divergence D_{KL} measures how much a distribution is different from another in terms of the quantity of information that is lost when one distribution is approximated with the other. We use it to compute the difference between the distribution of the known passive execution trace of a malware sample and the distribution of another execution trace.

4 PROPOSED METHODOLOGY

Our proposed methodology, BAMA, is to the best of our knowledge the first to consider the link between malware families and the notion of types in Bayesian games. The goal is to be able to match an unknown malware sample to the family (class) in which it belongs, by performing as few analyzer triggering actions as possible. In order to achieve this goal, we make use of a priori information about the malware families that BAMA employs, instead of requiring underlying models to reason about as in [19, 30]. Such information is represented as a list of all the malware families that respond to a specific analyzer triggering action. Indeed, nowadays almost every repository of malware reports detailed information for every family, including the triggering mechanisms, such as the one we use in the experiments [29]. Furthermore, in our application domain (Android systems), the triggers we use are actions that an average smartphone user would perform, e.g., sending an sms, making a call, opening the browser etc., for which an application has to declare listeners in the manifest to intercept, thus easily obtainable by an analyzer. In contrast to previous works [15, 19], in order to devise the strategy for the analyzer to identify the malware type, we do not make use of detailed information, i.e., exact list of APIs, of an execution trace, but rather only distinguish between a passive execution trace that could have been observed also without interacting with the malware, and a reactive execution trace that is triggered by the specific action performed by the analyzer. Before giving the formalization of BAMA, we first explain the problems arising

[1] There are some technicalities that for brevity we do not report here, such as the fact that the support of a Dirichlet probability density function is actually the open $(k-1)$-dimensional simplex. We refer the interested reader to [17] for more details.

from the uncertainty on the observations of malware behaviors as execution traces, and how to solve them.

Our approach makes use of an Android sandbox emulator to run malicious applications, to execute analyzer triggering actions, and to observe and extract the corresponding execution traces. However, uncertainty affects the readings of the execution traces from the sandbox due to several reasons. The first is given by the intrinsic slow nature of AMA: the emulator has to be reset to a clean state and rebooted after every interaction, hence requiring time. After executing a triggering action, the analyzer waits for a fixed amount of time before registering the malware response.[2] This is motivated by the fact that in Android malicious applications it is often the case that if a malware is reactive, it will probably respond in a reasonable time after having been triggered. Nevertheless, for this reason the very same execution trace can be "cut" in different places when read multiple times as a response to a trigger. The second reason of uncertainty is that malware also often employ deception techniques that intentionally inject noise (random or non-correlated APIs) within execution traces, to deceive an analyzer by trying to hide their malicious behavior [14, 20]. In addition, a malware could have been designed to activate the payload in response to a specific trigger with some probability, instead of being deterministic. Finally, malware samples belonging to the same family may differ, possibly not responding to a trigger that is instead common to other samples of the same family, due to code repackaging and other changes inserted by criminals that often modify existing malware rather than creating them from scratch [26]. Hence, based on system workload, timing constraints, deception techniques, and other factors, different execution traces from the same malware could be retrieved in response to the same triggering action, complicating the comparison.

As mentioned, we are only interested in distinguishing between passive and reactive responses without analyzing in detail the content of the execution traces. A passive response is extracted before starting the game by simply executing a malware and observing its behavior without any interaction. Thus, we employ the Kullback-Leibler divergence D_{KL} of Definition 3.4 between the distribution of APIs P of the passive execution trace, and the distribution of APIs Q of another execution trace. In particular, we compute a threshold value ϵ on a training set of Android applications (both benign and malicious) by randomly executing triggering actions and measuring the mean value of D_{KL} between the passive and the reactive responses. Given P and Q, if $D_{KL}(P \parallel Q) \geq \epsilon$ the trace from which Q has been extracted is considered reactive, otherwise it is considered passive.

4.1 BAMA formalization

The goal of our proposed technique is to analyze a malware by repeatedly interacting with it in order to infer its type. Thus, BAMA is a game that we model from the point of view of the analyzer, meaning that it reflects how the analyzer sees the whole process. In particular, the utility function is designed from an information-centric perspective aimed at guiding the analyzer in acquiring information on the type of the adversary faced during the game.

[2]The amount of time is a parameter of the analysis depending on the host system. Usually varies between 10 and 30 seconds.

Definition 4.1 (BAMA game). The game of BAMA is a Bayesian game with

- $N = \{n_1, n_2\}$ where n_1 is the analyzer and n_2 is the malware
- $\boldsymbol{A} = A_1 \times A_2$ where
  - $A_1 = \{t_1, ..., t_m\}$ are all the possible triggering actions for the analyzer ($call$, $wifi$, etc.)
  - $A_2 = \{passive\,(p), reactive\,(r)\}$ consists of a passive execution trace or a reactive response to a trigger for the malware
- $\boldsymbol{\Theta} = \Theta_1 \times \Theta_2$ where
  - $\Theta_1 = \{\theta_1\}$ the fixed type of the analyzer
  - $\Theta_2 = \{f_1, ..., f_k\}$ where f_j with $j = 1, \ldots, k$ is a malware family
- $\boldsymbol{u} = (u_1, u_2)$ is a profile of utility functions with $u_1 : \boldsymbol{A} \times \boldsymbol{\Theta} \rightarrow \mathbb{R}_{\leq 0}$ utility function for the analyzer
- $p = Dir(\boldsymbol{\alpha})$ is the Dirichlet prior over $\boldsymbol{\Theta}$

BAMA is clearly an instance of a Bayesian game of Definition 3.1 between two players: the analyzer n_1 and the malware n_2. The action set A_1 available to the analyzer comprises all the possible triggering actions that can be performed on the system (send/receive an sms, make/receive a call, enable/disable wifi, etc.) and that could possibly cause a reaction in the adversary. The malware action set A_2 instead is an abstraction over all the concrete actions that can be observed, i.e., the execution traces, that are grouped in either $passive$ or $reactive$. The type of the analyzer θ_1 is fixed, whereas for the type set Θ_2 available to the malware we build a one-to-one correspondence with the possible malware families. The player's type θ_i encodes all the relevant private information for player i that in our context maps to how a malware responds to the possible analyzer triggering actions. Since $|\Theta_2| \geq 2$, giving a multinomial probability distribution as uncertainty measure over the types, our choice for prior p is the Dirichlet distribution, which is the conjugate of the multinomial distribution. The initialization of p can either be a uniform distribution or else reflect the distribution of the families in the dataset (or in the wild). The analyzer uses the prior p to reason about the next action to play during the game, updating it accordingly to the observation of the outcomes. The utility function is explained in Section 4.3 since it is based on the prior update process, therefore we first present that in Section 4.2 for a better understanding.

BAMA is intended to be played as a repeated game in multiple stages. In detail, every time a malware sample has to be analyzed, the analyzer starts a BAMA game of length n, i.e., of n stages in total. At each stage l, the analyzer selects a triggering action, observes the malware response, obtains the reward, updates the prior p into p' accordingly and moves to stage $l + 1$ against the same malware sample but with the new prior (posterior) p'. At the end of stage n, the prior p is reset to its initial distribution and the analysis process starts again with a new malware sample from stage 1. Figure 2 depicts a BAMA stage game. Formally, since the type of the malware sample is initially unknown, we assume its type is drawn by nature at stage 1 and remains fixed for the next n stages (or rather that nature always draws the same type for n stages), until the game resets to stage 1 again.

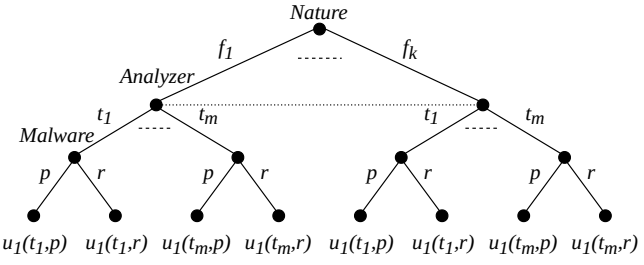

Figure 2: A stage of the BAMA game

4.2 Prior update

The Dirichlet prior is updated into a posterior by adding to parameters $\boldsymbol{\alpha}$ (a vector of pseudo-counts) the count of the new observations per class [17]. However, in contrast to many classical instances of Bayesian games, after receiving a reward we are still uncertain about the type of the adversary since more families can share the same triggers, requiring observations to be treated accordingly. We first define a function $g()$ that maps each analyzer triggering action *to the set of families that are known to respond to it* based on the a priori available information:

$$g : A_1 \rightarrow \mathcal{P}(\Theta_2) \tag{1}$$

Given a prior p with parameters $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ at stage l, the posterior p' with parameters $\boldsymbol{\alpha'} = (\alpha'_1, \ldots, \alpha'_k)$ at stage l+1 depends on the action a_2 of the malware after the analyzer played action a_1 at l. The α_j's are one per malware family f_j. The prior update is performed via function $w()$:

$$w(\boldsymbol{\alpha}, a_1, a_2) = \boldsymbol{\alpha'}$$

where

$$\alpha'_j = \begin{cases} \alpha_j + \frac{1}{|g(a_1)|} & f_j \in g(a_1) \wedge a_2 = reactive \\ \alpha_j + \frac{1}{|g(a_1)|} & f_j \notin g(a_1) \wedge a_2 = passive \\ \alpha_j & \text{otherwise} \end{cases} \tag{2}$$

with $1 \leq j \leq k$ and $f_j \in \Theta_2$. That is, if a malware actively responds to a triggering action a_1, i.e., $a_2 = reactive$, we split the observation across all the families that are known to respond to a_1, which are given by $g(a_1)$. Conversely, if a malware does not respond to a_1, i.e., $a_2 = passive$, we split the observation across all the families that are known to not respond to a_1. Notice that, as explained before, given the uncertainty in the readings from the emulator, we have to stay conservative as an execution trace may be detected as *passive* when it was *reactive* instead or vice versa. As such, we never force α'_j to be close to 0 as this can result in wrong inference in the remainder of the game.

4.3 Utility function

The key point of our formalization is the utility function u_1 for the analyzer. This is designed with the aim of guiding the selection of triggering actions so as to maximize the information acquired on the type of the adversary. The current belief on the adversary type is encoded by the prior $p = Dir(\boldsymbol{\alpha})$, and the multinomial distribution $\boldsymbol{\theta} \sim Dir(\boldsymbol{\alpha})$ has an uncertainty degree tied to the entropy $H_D(\boldsymbol{\alpha})$ of Definition 3.3. The reward received by the analyzer then is the entropy of the posterior obtained by updating p (with Equation 2) after $a_1 \in A_1$ and $a_2 \in A_2$ are performed:

$$u_1(a_1, a_2) = H_D(w(\boldsymbol{\alpha}, a_1, a_2)) \tag{3}$$

where $w()$ is computed with Equation 2. Essentially, the utility is a function that corresponds to the amount of uncertainty on the type of the adversary resulting after the joint actions (a_1, a_2) have been played, hence on how the prior (and consequently the belief on the type of malware) changes due to an analyzer action. Our formalization allows to avoid the need to explicitly capture every single factor that contributes to the uncertainty (time, deception, etc., as mentioned in the first part of Section 4) by abstracting between passive and reactive responses and maintaining a prior probability distribution used in the utility function of the analyzer.

4.4 Analyzer strategy

BAMA is a game where the analyzer's end goal is to infer the type of the adversary. Indeed, knowing with enough confidence what is the type of the current adversary, allows the analyzer to pick the correct actions in order to trigger the malware and observe its malicious behavior in response. Nonetheless, in the context of malware analysis it is crucial to characterize the payload of a malware and this is often done inferring its behavioral category with respect to other known malicious samples. However, our aim is not to reach full code coverage, i.e., to enumerate all the reactive traces, but rather to acquire enough information that allows us to correctly classify a malware into its family, and consequently to infer its behavior by similarity to others of the same category. If a malware family has 5 known triggers, and after executing 2 of them the prior points precisely to that family, i.e., low uncertainty in the resulting multinomial distribution, it is useless to perform also the other 3 triggers and observe the corresponding new traces.

As such, we devise the analyzer strategy with the aim of reducing the entropy in the prior p at every stage. The key point is that after performing a trigger a_1, the analyzer acquires information on the response that changes the prior accordingly (Equation 2). Since the aim is to reduce the uncertainty on adversary type when sampling the prior, we base the analyzer strategy on the entropy minimization principle for p, employing a 1-step lookahead that considers the two known possible updates from p at stage l to p' at stage $l+1$ (passive or reactive response). Thus, the analyzer employs its current belief on the type of adversary, and chooses the action that reduces the entropy of p' the most. Formally, action selection starts with a sampling of the current prior p giving $\boldsymbol{\theta} \sim Dir(\boldsymbol{\alpha})$, and then picks an entropy minimizing action, as follows:

$$\operatorname*{argmin}_{a_1 \in A_1} [q \cdot u_1(a_1, reactive) + (1 - q) \cdot u_1(a_1, passive)]$$

that is

$$\operatorname*{argmin}_{a_1 \in A_1} [q \cdot H_D(w(\boldsymbol{\alpha}, a_1, reactive)) + (1 - q) \cdot H_D(w(\boldsymbol{\alpha}, a_1, passive))] \tag{4}$$

with

$$q = \sum_{f_j \in g(a_1)} \theta_j$$

where $w()$ is defined in Equation 2 and H_D is the entropy of the Dirichlet from Definition 3.3. The value of q sums up to the probability for the adversary to respond to the analyzer triggering action a_1, based on the current prior p and the information on the families and their triggers (function $g()$ of Equation 1). Equation 4 then corresponds to selecting the action that minimizes the expectation over H_D at the next stage. Now, if $g()$ is imprecise, the analyzer may not be able to pick actions that effectively lower the entropy of the Dirichlet prior, with a negative impact on the final results. Nonetheless, the amount of wrong mappings inside $g()$ has to be consistent for BAMA to be ineffective, since (as explained at the beginning of Section 4) uncertainty is already taken into account by the formalization.

After a BAMA game ends, the state of the prior p should reveal the type of the adversary the analyzer has been confronting. However, the very same set of triggers may be shared among multiple malware families, making them to be seen as the same type in our formalization. For this reason, we make use of Markov chain based models (Figure 1) as a tie breaker. Specifically, such a model is generated using the execution traces observed while playing BAMA, but it is not considered at all during the game: it is only used at the end to clear the uncertainty in the prior, if so required. Moreover, building such models allows us to compare with state-of-the-art techniques in terms of model classification score, as detailed in Section 5.

Algorithm 1 BAMA Analysis

Require:
$p = Dir(\boldsymbol{\alpha})$ - prior over Θ_2
n - game length
ϵ - threshold value for D_{KL}

1: Retrieve distribution P of passive trace
2: **for** n times **do**
3: Sample $\boldsymbol{\theta} \sim Dir(\boldsymbol{\alpha})$
4: Select action a_1 with Equation 4 using $\boldsymbol{\theta}$
5: Execute a_1 and retrieve distribution Q of the trace
6: **if** $D_{KL}(P \parallel Q) \geq \epsilon$ **then**
7: $a_2 \leftarrow reactive$
8: **else**
9: $a_2 \leftarrow passive$
10: $\boldsymbol{\alpha'} \leftarrow w(\boldsymbol{\alpha}, a_1, a_2)$ ▷ Update with Equation 2
11: Update p with $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha'}$

Algorithm 1 details the BAMA analysis. The first step (line 1) retrieves the distribution of the passive trace for the computation of D_{KL} later. At this point the game begins by sampling the prior p for action selection (lines 3-4). The selected action a_1 is then executed on the emulator and the subsequent execution trace of the malware is retrieved along with its distribution Q (line 5). Based on the value of $D_{KL}(P \parallel Q)$ with respect to the threshold ϵ, action a_2 of the malware is assigned as *passive* or *reactive* (lines 6-9). Finally, the prior parameters $\boldsymbol{\alpha}$ are updated according to the outcome, and the game progresses to the next stage (lines 10-11). At every stage l the analyzer selects the action that minimizes the entropy H_D of the prior p at stage $l + 1$. Reducing the entropy at every stage achieves the result of reducing the uncertainty on the type of adversary when sampling the prior, as confirmed by experiments and visible in Figure 3.

5 EMPIRICAL EVALUATION

In the experimental setting we compare BAMA with other three AMA techniques: MYOPIC [30], MCA [19] implemented in the SECUR-AMA framework [21], and CANDYMAN [15]. The first step to analyze an unknown application is to conduct a preliminary analysis to assess whether it could be malicious [1, 13]. If that is the case, a crucial task becomes the identification of the specific malware family in order to use possibly already known countermeasures to defend against the threat. All the techniques tested in the experiments share the end goal of identifying the family of an unknown malicious application by employing different strategies for the analyzers (as previously detailed).

We use the same dataset of [29] in a subset composed of about 1400 real Android malware partitioned into 24 families. For instance, the family Finspy concerns the logging and exfiltration of personal information of the user on an Android device, thus it is sensitive to calls, SMS activities, browser navigation history updates, etc. Furthermore, some of the families included in this experiment can be seen as challenging to correctly classify, since they employ specific mechanisms aimed to deceive the analysis. In particular, Gorpo and Kemoge employ a combination of anti-analysis techniques such as the dynamic loading of the malicious code at runtime and the execution of noisy unrelated API calls, i.e., that are not useful to implement the malware payload but serve as a method to mislead an analyzer that focuses on the sequence of actions performed by the malicious sample. Hence, behaviors related to the damaging payload interweaved by noisy APIs can induce an analyzer to overlook malicious characteristics. Moreover, AndroRat and GoldDream families distinguish themselves on the type of infection vector, as they are composed by small malware injected into complex harmless applications such as games. This peculiar feature causes only a small portion of the observed execution traces to depict malicious behaviors, while the rest being related to the harmless application that has been injected, thus making the malware identification hard. Malware samples belonging to Opfake are designed to receive commands from an external server controlled by an attacker in order to be triggered and show their malicious behavior. This happens also for samples of Tesbo that additionally also try to hide themselves by not having a GUI for the user to see. The rest of the families involved in the dataset can be considered less sophisticated because they do not employ advanced anti-detection mechanisms and do not hide themselves through injection into other applications.

The dateset employed for the experiments contains families with more than 200 samples and others with as few as 5. Such characteristic reflects how malicious software appear in the real world in relation to their families: recently, the focus in malware design has shifted from the creation of new types of malicious payloads from scratch, i.e., the code slice of a malware aimed at causing harm, to the engineering of the stealthy system[3], while the payload is reused from older deployed malware as is or with minor modifications. As a consequence of this trend, most of the malware

[3]The code of a malware that makes it to remain undetected during its execution.

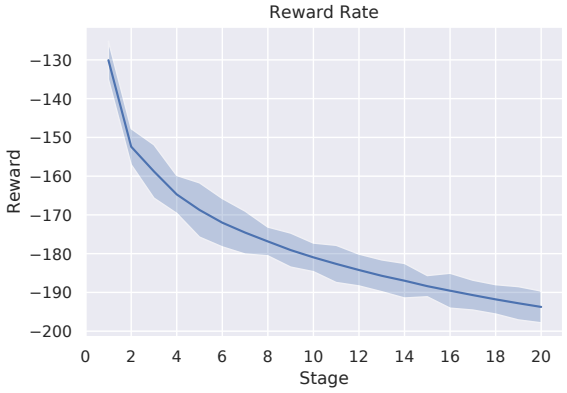

Figure 3: Rewards obtained over time in a BAMA game

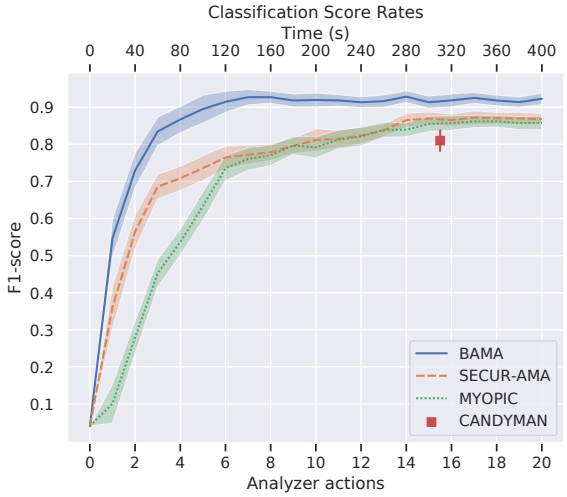

Figure 4: Comparison of classification score rates

in the wild fall in few families with unbalanced proportions [27, 28]. The detailed report available for each family has been used to build function $g()$ of Equation 1 with the a priori information about which families respond to which triggers; $g()$ is used then in turn to construct the initial Dirichlet prior for our experiments.

Results in Figure 3 show that rewards, i.e., the entropy of the posterior Dirichlet distribution, clearly decrease at each stage of BAMA, hence the analyzer progressively reduces the uncertainty on the sampling of the type of adversary. Next, we perform a comparison by classifying, as in previous works, the transition matrices of the Markov chain models (Figure 1) obtained after every step of the analysis for each of the techniques employed in the experiments. In particular, MCA outputs such model by default, we apply the same model generation method to the MYOPIC algorithm, and we build the model also while playing BAMA as explained before (without using it in any way during the analysis). CANDYMAN instead outputs models of the same kind but without the conditioning of the probabilities based on the analyzer action, i.e., the model is composed by a single Markov chain. This allows us to fairly compare the different AMA techniques as they all share the same type of output model. The set of triggering actions for the analyzer is the same for BAMA, MCA and MYOPIC, and is composed of 17 different actions that mimic a standard user's behavior: *send/receive sms, make/receive call, switch on/off Wifi/GPS/screen, charge/discharge battery, add/remove contact, install/remove app, set clock*. CANDYMAN instead uses different triggering actions that are related to the GUI of every specific application, randomly selecting up to 5000 of such actions in 5 minutes. Therefore, we are able to show the classification score rate of BAMA, MCA and MYOPIC, as the progression in terms of analyzer actions performed is comparable; while for CANDYMAN we can only show the score after the analysis is complete, since the number and type of triggering actions are of different nature.

We employed a Stratified K-Fold Cross Validation with $K = 5$, while the quality of results is assessed with unweighted standard measures, i.e., precision, recall, and F_1-score.[4] Implementations of the classifier, i.e., a linear Support Vector Machine (SVM),[5] quality measures and cross-validation make use of Scikit-Learn [18].

Figure 4 shows the classification score rates in term of F_1-score obtained. It is clearly visible that BAMA learns faster compared to the other techniques: it only requires 4 analyzer actions (about 80 seconds) to reach the same best overall classification score (0.87) of MCA. Furthermore, BAMA also reaches the highest global classification score (0.92), due to the fact that by using the information on the adversary type, final malware models contain less noise, i.e., fewer Markov chains associated to reactions to triggers that are not meaningful for that malware, augmenting the classification process. The difference between BAMA and the other techniques is statistically significant according to a Student's paired two-tailed t-test with $p < 0.05$. It is clear that BAMA improves the strategy of the analyzer: it allows her to pick a sequence of actions that is shorter and more precise compared to the other techniques in order to generate a good malware model based on the observation of the traces. The SVM classifiers used on the models extracted by MCA, MYOPIC and CANDYMAN reach a worse classification score than when BAMA is used to create the models. Therefore, the BAMA decision-making strategy is clearly of value in terms of results. Table 1 shows the best overall F_1-score classification results for the best instance of each technique: 8 actions in 160 seconds for BAMA, 14 actions in 280 seconds for MCA, 18 actions in 360 seconds for MYOPIC, and 310 seconds for CANDYMAN.

Results show that BAMA allows us to perform AMA more effectively, not only with respect to the final average overall classification score but, most importantly, in terms of speed (number of analyzer actions required to reach that score), which is crucial in malware analysis since security firms have to analyze a huge amount of new malware discovered every day; thus, reducing the number of analyzer actions required to identify a malware has a big impact on the overall analysis time.

[4] Due to space constraints we only report the values for the F_1-score since it incorporates both precision and recall measures.

[5] We also tested other classifiers but the linear SVM gives the best results.

Table 1: Best overall classification results

	BAMA	MCA	MYOPIC	CANDYMAN
Analyzer actions	**8**	14	18	-
Seconds	**160**	280	360	310
F_1-score	**0.92**	0.87	0.86	0.81

Although BAMA performs better overall when compared to the other techniques, it does not perform better for every malware family in the dataset. Table 2 details the per-family F_1-score for the best instance of each technique tested, i.e., the BAMA, MCA, MYOPIC, and CANDYMAN instances reported in Table 1. In the case of AndroRAT and GoldDream for example, the BAMA entropy minimization of the prior strategy is able to overcome the injection problem (mentioned at the beginning of this section) by executing a limited subset of triggers that are specific to the injected malicious part of the application, allowing it at the same time to identify the correct family. For the other techniques that instead rely on the model for their decision making strategy (whereas BAMA does not use it at all), the presence of a small malicious behavior within a bigger harmless application is harder to detect. The anti-analysis routines employed by the Kemoge family make the samples load the code at runtime when triggered by specific actions on the GUI: since CANDYMAN triggering mechanism is specifically designed around the GUI, it is more effective in stimulating the samples belonging to such family. On the other hand, since Tesbo does not have a GUI, CANDYMAN is not able to trigger any behaviors from such samples. Nevertheless, all the techniques perform badly on Tesbo because of the code coverage problem of dynamic analysis: some malicious behaviors are triggered by commands coming from an external server, therefore they are never exhibited without designing application specific triggers (this is the case also for the Opfake family). A main reason for BAMA to perform worse with some families lies in the stochasticity in the retrieval of malware responses since the threshold value ϵ for the Kullback-Leibler divergence may result in a wrong identification of the trace type and hence impact on trigger selection. Another reason for errors in classification comes from the fact, explained in Section 4, that a specific malware sample may respond not only to the triggers for which its family is known to respond to, but also to triggers of other families as well. Conversely, a malware sample could also not respond to some triggers listed for its family. All these factors contribute to making the problem of malware analysis really complex, and consequently the classification task difficult to solve perfectly with any particular technique.

Table 2: Per-family F_1-score classification for the best instance of each technique as reported in Table 1

Family	BAMA	MCA	MYOPIC	CANDYMAN
AndroRAT	**0.93**	0.84	0.84	0.85
Boqx	0.95	**0.96**	0.93	0.90
Cova	**0.97**	0.94	0.89	0.92
FakeAV	0.89	0.89	0.89	0.89
FakeDoc	1.00	1.00	1.00	0.98
Finspy	1.00	1.00	1.00	1.00
Fjcon	**0.94**	0.87	0.79	0.55
GoldDream	**0.94**	0.92	0.87	0.68
Gorpo	**0.87**	0.81	0.82	0.81
Kemoge	0.70	0.44	0.35	**0.76**
Kuguo	**0.94**	0.93	0.91	0.84
Leech	0.99	**1.00**	0.98	0.98
Mseg	**0.99**	0.98	0.97	0.93
Obad	1.00	1.00	1.00	1.00
Opfake	**0.75**	0.63	0.67	0.57
SmsZombie	1.00	1.00	1.00	1.00
SpyBubble	**0.95**	0.63	0.46	0.36
Stealer	1.00	1.00	1.00	1.00
Svpeng	1.00	1.00	1.00	0.96
Tesbo	**0.57**	0.33	0.57	0.00
Triada	**0.91**	0.81	0.84	0.74
Vidro	0.96	1.00	1.00	1.00
Vmvol	1.00	1.00	1.00	0.92
Winge	**1.00**	0.88	0.91	0.76

6 CONCLUSIONS

We propose BAMA, a novel technique for dynamic malware analysis formalized as a Bayesian game between an analyzer and a malware agent. To guide the analyzer we design a utility function that expresses the amount of uncertainty on the type of the adversary after the execution of an action. The algorithm devised to play BAMA aims at minimizing the entropy of the analyzer's belief, a Dirichlet prior, at every stage of the game in a myopic fashion. Experiments on a dataset of real Android malware show that, when compared to other state-of-the-art techniques, BAMA requires fewer actions (and consequently time) to reach a satisfying classification score for malware identification. As such, our approach paves the way for using Bayesian malware analysis in a large and significant scale.

In future work we aim to evaluate how different levels of wrong information inside $g()$ impact the BAMA process in terms of classification results. Furthermore, we plan to also consider malware that actively tries to counter a strategic, e.g., a BAMA, analyzer. By this we do not mean malware that use simple anti-emulation mechanisms, but rather malware that, on top of the usual goal and ability to release the payload, also strategize to counter an analyzer's attempts to reveal their family. Of course, malware of such level of sophistication are not common yet in the real world, but the existing few are arguably among the most dangerous ones. Our formalization allows the easy modeling of such malware, requiring only the careful design of the utility function so that it accurately represents the adversary goals and abilities. Other future directions include the application of our approach to various fields of cyber-security, e.g., network security and honeypot configuration.

ACKNOWLEDGMENTS

The research reported in this publication has been partially supported by the project "Dipartimenti di Eccellenza 2018-2022" funded by the Italian Ministry of Education, University and Research (MIUR).

REFERENCES

[1] Yousra Aafer, Wenliang Du, and Heng Yin. 2013. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. In *Security and Privacy in Communication Networks*, Tanveer Zia, Albert Zomaya, Vijay Varadharajan, and Morley Mao (Eds.). Springer International Publishing, Cham, 86–103.

[2] Georgios Chalkiadakis and Craig Boutilier. 2007. Coalitional Bargaining with Agent Type Uncertainty. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI'07)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1227–1232. http://dl.acm.org/citation.cfm?id=1625275.1625474

[3] Georgios Chalkiadakis, Evangelos Markakis, and Craig Boutilier. 2007. Coalition Formation Under Uncertainty: Bargaining Equilibria and the Bayesian Core Stability Concept. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07)*. ACM, New York, NY, USA, Article 64, 8 pages. https://doi.org/10.1145/1329125.1329203

[4] Matthew Cheung. 2018. Market Share: Operating Systems, Worldwide, 2017. https://www.gartner.com/doc/3879167/market-share-operating-systems-worldwide. (6 2018). Gartner, Inc.

[5] Nader Ebrahimi, Ehsan S. Soofi, and Shaoqiong (Annie) Zhao. 2011. Information Measures of Dirichlet Distribution with Applications. *Appl. Stoch. Model. Bus. Ind.* 27, 2 (March 2011), 131–150. https://doi.org/10.1002/asmb.870

[6] C.C. Elisan. 2015. *Advanced Malware Analysis*. McGraw-Hill Education. https://books.google.it/books?id=17SUAwAAQBAJ

[7] Piotr Garbaczewski. 2006. Differential Entropy and Dynamics of Uncertainty. *Journal of Statistical Physics* 123, 2 (14 April 2006), 315. https://doi.org/10.1007/s10955-006-9058-2

[8] John C. Harsanyi. 1967. Games with Incomplete Information Played by "Bayesian" Players, I–III Part I. The Basic Model. *Management Science* 14, 3 (1967), 159–182. https://doi.org/10.1287/mnsc.14.3.159 arXiv:https://doi.org/10.1287/mnsc.14.3.159

[9] Manish Jain, James Pita, Milind Tambe, Fernando Ordóñez, Praveen Paruchuri, and Sarit Kraus. 2008. Bayesian Stackelberg Games and Their Application for Security at Los Angeles International Airport. *SIGecom Exch.* 7, 2, Article 10 (June 2008), 3 pages. https://doi.org/10.1145/1399589.1399599

[10] Xinyu Jin, Niki Pissinou, Sitthapon Pumpichet, Charles A. Kamhoua, and Kevin A. Kwiat. 2013. Modeling cooperative, selfish and malicious behaviors for Trajectory Privacy Preservation using Bayesian game theory. In *38th Annual IEEE Conference on Local Computer Networks, Sydney, Australia, October 21-24, 2013*. 835–842. https://doi.org/10.1109/LCN.2013.6761339

[11] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.

[12] Xinxin Liu, Kaikai Liu, Linke Guo, Xiaolin Li, and Yuguang Fang. 2013. A game-theoretic approach for achieving k-anonymity in Location Based Services.. In *INFOCOM*. IEEE, 2985–2993. http://dblp.uni-trier.de/db/conf/infocom/infocom2013.html#LiuLG0F13

[13] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J. Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *NDSS*. The Internet Society.

[14] J. A. P. Marpaung, M. Sain, and Hoon-Jae Lee. 2012. Survey on malware evasion techniques: State of the art and challenges. In *2012 14th International Conference on Advanced Communication Technology (ICACT)*. 744–749.

[15] Alejandro Martín, Víctor Rodríguez-Fernández, and David Camacho. 2018. CANDYMAN: Classifying Android malware families by modelling dynamic traces with Markov chains. *Engineering Applications of Artificial Intelligence* 74 (2018), 121 – 133. https://doi.org/10.1016/j.engappai.2018.06.006

[16] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Exploring Multiple Execution Paths for Malware Analysis. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP '07)*. IEEE Computer Society, Washington, DC, USA, 231–245.

[17] K.W. Ng, G.L. Tian, and M.L. Tang. 2011. *Dirichlet and Related Distributions: Theory, Methods and Applications*. Wiley. https://books.google.it/books?id=k8GS868oyo4C

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[19] 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

[20] Riccardo Sartea, Alessandro Farinelli, and Matteo Murari. 2019. Agent Behavioral Analysis Based on Absorbing Markov Chains. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 647–655. http://dl.acm.org/citation.cfm?id=3306127.3331752

[21] Riccardo Sartea, Alessandro Farinelli, and Matteo Murari. 2020. SECUR-AMA: Active Malware Analysis Based on Monte Carlo Tree Search for Android Systems. *Engineering Applications of Artificial Intelligence* 87 (2020), 103303. https://doi.org/10.1016/j.engappai.2019.103303

[22] K. Sobczyk. 2001. Information Dynamics: Premises, Challenges and Results. *Mechanical Systems and Signal Processing* 15, 3 (2001), 475 – 498. https://doi.org/10.1006/mssp.2000.1378

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

[24] Milind Tambe. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned* (1st ed.). Cambridge University Press, New York, NY, USA.

[25] Wiem Tounsi and Helmi Rais. 2018. A Survey on Technical Threat Intelligence in the Age of Sophisticated Cyber Attacks. *Comput. Secur.* 72, C (Jan. 2018), 212–233. https://doi.org/10.1016/j.cose.2017.09.001

[26] Jason Upchurch and Xiaobo Zhou. 2016. Malware Provenance: Code Reuse Detection in Malicious Software at Scale. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*. 1–9. https://doi.org/10.1109/malware.2016.7888735

[27] Andrew Walenstein and Arun Lakhotia. 2006. The Software Similarity Problem in Malware Analysis. In *Duplication, Redundancy, and Similarity in Software*.

[28] Andrew Walenstein, Michael Venable, Matthew Hayes, Christopher Thompson, and Arun Lakhotia. 2007. Exploiting Similarity Between Variants to Defeat Malware " Vilo " Method for Comparing and Searching Binary Programs.

[29] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep Ground Truth Analysis of Current Android Malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17)*. Springer, Bonn, Germany, 252–276.

[30] 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

[31] Haifeng Xu, Rupert Freeman, Vincent Conitzer, Shaddin Dughmi, and Milind Tambe. 2016. Signaling in Bayesian Stackelberg Games. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS '16)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 150–158. http://dl.acm.org/citation.cfm?id=2936924.2936950