# Agent Behavioral Analysis Based on Absorbing Markov Chains

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

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

## ABSTRACT

We propose a novel technique to identify known behaviors of intelligent agents acting within uncertain environments. We employ Markov chains to represent the observed behavioral models of the agents and we formulate the problem as a classification task. In particular, we propose to use the long-term transition probability values of moving between states of the Markov chain as features. Additionally, we transform our models into absorbing Markov chains, enabling the use of standard techniques to compute such features. The empirical evaluation considers two scenarios: the identification of given strategies in classical games, and the detection of malicious behaviors in malware analysis. Results show that our approach can provide informative features to successfully identify known behavioral patterns. In more detail, we show that focusing on the long-term transition probability enables to diminish the error introduced by noisy states and transitions that may be present in an observed behavioral model. We pose particular attention to the case of noise that may be intentionally introduced by a target agent to deceive an observer agent.

**ACM Reference Format:**
Riccardo Sartea, Alessandro Farinelli, and Matteo Murari. 2019. Agent Behavioral Analysis Based on Absorbing Markov Chains. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

## 1 INTRODUCTION

Many practical problems of interest can be represented as systems where intelligent agents interact within complex uncertain environments, gathering information and adapting their behaviors accordingly, e.g., security, robotics, entertainment and so forth. In such scenarios, an important issue is to identify whether a known behavior appears within a behavioral model that has been learned through observations [2, 8]. A key challenge in this context is to deal with the presence of noise, e.g., while performing a difficult task an agent might make mistakes trying to follow its policy, consequently injecting noise into the behavioral model learned by observing the execution of that task. Moreover, noise injection could be intentional, e.g., a malicious agent might try to mask its real intentions and to deceive potential observers [11].

In this work we aim at improving the identification of behavioral models performed by an *analyzer* agent that can interact with and observe the effect of the actions taken by a *target* agent which follows an unknown policy. The main hypothesis of our approach is that in the long-term, the intended behavior of the target agent will emerge, hence allowing the analyzer to filter out possible misleading observations. Following relevant literature [1, 10, 13, 19, 21, 28], we represent the behavior of the target agent by using Markov chains. However, a novelty of our work is to propose the use of the long-term transition probability of the Markov chain as features to identify the target agent's behavior. With long-term transition probability, we refer to the probability values of going from each state to every other, giving the process represented by the Markov chain enough time to reach a fixpoint, i.e., when the result would not change anymore from that point onward. We design our methodology in order to make it applicable independently of the technique used to obtain the behavioral models of the target agents (as long as they can be interpreted as Markov chains). Consequently, we can not make any assumptions on the various properties that the Markov chains may hold. A crucial technical difficulty related to this idea is the computation of the long-term transition probability values as features for generic Markov chains. In fact, while there are methods to compute different long-term characteristics, e.g., the stationary distribution or the time to absorption [9], their application is subject to the presence of specific properties, irreducibility and absorbency respectively, that generic Markov chains may not have. For example, in our experiments we have large models that are generated through the observation of an agent's policy. For such models, the absorbency property, which is a fundamental requirement to compute long-term transition probability, never holds. To overcome this problem we propose a transformation of the Markov chain that enforces the absorbency property and allows to derive the long-term transition probability for the original Markov chain.

Our approach is designed to be employed as an analysis tool following any behavioral model generation techniques when the presence of noise or deceitful behaviors should be considered. Hence, the focus of this paper is not to improve the process of obtaining the behavioral models (although obviously some techniques can give better final results for particular domains), but rather the approach used to analyze them after they have been generated. In more detail, we pose the identification problem as a classification task, where we are given a set of known models divided into classes, i.e., a set of labeled Markov chains, and a set of models representing the observed unknown behavior of the target agents, i.e., a set of unlabeled Markov chains. Our aim is to identify such unknown models assigning them to the known classes. The long-term transition probability values of moving between states are used as features to train a standard classifier, which is a Support Vector Machine (SVM) in our case. We conduct two sets of experiments: the first one aims at identifying known strategies within classical games, namely the iterated Prisoner's Dilemma and a repeated lottery game. The second one aims at identifying specific malicious behaviors in models representing dynamics of malicious software agents, i.e., malware.

*Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.* © 2019 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

The first experiment evaluates whether our approach can be used in a generic setting that involves the interactions of intelligent agents, whereas the second experiment focuses on the concrete cyber-security application of identifying malicious behaviors of software programs.

In summary, our contributions are the following:

(1) We consider the use of a classification approach to identify unknown behaviors w.r.t. a set of known classes, where behaviors are modeled as Markov chains. The long-term transition probability values of moving between states are employed as features for the task.
(2) We define a transformation for Markov chains enforcing the absorbency property. This allows us to use standard techniques to derive the long-term transition probability for generic Markov chains without requiring any specific properties.
(3) We empirically evaluate our approach on behavioral models of players interacting within classical games and of real malicious software agents respectively. Results show that our technique provides informative features suitable to successfully identify known behaviors in both empirical settings. In particular, this method allows to diminish the effect of noise injected into the behavioral models by malicious software agents, overcoming a limitation of the current state-of-the-art malware analysis techniques.

## 2 RELATED WORK

The main focus of this work is to analyze the behavioral models extracted by observing the effect of the actions performed by a target agent within a fully observable environment. Markov chains are particularly suited for such task and have been widely used in scenarios that range from reliability analysis of software programs [24] to HTTP traffic optimization [13, 21, 28]. However, the methods that such previous approaches propose to compare the different models either impose constraints on the models to be analyzed, e.g., [24] requires one Markov chain to be a sub-graph of the other, or perform transformations that are application specific, e.g., [13, 28] require cycles to be removed and arbitrary states to be grouped together, hence neglecting significant information for what concerns behavior identification. Other works [3, 4] provide methods to compare Markov chains relying on the mixing time and by directly computing or estimating the stationary distribution. However, the behavioral models we deal with are generated by generic techniques, hence we have no guarantee for the existence of a meaningful stationary distribution, as it requires specific properties to hold (see Section 3). An interesting work in the context of detecting opponents is [7], where authors propose a framework for stochastic games aimed at learning the policy of multiple unknown adversaries drawn from different populations. We differ from [7] as we explicitly take into account adversarial agents that may intentionally perform some random or completely unrelated actions in order to mask their real policy, injecting noise in the behavioral model. Consequently, an observer agent can be deceived if it does not consider such potential deviation during the analysis process.

An important domain where agents try to hide their real intentions is cyber-security. Real malicious software agents often employ anti-detection techniques that inject noise in their execution trace, making difficult for defense systems to detect such threats. In our empirical evaluation we consider also this real application context as it could benefit from our proposed approach. Markov chains have been employed before to represent malware models as well [1, 10]. Such methods are based on static analysis to extract the API call sequences, i.e., features are extracted without executing the program. However, static analysis suffers from some known drawbacks such as the difficulty in analyzing obfuscated or encrypted binaries and the inability to analyze code downloaded and executed at runtime. To overcome the mentioned problems, we focus on dynamic analysis, that is best suited for an interactive approach as the one we aim to. In this context, machine learning methods such as clustering or classification are commonly used to analyze threats. A key point in both schemes concerns how to extract informative features resulting in good learning performance. A proposed solution often recurring in literature is to use *n-grams* [18, 26], i.e., sequences of $n$ API calls. Even though good results can be achieved with such technique, a significant issue is the exponential space requirements when $n$ increases. Moreover, since n-grams are an approximation of atomic behaviors embedded in malware, it is difficult to decide the proper granularity degree of the information represented through such feature type, i.e., how to select a proper value for $n$. Now, classical dynamic analysis is *passive*, meaning that no interaction happens during execution between the analyzer and the target code [6, 14, 27]. As a consequence, malicious behaviors might be overlooked since it is often the case that a specific action is required to trigger them [15]. In recent years, there has been an increasing interest in *active* dynamic analysis. In [25] authors propose a game-theoretical framework called Active Malware Analysis (AMA), later applied to API call graphs in [20]. The analysis is formalized as a stochastic game where the analyzer tries to find the best action to perform on the system in order to trigger malicious reactions by the malware. In [19], AMA is improved by generating the malware model at runtime, and representing it with multiple Markov chains. The aim is to group the models with clustering techniques trying to obtain the same partitioning of the ground truth in terms of malware families. The features extracted by AMA are the probability values of transitioning between states of the Markov chains. Such representation however, prevents to clearly identify small malicious behaviors appearing within a bigger behavioral model or purposely injected noise to deceive the analyzer. Indeed, AMA is designed to be fast and so limited in time, hence it may not capture the behavior that might arise only in the long-term, as we aim to do instead.

## 3 BACKGROUND

### 3.1 Markov chain tools

Markov chains are formal models to represent fully observable states of a system with a random variable that changes over time according to some probability distribution. The following definitions and theorems, along with their proofs, can be found in [9].

*Definition 3.1 (Markov chain).* Let $P$ be a $k \times k$ matrix with elements $\{P_{ij} : i, j = 1, ..., k\}$. A random process $(X_0, X_1, ...)$ with finite space $S = \{s_1, ..., s_k\}$ is a Markov chain with transition matrix $P$ if for all $n$, all $i, j \in \{1, ..., k\}$ and all $i_0, ..., i_{n-1} \in \{1, ..., k\}$ we have

$$\mathbb{P}(X_{n+1} = j | X_0 = i_0, ..., X_{n-1} = i_{n-1}, X_n = i) = \mathbb{P}(X_{n+1} = j | X_n = i) = P_{ij} \tag{1}$$

Equation 3.1 expresses the Markov property, i.e., the conditional probability distribution of the next state depends only on the current one. Such assumption, even though not realistic in some cases, is an acceptable approximation in many application domains as in ours. From now onward we will identify a Markov chain with its transition matrix and we will also make use of the corresponding graph representation.

*Definition 3.2 (Irreducible Markov Chain).* A set of states is irreducible if it is possible to go from each state to any other in an arbitrary (finite) number of steps. A Markov chain is irreducible if it consists of a single irreducible set.

THEOREM 3.3 (STATIONARY DISTRIBUTION). *Given a Markov chain $P$, the vector $\pi$ such that $\pi P = \pi$ is the stationary distribution of $P$. For any finite, irreducible Markov chain, $\pi$ is unique.*

The stationary distribution $\pi$ represents the fraction of times a Markov chain will spend in each state when the number of steps $n$ becomes large, i.e., as $n \longrightarrow \infty$

*Definition 3.4 (Absorbing Markov Chain).* Given a Markov chain $P$, a state $s_i$ is absorbing if $P_{ii} = 1$, otherwise it is transient. A Markov chain is absorbing if at least one of its states is absorbing and if from every transient state an absorbing one will be eventually reached.

If we deal with an absorbing Markov chain, it is usually preferable to reorder the states in a canonical transition matrix in order to clearly identify whether they are transient or absorbing. In our case, the block decomposition given by the canonical form will be useful to isolate long-term characteristics of a model we are interested in studying.

*Definition 3.5 (Canonical form of an absorbing Markov chain).* If an absorbing Markov chain $P$ has $n$ transient states and $r$ absorbing states, its transition matrix can be rewritten as

$$P = \left[\begin{array}{c|c} Q & R \\ \hline \emptyset & I \end{array}\right]$$

where $Q$ is an $n \times n$ matrix of the transition probability between the transient states, $R$ is a $n \times r$ non-null matrix of the transition probability from the transient to the absorbing states, $\emptyset$ is a $r \times n$ null matrix, and $I$ is a $r \times r$ identity matrix.

LEMMA 3.6. *For any absorbing Markov chain in canonical form we have that $Q^k \longrightarrow 0$ as $k \longrightarrow \infty$.*

Lemma 3.6 is useful to derive Theorems 3.7 and 3.8 and will be extensively used in our methodology (Section 4).

THEOREM 3.7 (FUNDAMENTAL MATRIX OF AN ABSORBING MARKOV CHAIN). *The fundamental matrix $N$ of an absorbing Markov chain $P$ in canonical form is defined as*

$$N = I + Q^1 + ... + Q^k = \sum_{k=0}^{\infty} Q^k = (I - Q)^{-1}$$

*where each entry $N_{ij}$ represents the mean of the total number of times that the chain is in a given transient state $s_j$ if starting from the transient state $s_i$. The inverse of $(I - Q)$ is guaranteed to exist for every absorbing Markov chain.*

THEOREM 3.8 (TRANSIENT STATES PROBABILITY).

$$H = (N - I)N_{dg}^{-1}$$

*Each entry $H_{ij}$ represents the probability of reaching transient state $s_j$ starting from transient state $s_i$ before the process is completely absorbed. $N_{dg}$ is the diagonal of $N$.*

Note that the transient states probability (Theorem 3.8) is different from the stationary distribution (Theorem 3.3). In this paper we make use of the former in order to extract the long-term probability values of going from each state to every other.

### 3.2 Active Malware Analysis

In our work we employ the AMA technique described in [19] to extract the behavioral models of target agents that we then analyze with our proposed methodology. The goal of the analyzer agent is to learn as much information as possible on the target agent and to generate the corresponding behavioral model, representing it with Markov chains. The analyzer chooses its action to play at each stage, from the set of all possible actions, with a Monte Carlo Tree Search (MCTS) using an information-centric reward function based on entropy. Intuitively, from the behavioral model generated so far, the analyzer performs an action that is expected to lower the entropy of such model the most, gaining information on the target agent. The entropy is computed by using the probability values of the Markov chain. A behavioral model is represented with a set of Markov chains extracted from the observation of the target agent's actions. Each Markov chain represents the behavior of the target agent w.r.t. a specific action executed by the analyzer agent. The described technique takes its name from the application to malware analysis, but the interesting concepts employed to select the best actions to generate informative models based on what is observed can be easily generalized to other domains.

## 4 PROPOSED METHODOLOGY

The problem we face in this work is the following: we are given a set of known behavioral models $K$ partitioned into a set of classes $C$, and a set of unknown (unlabeled) behavioral models $B$ of target agents. All models are represented with Markov chains and have been generated by observing the changes in the environment as a consequence of the interaction between an analyzer agent and a target agent. Our goal is then to assign the unknown elements of $B$ to the known classes of $C$. The solution we propose is to employ supervised learning techniques to train a classifier for $B$ given $K$. The proposed approach is explicitly designed for situations where an agent reacts to stimuli provided by some other agent. This is typical of adversarial environments and fits well with several practical scenarios e.g., generic adversarial games, malware analysis, but we also apply our methodology to non-adversarial settings such as single player lottery game (see Section 5). Now, the probability values of transitioning between every state that are specified by the transition matrix of the Markov chains could be used as features to train the classifier. However, such features may be unreliable as they

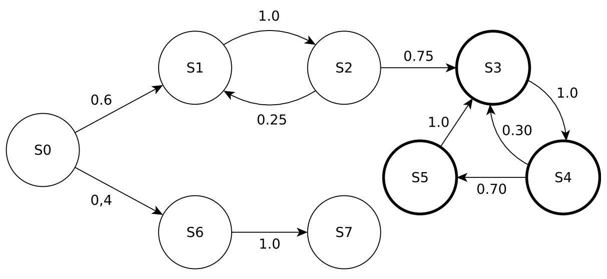

**Figure 1: Markov chain with states in bold (S3, S4, S5) forming a terminal SCC**

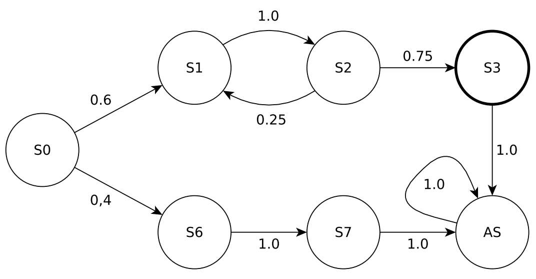

**Figure 2: Absorbing transformation applied to the Markov chain of Figure 1. State S3 has been selected as $s_m$ for the terminal SCC (S3, S4, S5)**

only represent short-term transition probability, hence neglecting important information about the long-term behavior of the agent. Our approach instead, aims at extracting such long-term behavior from every model by using the long-term transition probability. This is performed by giving the process represented by the Markov chain enough time to reach a fixpoint, i.e., when the long-term transition probability would not change anymore from that point onward. The long-term transition probability is important as it represents the probability values of going from every state to any another without considering the states that are crossed in between. This is crucial to discard noise that may be present in the behavior of the agent. For example, in a learning by demonstration setting for a complex task, a teacher agent might make mistakes while trying to follow its default policy, injecting noise in its execution trace. A deliberately harmful scenario instead is when a malware designer intentionally inserts fake API calls to deceive malware detection tools. The approach presented in this work allows also to exploit all the possible paths in the model, considering the presence of cycles efficiently.

We propose to exploit some well known properties to compute the long-term behavior of the agents by using the transient states probability (Theorem 3.8). However, this approach can be used only for absorbing Markov chains, but since our behavioral models come from generic extraction techniques, there are no guarantees that the corresponding Markov chains are absorbing (in our case studies we are never given Markov chains already absorbing). The proposed methodology is independent from how the Markov chain was generated as no assumption is required about the meaning of the states, or the structure of the Markov chain. In fact, we use representations with different meanings for the three experimental settings (see Section 5). To overcome this problem, we define a procedure to transform any Markov chain into an absorbing one. The goal is to design a transformation procedure that given as input any Markov chain provides in output an absorbing Markov chain, allowing to derive the long-term transition probability for the original Markov chain we are interested in.

## 4.1 Absorbing Transformation

Algorithm 1 (AbsEnforcer) details our transformation procedure. Given a Markov chain $M$ of $n$ states, a corresponding absorbing Markov chain $M'$ with $g \leq n$ transient states and an absorbing state $s_a$ is created. Only the block matrix $Q$ of the canonical form (Definition 3.5) is returned since it is the only part used in the subsequent computations. $R$ $(g \times 1)$, $I$ $(1 \times 1)$ and $\emptyset$ $(1 \times g)$ can be easily derived knowing that exactly one absorbing state exists in $M'$ and that every row of $M'$ must sum to 1 (if a state had an outgoing probability value of 0 it would not reach an absorbing state, hence $M'$ would not be absorbing). The first step is to compute the Strongly Connected Components (SCCs) of the Markov chain [22]. A SCC is a set of states and we distinguish between terminal and non-terminal SCC. In Figure 1, states $(S_1, S_2)$ form a non-terminal SCC, whereas states $(S_3, S_4, S_5)$ form a terminal SCC.

*Definition 4.1.* We say a SCC $A$ is terminal if there does not exist a path from a state $s_i \in A$ to a state $s_j \notin A$, otherwise $A$ is defined as non-terminal.

**Algorithm 1** AbsEnforcer

**Require:**
  $M$ -transition matrix of a Markov chain
**Ensure:**
  $Q$ - block matrix of new absorbing Markov chain $M'$

```
1:  $sccs \leftarrow$ TARJAN($M$)                          ▷ Find SCCs
2:  for all $T \in sccs$, with $T$ terminal do
3:      $s_m \leftarrow s \in_R T$               ▷ Randomly select a merge state
4:      for all $s_i \in M$, with $s_i \notin T$ do
5:          $p \leftarrow 0$
6:          for all $s_j \in T$ do  ▷ Remove edges and record weights
7:              $p \leftarrow p + M_{ij}$
8:              $M_{ij} \leftarrow 0$
9:          $M_{im} \leftarrow p$                 ▷ Redirect edges to $s_m \in T$
10:     Merge $T$ into the single state $s_m$
11:     $M_{mm} \leftarrow 0$              ▷ Connect $s_m$ to $s_a$ with $P = 1$
12: $Q \leftarrow M$
13: return $Q$
```

Given a Markov chain $M$, for each terminal SCC $T$, AbsEnforcer merges all the states $s_i \in T$ into a single state $s_m \in T$, connecting it to an absorbing state $s_a$ with a new edge. Since we work with the canonical form (Definition 3.5), and we impose $M_{mm} = 0$ (line 11), the state $s_m$ is consequently connected to $s_a$ with probability value 1, i.e., in the $R$ block matrix not explicitly represented. The update of $M$ (lines 4-9) redirects edges entering any state $s_i \in T$ to the designated merged state $s_m \in T$ before making it the only state of $T$ (line 10). Figure 2 shows an application example of AbsEnforcer to the Markov chain in Figure 1.

In the following we prove that the output of AbsEnforcer is an absorbing Markov chain $M'$ w.r.t. a generic Markov chain $M$. This is fundamental as it allows to apply Theorems 3.7 and 3.8 and then to derive the long-term transition probability for $M$.

Theorem 4.2. *The application of AbsEnforcer to a Markov chain $M$ always results in an absorbing Markov chain $M'$*

Proof. Every Markov chain $M$ contains at least one terminal SCC. Notice that a single state is a SCC since there is always a 0-length path from it to itself. AbsEnforcer creates a new Markov chain $M'$ by merging each terminal SCC $T \in M$ into a chosen state $s_m \in T$, and redirecting all incoming edges of the removed states toward $s_m$. The definition of probability is maintained accumulating the weights of all the edges redirected for each source state and assigning the same sum of weights to the new edge toward $s_m$ (lines 4-9). Additionally, $s_m$ is also connected to the absorbing state $s_a$ with $P = 1$ as a consequence of removing any outgoing edge from it (line 11). Every state $s_i \in M'$ is either contained in a terminal SCC $T$ for $M$, or in a non-terminal SCC $U$ for $M$. In the first case $s_i$ is a merged state $s_m \in M'$ for $T$. Consequently there exists a direct edge in $M'$ such that $s_i = s_m \longrightarrow s_a$. In the second case there exists a path in $M'$ such that $s_i \rightsquigarrow s_m$, where $s_m$ is the result of merging a terminal SCC of $M$. This is true because since the number of states is limited, following an outgoing path from $U$, a terminal SCC will eventually be reached. Then $s_i \rightsquigarrow s_m \longrightarrow s_a$ in $M'$. Therefore, every state of $M'$ is either the absorbing state $s_a$ or a transient state that will eventually reach the absorbing state $s_a$. Hence, from Definition 3.4, $M'$ is an absorbing Markov chain. □

The transformation described above, even though removing states forming terminal SCCs, allows to derive the long-term transition probability for the original Markov chain, including all the removed states. To show this we make use of Lemma 4.3. Also notice that every state of $M$ is a transient state in $M'$ (possibly merged in a $s_m$).

Lemma 4.3. *Within a terminal SCC $T$, the long-term transition probability values of going from any state $s_i \in T$ to any state $s_j \in T$ converge to 1 as the number of steps $n \longrightarrow \infty$.*

Proof. By Definition 3.2, a terminal SCC $T$ is an irreducible Markov chain, meaning that it is possible to go from each state to every other with non-zero probability. Moreover, being terminal, there exists no outgoing path from $T$. Consequently, starting from any state $s_i \in T$, the probability value of reaching any other state $s_j \in T$ increases, approaching 1, as the number of steps $n$ increases. □

As from Theorem 4.2, AbsEnforcer produces an absorbing Markov chain (specifically, its $Q$ block matrix). Therefore, Theorems 3.7 and 3.8 can be applied to its output in order to compute the long-term transition probability as of Definition 4.4.

*Definition 4.4 (Long-term transition probability).* Given a Markov chain $M$, the long-term transition probability value $L_{ij}$ of going from state $s_i \in M$ to state $s_j \in M$ can be computed from the transient states probability $H$ (Theorems 3.7 and 3.8) with $Q = \text{AbsEnforcer}(M)$ as follows

$$L_{ij}(H) = \begin{cases} 1 & \text{if } s_i \text{ and } s_j \text{ are in the same terminal SCC in } M \\ 0 & \text{if } s_i \text{ is in a terminal SCC } T \text{ in } M \text{ and } s_j \notin T \\ H_{im} & \text{if } s_i \text{ is not in a terminal SCC in } M \text{ and } s_j \text{ was} \\ & \text{merged into a state } s_m \text{ in } M' \\ H_{ij} & \text{otherwise} \end{cases}$$

The first case is a direct application of Lemma 4.3, whereas the third one is a consequence: in the long-term, the probability value of reaching a state of a terminal SCC $T$ is the same of reaching any other state of $T$, as in the long-term they can reach each other with probability value 1. The second case is trivial: states within a terminal SCC can only reach other states of the same SCC. The fourth case is where no adjustment has to be made and the standard transient states probability can be used.

## 4.2 Feature Extraction

The feature extraction process is tailored on our supervised learning approach. As we aim at recognizing known behaviors, we require a "blueprint" $D$ as input, along with the actual model $x$ from which to extract the feature vector. The blueprint is used to retrain from $x$, only the long-term transition probability values between the states we are interested in. Hence, the only information that $D$ needs to contain are states and corresponding edges between states. The probability values on the edges (for $D$) are not required since they are not used in the feature extraction process. Essentially, the blueprint $D$ is a "shape" on which to project the long-term transition probability extracted from a model $x$ to analyze.

**Algorithm 2** Extractor

```
Require:
    $D$ - $G(V, E)$ blueprint model with $k = |V|$
    $M$ - Markov chain of model $x$
Ensure:
    $F$ - feature vector

1: $Q \leftarrow$ AbsEnforcer($M$)                     ▷ Algorithm 1
2: $N \leftarrow (I - Q)^{-1}$                         ▷ Theorem 3.7
3: $H \leftarrow (N - I)N_{dg}^{-1}$                   ▷ Theorem 3.8
4: $D' \leftarrow k \times k$ empty matrix
5: for all edges $(s_i, s_j) \in E$ do
6:     if states $s_i, s_j$ exist also in $M$ as $s_v, s_u$ then
7:         $D'_{ij} \leftarrow L_{vu}(H)$              ▷ Definition 4.4
8: return Flatten($D'$)
```

Algorithm 2 (Extractor) details the feature extraction procedure for an unknown model $x$, given a blueprint $D$. The first step is to apply AbsEnforcer to transform $M$ into an absorbing Markov chain $M'$, obtaining its block matrix $Q$ (line 1). Now we can apply Theorems 3.7 and 3.8 to $Q$ as second step, retrieving the transient states probability values $H$ of going from each state to every other for $M'$ (lines 2-3). The last step is to extract the long-term transition probability (Definition 4.4) from $H$, for the states of $x$ that also appear in the blueprint $D$ (lines 5-7). In our experiments we label the states of the models we generate in a consistent manner, therefore, to check if a state of $D$ exists also in $x$ we perform a simple label comparison (line 6).

We can now solve our classification problem by training a classifier using the features extracted by EXTRACTOR. We first create blueprints for every class in $C$. These can be manually crafted by a domain expert or, as we did in the experiments, can simply be created from the known models $K$ by choosing representatives for the classes and retraining their graphs (states and edges, no probability values). It is also possible to select more than one representative per class and to perform a merge to obtain a single blueprint for such class. Then all the blueprints for the classes in $C$ are merged together to obtain a single blueprint $D$. Successively, we train a classifier extracting the training features from each $d \in K$ by calling EXTRACTOR($D$, $M_d$) and using the knowledge of which class $c \in C$, $d$ belongs to. We then classify the unknown behavioral models $x \in B$ by extracting their features, i.e., by calling EXTRACTOR($D$, $M_x$), and then querying the trained classifier. $M_d$ and $M_x$ are the transition matrices of $d$ and $x$ respectively.

## 5 EMPIRICAL EVALUATION

We divide the empirical analysis in two types of experiments: in the first one we focus on agents interacting within classical games, i.e., the iterated Prisoner's Dilemma [17] and a repeated lottery game, while in the second one we analyze real Android malware trying to identify malicious behaviors. The first experimental setting is interesting as it shows that our approach can be used in a generic domain where multiple agents interact and/or observe the effect on the environment of each other actions. The second experimental setting instead is crucial in real world IT defense systems. Figure 3 shows an overview of the empirical evaluation we conducted, where an analyzer agent performs the Monte Carlo Analysis (MCA) described in [19] to generate the behavioral models of different agents, e.g., players of classical games, malicious software agents. As explained in Section 4, the specific technique employed to generate the behavioral models is independent from our methodology used to analyze them. We chose [19] as it is particularly suited to obtain informative behavioral models within interactive settings. From such models we apply our approach to extract the long-term transition probability values as informative features, and compare the learning quality w.r.t. other different features proposed in literature, i.e., 1-step transition probabilities (the classical transition matrix) [19] and n-grams [5, 17].

We design our analyzer agent (a player in the case of the Prisoner's Dilemma) to employ the MCA technique of [19] for the interaction, i.e., to select the actions to perform in order to gain as much information as possible on the adversary, trying to minimize the entropy of the behavioral model being generated. Following [19], a behavioral model is represented with a set of Markov chains extracted from the observation of an agent's actions (see Section 3.2). For all the experiments we trained a Linear SVM performing a $k$-fold cross validation with $k = 5$. Classification quality is evaluated using precision, recall, and $F_1$-score. Since each behavioral model encodes multiple Markov chains, EXTRACTOR is applied to each of them individually in order to extract a feature vector for each Markov chain, and then performing a concatenation. Figure 4 for example, encodes two different Markov chains, i.e., one for action $C$ and one for action $D$.

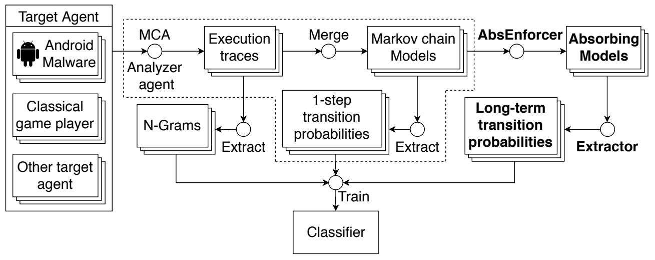

**Figure 3: Overview of our methodology for behavioral analysis. The dashed area contains the method of [19], whereas bold text represent our contribution**

### 5.1 Classical Games

In the iterated Prisoner's Dilemma, two players are given the choice to cooperate ($C$) or to defect ($D$) in a repeated interaction of the same stage game. While there are various approaches to find strategies that optimize agents' payoffs studying equilibria [16], here we focus on the identification of known strategies only by observing the interaction between players. Hence, we design 6 strategies previously used in literature [5, 17] for player $B$ (target agent), while player $A$ (analyzer agent) chooses its actions following the MCA technique of [19]. The six strategies are: i) tit-for-tat, ii) retaliation, iii) random, iv) always cooperate, v) always defect, vi) mixed. Strategy i) always plays the action played by the adversary in the last game, whereas strategy ii) cooperates until the adversary defects for the first time and then defects forever. For strategy vi), cooperate and defect are chosen with 4/5 and 1/5 probability values respectively. States are labeled with the joint actions that made the game reach such state, whereas edges are labeled with the action of player $A$ that triggered such transition. Figure 4 shows an example of an observed random behavioral model for player $B$. States are represented with the joint actions of the players: $CD$ for example is the result of actions *(cooperate, defect)* by player $A$ and $B$ respectively. Every strategy has been played 20 times in an iterated Prisoner's Dilemma of length 100, obtaining 120 behavioral models for player $B$. The aim then is, given such models, to classify them over the 6 known strategies (classes). The blueprint $D$ has been created selecting random representatives from each strategy and merging all their graphs together. Table 1 reports the evaluation of the process. Results show that strategies i), ii), iv) and v) are perfectly identified by our method. However, since strategies iii) and vi) differ only in the probability values they assign to actions cooperate and defect, if the Markov chain states are highly connected, e.g., if they form a single SCC, the long-term behavior tends to flatten the differences between transition probability values, making harder to distinguish behaviors in the long-term compared to the short. This is a limitation of our approach that arises in the pathological case of models composed only by few terminal SCCs. However, this is unlikely to happen with more complex models and real world scenarios such as in our next experiments.

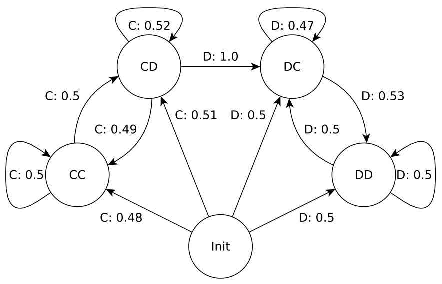

**Figure 4: Example of an observed random behavioral model for player $B$ in the iterated Prisoner's Dilemma**

**Table 1: Player's strategy identification for the iterated Prisoner's Dilemma**

| | Strategy | Precision | Recall | $F_1$-score |
|---|---|---|---|---|
| i) | Tit-for-tat | 1.00 | 1.00 | 1.00 |
| ii) | Retaliation | 1.00 | 1.00 | 1.00 |
| iii) | Random | 1.00 | 0.65 | 0.79 |
| iv) v) | Always C/D | 1.00 | 1.00 | 1.00 |
| vi) | Mixed | 0.74 | 1.00 | 0.85 |
| | Total | 0.96 | 0.94 | 0.94 |

As second evaluation setting, we define a custom non-adversarial repeated lottery game [12] as follows: at every iteration, a player chooses between a safe ($S$) and a risky ($R$) lottery, accumulating the reward at each stage. $S$ gives reward 4 with $P = 0.9$ and halves the current accumulated reward with $P = 0.1$. $R$ gives a reward of 8 with $P = 0.5$, halves the current accumulated reward with $P = 0.4$, and sets it to 0 with $P = 0.1$. Figure 5 shows an example of an observed behavioral model for the repeated lottery game. The internal state of the player is assumed to be observable and containing the value of the current accumulated reward. Such value is used to label the states of the model, whether edges instead are labeled with the lottery chosen by the player. In this case, behavioral models are generated by the analyzer only by observing the player agent, i.e., no interaction is involved between the two. We design some similar strategies w.r.t. the iterated Prisoner's Dilemma, i.e., i) always $S$, ii) always $R$, iii) $R$ until loss (always $R$ until the first loss happens, $S$ from that stage onward), iv) $S$ until loss (always $S$ until the first loss happens, $R$ from that stage onward), v) random, vi) mixed ($S$ with $P = 4/5$, $R$ with $P = 1/5$). Additionally, we also design more complex behaviors where $R$ is played when the current accumulated reward is within a specific range $[a, b]$, $S$ otherwise: vii) $R$ between $[10, 20]$, viii) $R$ between $[10, 50]$, ix) $R$ between $[20, 40]$. Every strategy has been played 20 times in a repeated lottery game of length 500, obtaining 180 behavioral models for the player agent. The aim, again, is to classify such models over the 9 known strategies (classes). This experiment is different from the previous one as the generated behavioral models are much bigger (up to 1913 states) and contain a various number of terminal and non-terminal SCCs, resulting in a comprehensive evaluation setting for our approach. Also in this case, the blueprint $D$ has been created selecting random representatives from each strategy and merging all their

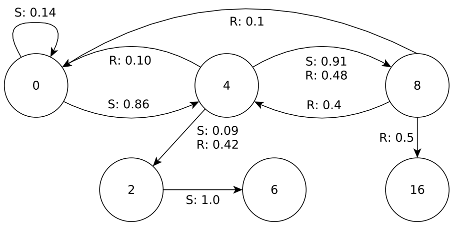

**Figure 5: Example of an observed behavioral model in the repeated lottery**

**Table 2: Player's strategy identification for the repeated lottery game**

| | Strategy | Precision | Recall | $F_1$-score |
|---|---|---|---|---|
| i) | Always S | 0.86 | 0.90 | 0.88 |
| ii) | Always R | 0.95 | 1.00 | 0.98 |
| iii) | R until Loss | 0.89 | 0.85 | 0.87 |
| iv) | S until Loss | 1.00 | 0.95 | 0.97 |
| v) | Random | 1.00 | 0.95 | 0.97 |
| vi) | Mixed | 0.95 | 1.00 | 0.97 |
| vii) viii) ix) | R between [a, b] | 1.00 | 1.00 | 1.00 |
| | Total | 0.96 | 0.96 | 0.96 |

graphs together. Table 2 reports the results of the process, where it is visible that the strategies are overall well classified. We notice that strategies v) and vi) are identified more clearly w.r.t. to Table 1. As we mentioned before, in bigger and realistic models, the flattening problem of the long-term probabilities is much less prominent. Strategies i) and iii), and strategies ii) and iv) instead, can be confused with each other depending on when the first loss happens during the game. Strategy iv) for example becomes exactly strategy ii) after the first loss. If this change happens at the very beginning of a game, the two strategies become indistinguishable.

## 5.2 Malware Analysis

In the malware analysis application scenario, the first step to analyze an unknown software is to decide whether it could be malicious [1, 10]. In this case, the type of countermeasures that an analyzer should take depends on the type of malicious applications. Detecting whether a malicious behavior shares common characteristics with known malware families is extremely important to take effective countermeasures. Notice that in a malware behavioral model, a state is not malicious by itself. It is the composition of multiple states (how they are connected together) that can make a behavior malicious overall. A key interesting aspect is that advanced malware inject noise, e.g., sequences of random or non-dangerous actions, in their behavior as an anti-detection mechanism, hence creating an extremely challenging scenario for the analyzers. Another complication for the analyzer comes from small malware injected into bigger, benign applications, e.g., a password stealer inserted into the code of a game. In this case, the major portion of the behavioral model corresponds to actions belonging to the benign gaming application, whereas the few related to the password stealing process appear within them.

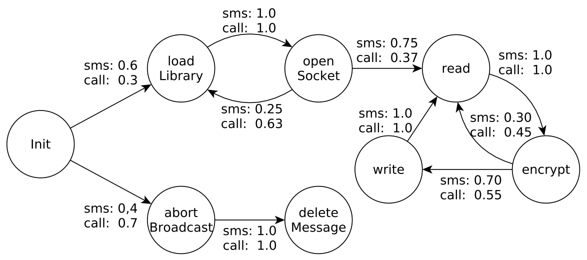

**Figure 6: Malware model example**

From the dataset collected in [23], we selected approximately 1200 malware samples, consisting of 23 families, that better suit an interactive analysis such as AMA, i.e., reacting to user's actions. AndroRAT and GoldDream families are an example of small malware injected into bigger applications (games and others). They steal personal information such as contact numbers, sms and call contents. Gorpo and Kemoge families instead employ anti-detection techniques such as dynamically loading the malicious code at runtime and performing unrelated actions to intentionally inject noise. The other families reported are of classical malware, i.e., not injected and without advanced anti-detection techniques. Models obtained by analyzing this dataset represent a concrete application context on real data that is crucial for cyber-security. We compare our approach with the MCA presented in [19] where the same process is used to generate the malware models (the difference lies in the feature extraction as summarized in Figure 3). In this experiment the internal state of the malware agent is not observable, consequently states are represented with the effect of the agent's actions on the environment (API calls). Figure 6 shows an example of malware model generated by MCA, where states are labeled with API calls and edges connect two consecutive API calls observed in a malware execution trace. As for the previous representations, edges are labeled with transition probabilities conditioned by analyzer actions. We also compare with the n-gram features extensively used in literature [18, 26]. As suggested in such works we experimented with SVM and K-Nearest Neighbors (KNN) classifiers using gram lengths in the range [1, 4]. To perform a fair comparison, we extract the n-grams directly from the same execution traces used by the other two methods, i.e., MCA and ours. Regarding our approach, the blueprint $D$ has been generated merging the graphs of the representatives for each family. Specifically, if the standalone (not injected) or clean (without anti-detection mechanisms) version of a malware for a family is known, its behavioral model is used as representative of such family, otherwise random behavioral models are chosen from the same family.

Table 3 reports our empirical best results obtained using SVMs and gram lengths of 1 and 4. In the interest of space we report only a subset of families, but the considerations made hold for the entire dataset. The two methods using features extracted from the Markov chains, i.e., MCA and the one proposed in this work, have overall better results when compared to n-grams. Using Markov chain transition probability values as features allows to better capture distinctive characteristics of the malware dynamics, improving the classifier performance. Moreover, when comparing MCA to our approach we can notice that the performance of the classifier are always comparable and most of the time are significantly better in favor of ours. In more detail, our technique performs significantly better for malware that are injected into benign applications (AndroRAT and GoldDream) or for malware employing anti-detection techniques preforming a lot of noisy actions (Gorpo and Kemoge). This confirms that using the long-term transition probability values as features allows to effectively removes noise, significantly improving the classification performances for such families. With classical malware (lower half of Table 3) instead, our approach does not consistently provide significant gains and is comparable to the others. This suggests that our proposed methodology should complement existing techniques to provide benefits in specific and important situations, e.g., malware injection and countering anti-detection mechanisms.

**Table 3: Malware classification comparison $F_1$-score**

| Family | 3-grams | 4-grams | MCA | Ours |
|---|---|---|---|---|
| AndroRAT | 0.83 | 0.86 | 0.84 | **0.94** |
| GoldDream | 0.88 | 0.92 | 0.92 | **0.95** |
| Gorpo | 0.77 | 0.66 | 0.81 | **0.93** |
| Kemoge | 0.58 | 0.57 | 0.44 | **0.92** |
| Cova | 0.91 | 0.92 | 0.94 | **0.97** |
| FakeAV | 0.89 | 0.89 | 0.89 | 0.89 |
| Kuguo | 0.87 | 0.85 | **0.93** | 0.90 |
| SpyBubble | 0.94 | 0.94 | 0.63 | **0.95** |
| Winge | 0.78 | 0.72 | **0.88** | 0.86 |

## 6 CONCLUSIONS

We propose the use of Markov chains to identify known behaviors of intelligent agents acting within uncertain environments. More in detail, we employ classification to solve the problem and we use the long-term transition probability values as features. We design a transformation to enforce the absorbency property for Markov chains, enabling the computation of such features for generic Markov chains. We evaluate our methodology in three domains: two player games, a single player repeated lottery game, and malware analysis. The empirical evaluation shows that our approach provides informative features to successfully identify known behaviors. In particular, for the malware analysis scenario this method allows to significantly outperform state-of-the-art techniques when considering real-world injected malware samples and advanced anti-detection mechanisms. This work opens several future directions, including the application to different domains of cyber-security, e.g., web-based attacks, and beyond, e.g., malicious or anomalous behaviors of cyber-physical systems and drones.

## 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, Universities and Research (MIUR), and by University of Verona and Cythereal Inc. under Joint Projects 2017 Initiative (JPVR17ZMAL).

## 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] Damla Arifoglu and Abdelhamid Bouchachia. 2017. Activity Recognition and Abnormal Behaviour Detection with Recurrent Neural Networks. *Procedia Computer Science* 110 (2017), 86 – 93. https://doi.org/10.1016/j.procs.2017.06.121 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops.

[3] A. Busic, I.M.H. Vliegen, and A. Scheller-Wolf. 2009. *Comparing Markov chains : combining aggregation and precedence relations applied to sets of states*. Eurandom.

[4] Martin Dyer, Leslie Ann Goldberg, Mark Jerrum, and Russell Martin. 2006. Markov chain comparison. *Probab. Surveys* 3 (2006), 89–111. https://doi.org/10.1214/154957806000000041

[5] James W. Friedman. 1971. A Non-cooperative Equilibrium for Supergames. *Review of Economic Studies* 38, 1 (1971), 1–12. https://EconPapers.repec.org/RePEc:oup:restud:v:38:y:1971:i:1:p:1-12.

[6] 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

[7] Pablo Hernandez-Leal and Michael Kaisers. 2017. Towards a Fast Detection of Opponents in Repeated Stochastic Games. In *Autonomous Agents and Multiagent Systems*, Gita Sukthankar and Juan A. Rodriguez-Aguilar (Eds.). Springer International Publishing, Cham, 239–257.

[8] Kunihiko Hiraishi and Koichi Kobayashi. 2014. Detection of Unusual Human Activities Based on Behavior Modeling. *IFAC Proceedings Volumes* 47, 2 (2014), 182 – 187. https://doi.org/10.3182/20140514-3-FR-4046.00029 12th IFAC International Workshop on Discrete Event Systems (2014).

[9] J.G. Kemeny and J.L. Snell. 1983. *Finite Markov Chains: With a New Appendix "Generalization of a Fundamental Matrix"*. Springer New York. https://books.google.com.sg/books?id=0bTK5uWzbYwC

[10] 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.

[11] 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.

[12] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. 1995. *Microeconomic Theory*. Oxford University Press, New York.

[13] V.valli Mayil. 2012. Web Navigation Path Pattern Prediction using First Order Markov Model and Depth first Evaluation. *International Journal of Computer Applications* 45, 16 (May 2012), 26–31.

[14] 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

[15] 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. https://doi.org/10.1109/SP.2007.17

[16] J. F. Nash. 1950. Equilibrium Points in N-Person Games. *Proceedings of the National Academy of Sciences of the United States of America* 36, 48-49 (1950).

[17] Martin Nowak and Karl Sigmund. 1993. A Strategy of Win-Stay, Lose-Shift That Outperforms Tit-for-Tat in the Prisoner's Dilemma Game. *Nature* 364 (08 1993), 56–8.

[18] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19, 4 (2011), 639–668. https://doi.org/10.3233/JCS-2010-0410

[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, 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

[21] Ramesh R. Sarukkai. 2000. Link Prediction and Path Analysis Using Markov Chains. In *Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Netowrking*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 377–386. http://dl.acm.org/citation.cfm?id=347319.346322

[22] R. Tarjan. 1971. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*. 114–121. https://doi.org/10.1109/SWAT.1971.10

[23] 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.

[24] James A. Whittaker and Michael G. Thomason. 1994. A Markov Chain Model for Statistical Software Testing. *IEEE Trans. Softw. Eng.* 20, 10 (Oct. 1994), 812–824. https://doi.org/10.1109/32.328991

[25] 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

[26] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. 2013. A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AISec '13)*. ACM, New York, NY, USA, 67–76. https://doi.org/10.1145/2517312.2517316

[27] 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

[28] Jianhan Zhu, Jun Hong, and John G. Hughes. 2002. Using Markov Chains for Link Prediction in Adaptive Web Sites. In *Proceedings of the First International Conference on Computing in an Imperfect World (Soft-Ware 2002)*. Springer-Verlag, Berlin, Heidelberg, 60–73. http://dl.acm.org/citation.cfm?id=645974.758446