Cognitively-Inspired Inference for Malware Task Identification

Eric Nunes, Casey Buto, Paulo Shakarian, Christian Lebiere, Stefano Bennati, and Robert Thomson

Abstract Malware reverse-engineering, specifically, identifying the tasks a given piece of malware was designed to perform (e.g., logging keystrokes, recording video, establishing remote access) is a largely human-driven process that is a difficult and time-consuming operation. In this chapter, we present an automated method to identify malware tasks using two different approaches based on the ACT-R cognitive architecture, a popular implementation of a unified theory of cognition. Using three different malware collections, we explore various evaluations for each of an instance-based and rule-based model—including cases where the training data differs significantly from test; where the malware being evaluated employs packing to thwart analytical techniques; and conditions with sparse training data. We find that our approach based on cognitive inference consistently out-performs the current state-of-the art software for malware task identification as well as standard machine learning approaches—often achieving an unbiased F1 score of over 0.9.

1 Introduction

Identifying the tasks a given piece of malware was designed to perform (e.g. logging keystrokes, recording video, establishing remote access, etc.) is a difficult and time consuming task that is largely human-driven in practice [1]. The complexity of this task increases substantially when you consider that malware is constantly evolving, and that how each malware instance is classified may be different based on each

E. Nunes (✉) · C. Buto · P. Shakarian
Arizona State University, Tempe, AZ, USA
e-mail: enunes1@asu.edu; cbuto@asu.edu; shak@asu.edu

C. Lebiere · S. Bennati
Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: cl@cmu.edu

R. Thomson
United States Military Academy, West Point, NY, USA
e-mail: robert.thomson@westpoint.edu

M. A. Tayebi et al. (eds.), *Open Source Intelligence and Cyber Crime*, Lecture Notes in Social Networks, https://doi.org/10.1007/978-3-030-41251-7_7

cyber-security expert's own particular background. Automated solutions for this problem are highly attractive as they can significantly reduce the time it takes to conduct remediation in the aftermath of a cyber-attack.

Earlier work has sought to classify malware by similar "families", something which has been explored as a supervised classification problem [2–4]. However, differences over determining "ground truth" for malware families (i.e. Symantec and McAfee cluster malware into families differently) and the tendency for automated approaches to only succeed at "easy to classify" samples [5, 6] are two primary drawbacks of malware family classification. More recently, there has been work on directly inferring the tasks a malware was designed to perform [7]. This approach leverages static malware analysis (i.e. analysis of the malware sample conducted without execution, such as decompilation) and a comparison with a crowd-source database of code snippets using a proprietary machine learning approach. However, a key shortcoming of the static method is that it is of limited value when the malware authors encrypt part of their code—as we saw with the infamous Gauss malware [8]. This work builds upon recent developments in the application of cognitive models to intelligence analysis tasks [9] and our own preliminary studies on applying cognitive models to identify the tasks a piece of malware was designed to perform [10, 11]. Specifically, in this chapter, we report

– Experimental results illustrating consistent and significant performance improvements (in terms of precision, recall, and F1) of the instance-based cognitive model approach when compared with various standard machine learning approaches (including SVM, logistic regression and random forests) for two different sandboxes and for three different datasets.
– Experimental results showing a consistent and significant performance improvement of the instance-based cognitive model and several other machine learning approaches when compared to the current state-of-the-art commercial technology (based on static analysis).
– Experiments where we study cases where the malware samples are mutated, encrypted, and use different carriers—providing key insights into how our approach will cope with operational difficulties.
– Experimental results illustrating that a cognitively-inspired intermediate step of inferring probability distribution over malware families provides improved performance over the machine learning and rule-based cognitive model (though no significant change to the instance-based cognitive model).

Cognitive models have proved to significantly outperform classical machine learning approaches and state of the art products available in the market for malware task prediction [10–12]. This chapter consolidates the results presented in previous research by including additional results for the GVDG dataset, run-time comparisons of the experiments and discussing the cognitive models in terms of parameter selection and time complexity analyses. We also provide new experimental results utilizing a dataset based on the MetaSploit framework [13]—demonstrating how our framework adapts to features based on malware that utilizes the network protocol stack. We also explore the concept of predicting hacker intentions on a host machine.

This chapter is organized as follows. In Sect. 2 we state the technical preliminaries used in the chapter. In Sect. 3.1 we introduce our cognitive-based approaches, describing the algorithms and explaining our selection of parameter settings. This is followed by a description of the baseline approaches that we studied in our evaluation in Sect. 4.1 and a description of the two different dynamic malware sandbox environments we used in Sect. 4.2. In Sect. 5 we present our suite of experimental results which include experiments involving samples discovered by Mandiant, Inc. in their APT1 report [14], samples created using the GVDG [15] tool, and samples created using the MetaSploit framework. Finally, related work is discussed in Sect. 6.

2 Technical Preliminaries

Throughout this chapter, we shall assume that we have a set of malware samples that comprise a historical corpus (which we shall denote $\mathcal{M}$) and each sample $i \in \mathcal{M}$ is associated with a set of tasks (denoted $tasks(i)$) and a set of attributes (denoted $attribs(i)$). Attributes are essentially binary features associated with a piece of malware that we can observe using dynamic and/or static analysis while the tasks—which tell us the higher-level purpose of the malware—must be determined by a human reviewing the results of such analysis. As $\mathcal{M}$ comprises our historical knowledge, we assume that for each $i \in \mathcal{M}$ both $tasks(i)$ and $attribs(i)$ are known. For a new piece of malware, we assume that we only know the attributes. We also note that throughout the chapter, we will use the notation $|\cdot|$ to denote the size of a given set. Tables 1 and 2 provide examples of the attributes and tasks based on the malware samples from the Mandiant APT1 dataset (created from samples available at [16], see also [14]). For instance, *hasDynAttrib* looks at the behavior section of the analysis report and extracts all the activity of the malware on the host machine. The attribute *usesDll* enumerates all the libraries that were used by the malware on the host machine. The file activity and the registry activity is captured by *fileAct* and *regAct*. Finally all the processes initiated and terminated by the malware are captured by *proAct*. There is not a fixed number of any of these attributes for a

Table 1 Attributes extracted through automated malware analysis

Attribute	Intuition
usesDll(X)	Malware uses a library X
regAct(K)	Malware conducts an activity in the registry, modifying key K.
fileAct(X)	Malware conducts an activity on certain file X
proAct	Malware initiates or terminates a process

Table 2 Sample of malware tasks

Task	Intuition
beacon	Beacons back to the adversary's system
enumFiles	Designed to enumerate files on the target
serviceManip	Manipulates services running on the target
takeScreenShots	Takes screen shots
upload	Designed to upload files from the target

given malware. The number of attributes depends on the analysis report generated from the sandbox. A full description of this dataset is presented in Sect. 5.

Throughout the chapter, we will also often consider malware families, using the symbol $\mathcal{F}$ to denote the set of all families. Each malware sample will belong to exactly one malware family, and all malware samples belonging to a given family will have the same set of tasks. Hence, we shall also treat each element of $\mathcal{F}$ as a subset of $\mathcal{M}$.

3 Cognitively-Inspired Inference

While human inference has memory and attentional limitations, their cognitive processes are powerful, where adaptive heuristic strategies are adopted to accomplish the tasks under strong time constraints using limited means. An advantage of using a cognitive model to describe inferential processes is that the underling architecture provides the benefits of human-inspired inference while allowing for more flexibility over constraints such as human working memory. We believe that there is a valid use of cognitive architectures for artificial intelligence that makes use of basic cognitive mechanisms while not necessarily making use of all constraints of the architecture. In that case, it is arguably better to specifically state which aspects of the model are not constrained by data, and rather than mock up those aspects in plausible but impossible to validate manner, simply treat them as unmodeled processes. This approach results in simpler models with a clear link between mechanisms used and results accounted for, rather than being obscured by complex but irrelevant machinery. For instance, while the models described in this chapter use activation dynamics well-justified against human behavioral and neural data to account for features such as temporal discounting, we do not directly model working memory constraints to allow for more features of malware and more instances to be present in memory.

3.1 ACT-R Based Approaches

We propose two models built using the mechanisms of the ACT-R (Adaptive Control of Thought-Rational) cognitive architecture [17]. These models leverage the work on applying this architecture to intelligence analysis problems [9]. In particular, we look to leverage our recently-introduced instance-based (ACTR-IB) and rule-based (ACTR-R) models [10, 11]. Previous research has argued the ability of instance-based learning in complex dynamic situations making it appropriate for sensemaking [18]. On the other hand, the rule-based learning is a more compact representation of associating samples in memory with their respective families. In this section, we review some of the major concepts of the ACT-R framework that are relevant to these models and provide a description of both approaches.

We leveraged features of the declarative memory and production system of the ACT-R architecture to complete malware task identification. In ACT-R, recall from declarative memory (c.f., identification, for our purposes) depends on three main components: activation strengthening (i.e., the base-level activation of an element), associative (i.e., spreading) activation, and inter-element similarity (i.e., partial matching). These three values are summed together to represent an item's total activation. When a recall is requested from memory, the item with the highest total activation is retrieved.

Declarative Knowledge Declarative knowledge is represented formally in terms of *chunks*. Chunks have an explicit type, and consist of an ordered list of slot-value pairs of information. Chunks are retrieved from declarative memory by an activation process, and chunks are each associated with an *activation strength* which in turn is used to compute a *retrieval probability*. In this chapter, chunks will typically correspond to a malware family. In the version of ACTR-IB where we do not represent families explicitly, the chunks correspond with samples in the training data.

For a given chunk i, the activation strength A_i is computed as,

$$A_i = B_i + S_i + P_i \tag{1}$$

where, B_i is the base-level activation, S_i is the spreading activation, and P_i is the partial matching score. We describe each of these in more detail as follows.

Base-Level Activation (B_i) Technically, base-level for chunk i reflects both the frequency and recency of samples in memory, even though we are not using recency here but it could easily be applicable to weigh samples toward the more recent ones. More important, base-level is set to the log of the prior probability (i.e., the fraction of samples associated with the chunk) in ACTR-R; for instance-based (ACTR-IB), we set it to a base level constant β_i.

Spreading Activation (S_i) Spreading activation is a measure of the uniqueness of the attributes between a test sample i and a sample j in memory. The spread of activation to sample i is computed by the summing the strengths of association between sample j and the attributes of the current sample i being considered. To compute the spreading activation we compute the *fan* of attribute a (i.e., the number of samples in memory with attribute a) for each attribute. The strength of association is computed differently in both approaches and, in some cognitive model implementations, is weighted (as is done in ACTR-R of this chapter).

Partial Matching (P_i) A partial matching mechanism computes the similarity between two samples. In this work, it is only relevant to the instance-based approach. Given a test sample j, its similarity with a sample i in memory is computed as a product of the mismatch penalty (mp, a parameter of the system) and the degree of mismatch M_{ji}. We define the value of M_{ji} to be between 0 and -1; 0 indicates complete match while -1 complete mismatch.

As common with models based on the ACT-R framework, we shall discard chunks whose activation strength is below a certain threshold (denoted τ). Once the activation strength, A_i, is computed for a given chunk, we can then calculate the activation probability, p_i. This is the probability that the cognitive model will recall that chunk and is computed using the Boltzmann (softmax) equation [19], which we provide below.

$$Pr_i = \frac{(e^{\frac{A_i}{s}})}{\sum_j (e^{\frac{A_j}{s}})} \tag{2}$$

Here, e is the base of the natural logarithm and s is momentary noise inducing stochasticity by simulating background neural activation (this is also a parameter of the system).

3.2 ACT-R Instance-Based Model

The instance based model is an iterative learning method that reflects the cognitive process of accumulating experiences (in this case the knowledge base of training samples) and using them to predict the tasks for unseen test samples. Each malware instance associates a set of attributes of that malware with its family. When a new malware sample is encountered, the activation strength of that sample with each sample in memory is computed using Eq. 1. The spreading activation is a measure of the uniqueness of the attributes between a test sample i and a sample j in memory. To compute the spreading activation we compute the fan for each attribute a ($fan(a)$ finds all instances in memory with the attribute a) of the test sample i. The Partial matching is computed as explained above. The degree of mismatch is computed as the intersection between the attribute vector of the given malware and each sample in memory normalized using the Euclidean distance between the two vectors. The retrieval probability of each sample j in memory with respect to the test sample i is then computed using Eq. 2. This generates a probability distribution over families. The tasks are then determined by summing up the probability of the families associated with that task with an appropriately set threshold (we set that threshold at 0.5 (indicates that the model should be more than 50% confident before a task is predicted for a test malware sample)). Algorithm 1 shows the pseudo code for the instance-based model.

Algorithm 1: ACT-R instance-based learning

```
**INPUT:** New malware sample $i$, historical malware corpus $\mathcal{M}$.
**OUTPUT:** Set of tasks associated with sample $i$.
**for** query malware sample $i$ **do**
    **for all** $j$ in $\mathcal{M}$ **do**
        $B_j = \beta_j$
        $P_j = mp \times \frac{|attribs(i) \cap attribs(j)|}{\sqrt{|attribs(i)| \times |attribs(j)|}}$
        **for** $a \in attribs(i)$ **do**
            **if** $a \in attribs(j)$ **then**
                $s_{ij} \mathrel{+}= log(\frac{|\mathcal{M}|}{|fan(a)}|)$
            **else**
                $s_{ij} \mathrel{+}= log(\frac{1}{|\mathcal{M}|})$
            **end if**
        **end for**
        $S_j = \sum_j \frac{s_{ij}}{|attribs(i)|}$
        Calculate $A_j$ as per Equation 1
    **end for**
    Calculate $p_j$ as per Equation 2
    $p_f = \sum_{j \in f s.t. A_j \geq \tau} p_j$
    $t_p = \{t \in T | p_f \geq 0.5\}$
**end for**
```

Time Complexity of Instance-Based Model The Instance based model has no explicit training phase, so there are no training costs associated with it. For a given test sample the model computes the activation function for each sample in the knowledge base. Hence the time complexity increases linearly with the knowledge base. Let n be the number of the samples in the knowledge base and m is the number of attributes associated with the test sample, then the time complexity can be given as $O(nm)$ for each test sample, as we expect m to be relative small ($n >> m$), the relationship is linear in n.

3.3 ACT-R Rule-Based Model

In this version of ACT-R model we classify the samples based on simple rules computed during the training phase. Given a malware training sample with its set of attributes a, along with the ground truth family value, we compute a pair of conditional probabilities $p(a|f)$ and $p(a|\neg f)$ for an attribute in a piece of malware belonging (or not belonging) to family f. These probabilistic rules (conditional probabilities) are used to set the strength of association of the attribute with a family ($s_{a,f}$). The strength of association is weighted by the source activation w to avoid retrieval failures for rule-based models. We use empirically determined Bayesian priors $p(f)$ to set the base-level of each family as opposed to using a constant base-level for instance based. Only two components of the activation Equation 1 are used, namely the base-level and the spreading activation. Given the attributes for current malware, we calculate the probability of the sample belonging to each family according to Eq. 2, generating a probability distribution over families. The task are then determined in a similar way to that of instance-based model. Algorithm 2 shows the pseudo code for the rule-based model.

Algorithm 2: ACT-R rule-based learning

```
INPUT: New malware sample $i$, historical malware corpus $\mathcal{M}$.
OUTPUT: Set of tasks associated with new sample $i$.
TRAINING:
Let $X = \bigcup_{j \in \mathcal{M}} attrib(j)$
for all $a$ in $X$ do
    Compute the set of rules $p(a|f)$ and $p(a|\neg f)$
    (where $p(a|f) = \frac{|\{i \in \mathcal{M} \cap f | a \in attrib(i)\}|}{|f|}$
    and $p(a|\neg f) = \frac{|\{i \in \mathcal{M} - f | a \in attrib(i)\}|}{|\mathcal{M}| - |f|}$)
end for
TESTING:
for all $f \in \mathcal{F}$ do
    $B_f = log(p(f))$ (where $p(f) = \frac{|f|}{|\mathcal{M}|}$)
    for all $a \in attrib(i)$ do
        $s_{a,f} = log(\frac{p(a|f)}{p(a|\neg f)})$; $S_f \mathrel{=}+ \frac{w \times s_{a,f}}{|attribs(i)|}$
    end for
    $A_f = B_f + S_f$
end for
Calculate $p_f$ as per Equation 2
$t_p = \{t \in T | p_f \geq 0.5\}$
```

Time Complexity of Rule-Based Model For Rule-based model computing the rules for each attribute in the knowledge base significantly add to the computation time. Let n be the number of samples in the training set, m be the number of attributes in the new piece of malware, and m^* be the cardinality of $\bigcup_{j \in \mathcal{M}} attrib(j)$. The resulting time complexity for training is then $O(m^* n)$ for training, which is significant as we observed $m^* >> m$ in our study. While this is expensive, we note that for testing an individual malware sample, the time complexity is less than the testing phase for the instance based $O(|\mathcal{F}|m)$—though the instance based model requires no explicit training phase (which dominates the time complexity of the training phase for the rule-based approach).

Table 3 Parameters for the cognitive models

Model	Parameters
Instance based learning	$\beta = 20$ (base-level constant)
	$s = 0.1$ (stochastic noise parameter)
	$\tau = -10$ (activation threshold)
	$mp = 20$ (mismatch penalty)
Rule based learning	$s = 0.1$ (stochastic noise parameter)
	$w = 16$ (source activation)

3.4 Model Parameter Settings

The two proposed models leverage separate components of the activation function. Table 3 provides a list of parameters used for both the ACT-R models—we use standard ACT-R parameters that have been estimated from a wide range of previous ACT-R modeling studies from other domains [20] and which are also suggested in the ACT-R reference manual [21].

The intuition behind these parameters is as follows. The parameter s injects stochastic noise in the model. It is used to compute the variance of the noise distribution and to compute the retrieval probability of each sample in memory. The mismatch penalty parameter mp is an architectural parameter that is constant across samples, but it multiplies the similarity between the test sample and the samples in knowledge base. Thus, with a large value it penalizes the mismatch samples more. It typically trades off against the value of the noise s in a signal-to-noise ratio manner: larger values of mp lead to more consistent retrieval of the closest matching sample whereas larger values of s leads to more common retrieval of poorer matching samples.The activation threshold τ determines which samples will be retrieved from memory to make task prediction decisions. The base level constant β is used to avoid retrieval failures which might be caused due to high activation threshold. The source activation w is assigned to each retrieval to avoid retrieval failures for rule-based models.

4 Experimental Setup

4.1 Baseline Approaches

We compare the proposed cognitive models against a variety of baseline approaches—one commercial package and five standard machine learning techniques. For the machine learning techniques, we generate a probability distribution over families and return the set of tasks associated with a probability of 0.5 or greater while the commercial software was used as intended by the manufacturer. Parameters for all baseline approaches were set in a manner to provide the best performance.

Commercial Offering: Invencia Cynomix Cynomix is a malware analysis tool made available to researchers by Invencia industries [7] originally developed under DARPA's Cyber Genome project. It represents the current state-of-the-art in the field of malware capability detection. Cynomix conducts static analysis of the malware sample and uses a proprietary algorithm to compare it to crowd-sourced identified malware components where the functionality is known.

Decision Tree (DT) Decision tree is a hierarchical recursive partitioning algorithm. We build the decision tree by finding the best split attribute i.e. the attribute that maximizes the information gain at each split of a node. In order to avoid over-fitting, the terminating criteria was set to less than 5% of total samples. Malware samples are tested by the presence and absence of the best split attribute at each level in the tree till it reaches the leaf node. When it reaches the leaf node the probability distribution at the leaf node is assigned to the malware sample.

Naive Bayes Classifier (NB) Naive Bayes is a probabilistic classifier which uses Bayes theorem with independent attribute assumption. During training we compute the conditional probabilities of a given attribute belonging to a particular family. We also compute the prior probabilities for each family; i.e., fraction of the training data belonging to each family. Naive Bayes assumes that the attributes are statistically independent hence the likelihood for a sample S represented with a set of attributes a associated with a family f is given as, $p(f|S) = P(f) \times \prod_{i=1}^{d} p(a_i|f)$.

Random Forest (RF) Ensemble methods are popular classification tools. They are based on the idea of generating multiple predictors used in combination to classify new unseen samples. We use a random forest which combines bagging for each tree with random feature selection at each node to split the data, thus generating multiple decision tree classifiers [22]. Each decision tree gives its own opinion on test sample classification which is then merged to generate a probability distribution over families. For all the experiments we set the number of trees to be 100, which gives us the best performance.

Support Vector Machine (SVM) Support vector machines (SVM) are proposed by Vapnik [23]. SVMs work by finding a separating margin that maximizes the geometric distance between classes. The separating margin is termed as hyperplane. We use the popular LibSVM implementation [24] which is publicly available. The implementation has the option of returning the probability distribution as opposed to the maximum probability prediction.

Logistic Regression (LOG-REG) Logistic regression classifies samples by computing the odds ratio. The odds ratio gives the strength of association between the attributes and the family like simple rules used in the ACT-R rule based learning. We implement the multinomial logistic regression which handles multi-class classification.

4.2 Dynamic Malware Analysis

Dynamic analysis studies a malicious program as it executes on the host machine. It uses tools like debuggers, function call tracers, machine emulators, logic analyzers, and network sniffers to capture the behavior of the program. We use two publicly available malware analysis tools to generate attributes for each malware sample. These tools make use of a sandbox, which is a controlled environment to run malicious software.

Anubis Sandbox Anubis [25] is an online sandbox which generates an XML formatted report for a malware execution in a remote environment. It generates detailed static analysis of the malware but provides less details regarding the behavior of the malware on the host machine. Since it is hosted remotely we cannot modify its settings.

Cuckoo Sandbox Cuckoo [26] is a standalone sandbox implemented using a dedicated virtual machine and more importantly can be customized to suit our needs. It generates detailed reports for both static as well as behavior analyses by watching and logging the malware while its running on the virtual machine. These behavior analyses prove to be unique indicators (behavior patterns common to a single family) for a given malware for the experiments.

4.3 Performance Evaluation

In our tests, we evaluate performance based primarily on four metrics: precision, recall, unbiased F1, and family prediction accuracy. For a given malware sample being tested, precision is the fraction of tasks the algorithm associated with the malware that were actual tasks in the ground truth. Recall, for a piece of malware, is the fraction of ground truth tasks identified by the algorithm. The unbiased F1 is the harmonic mean of precision and recall. In our results, we report the averages for precision, recall, and unbiased F1 for the number of trials performed. Our measure of family accuracy—the fraction of trials where the most probable family was the ground truth family of the malware in question—is meant to give some insight into how the algorithm performs in the intermediate steps.

5 Results

All experiments were run on Intel core-i7 operating at 3.2 GHz with 16 GB RAM. Only one core was used for experiments. Except where explicitly noted, the ACT-R parameters were fixed as per Table 3 for all experiments (across all datasets and sandboxes).

5.1 Mandiant Dataset

Our first set of experiments uses a dataset based on the T1 cyber espionage group as identified in the popular report by Mandiant Inc [14]. This dataset consisted of 132 real malware samples associated with the Mandiant report that were obtained from the Contagio security professional website [16]. Each malware sample belonged to one of 15 families including BISCUIT, NEWSREELS, GREENCAT and COOKIEBAG. Based on the malware family description [14], we associated a set of tasks with each malware family (that each malware in that family was designed to perform). In total, 30 malware tasks were identified for the given malware samples (Table 2). On average, each family performed nine tasks.

We compared the four machine learning approaches with the rule-based and instance-based ACT-R models (ACTR-R and ACTR-IB respectively). We also submitted the samples to the Cynomix tool for automatic detection of capabilities. These detected capabilities were then manually mapped to the tasks from the Mandiant report. Precision and recall values were computed for the inferred adversarial tasks. On average the machine learning approaches predicted nine tasks per sample, ACTR-R predicted nine tasks per sample and ACTR-IB predicted ten tasks. On the other hand, Cynomix was able to detect on average only four tasks.

Leave One Out Cross-Validation (LOOCV)
In leave one out cross validation, for n malware samples, train on $n - 1$ samples and test on the remaining one. This procedure was repeated for all samples and the results were averaged. We performed this experiment using both sandboxes and compared the results (Table 4).

The average F1 increases by 0.03 when we use the attributes generated by the Cuckoo sandbox instead of Anubis. The statistical significance results are as follows: for ACTR-IB (t (132) = 1.94, $p = 0.05$), ACTR-R (t (132) = 1.39, $p = 0.16$), RF (t (132) = 0.56, $p = 0.57$), SVM (t (132) = 1.95, $p = 0.05$), LOG-REG (t (132) = 1.82, $p = 0.07$), NB (t (132) = 1.79, $p = 0.08$) and DT (t (132) = 0.83, $p = 0.4$). But the significant improvement was in the family prediction values with ACTR-IB improving by 0.12 from 0.81 to 0.93 (t (132) = 3.86, $p < 0.001$) and ACTR-R by 0.15 from

Table 4 Performance comparison of Anubis and Cuckoo Sandbox (Bold values indicates best performance)

Method	Anubis (F1)	Cuckoo (F1)	Anubis (Family)	Cuckoo (Family)
DT	0.80	0.80	0.59	0.63
NB	0.71	0.74	0.30	0.40
LOG-REG	0.82	0.85	0.65	0.84
SVM	0.86	0.90	**0.85**	0.86
RF	0.89	0.89	0.82	0.86
ACTR-R	0.85	0.88	0.73	0.89
ACTR-IB	**0.93**	**0.96**	0.81	**0.93**

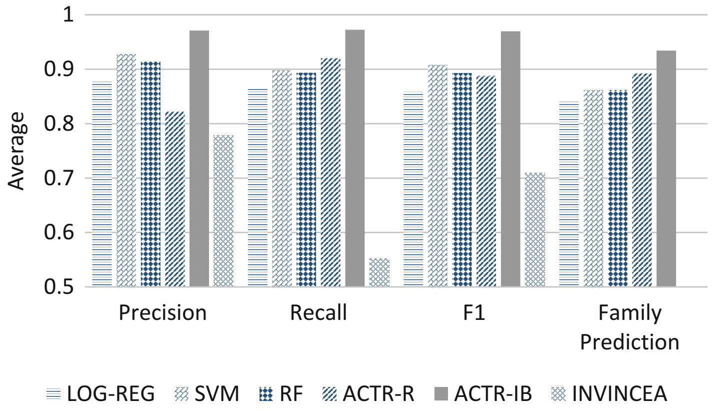


Fig. 1 Average precision, recall, F1 and family prediction comparisons using cuckoo sandbox for LOG-REG, RF, SVM, ACTR-R, ACTR-IB and INVINCEA

0.72 to 0.87 (t (132) = 3.78, $p < 0.001$) outperforming all other methods. Since having behavior analysis helps in better task prediction as seen from the comparison experiment, we use cuckoo sandbox for rest of our experiments.

Figure 1 compares the performance of the five best performing methods from Table 1 and compares it with the Cynomix tool of Invincea industries. ACTR-IB outperformed LOG-REG, SVM, RF and ACTR-R; average F1 = 0.97 vs 0.85 (t (132) = 7.85, $p < 0.001$), 0.9 (t (132) = 4.7, $p < 0.001$), 0.89 (t (132) = 5.45, $p < 0.001$) and 0.88 (t (132) = 5.2, $p < 0.001$) respectively. Both the proposed cognitive models and machine learning techniques significantly outperformed the Cynomix tool in detecting the capabilities (tasks).

These three approaches (LOG-REG, SVM, RF) were also evaluated with respect to predicting the correct family (before the tasks were determined). ACTR-IB outperformed LOG-REG, SVM, RF and ACTR-R; average family prediction = 0.93 vs 0.84 (t (132) = 3.22, $p < 0.001$), 0.86 (t (132) = 3.13, $p < 0.001$), 0.86 (t (132) = 3.13, $p < 0.001$) and 0.89 (t (132) = 2.13, $p = 0.03$) respectively. The Cynomix tool from Invincea does not have the capability to predict families.

Task Prediction Without Inferring Families

In the proposed models we infer the malware family first and then predict the tasks associated with that family. However, differences over "ground truth" for malware families in the cyber-security community calls for a direct inference of tasks without dependence on family prediction. In this section we adapt the models to predict tasks directly without inferring the family.

Figure 2 shows the performance of the cognitive and machine learning models without inferring the families. There is no difference in the performance of ACTR-IB and ACTR-R approaches as compared to Fig. 2 where we use families. On the other hand, direct task prediction reduces the F1 measure of machine learning techniques on average by almost 0.1. This is due to the fact that, now instead

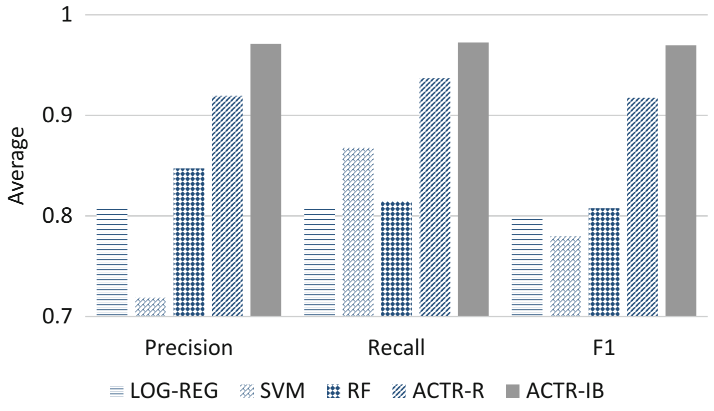


Fig. 2 Average precision, recall, and F1 comparisons for LOG-REG, RF, SVM, ACTR-R and ACTR-IB for Mandiant without inferring families

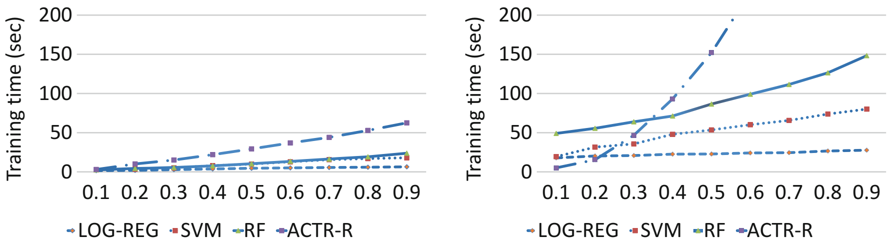


Fig. 3 Training time for LOG-REG, SVM, RF and ACTR-R with(left)/without(right) inferring families

of having a single classifier for each family we have multiple classifiers for each task that a malware sample is designed to perform. This not only degrades the performance but also adds to the training time for these methods (including the ACT-R rule-based approach). We compare the training time with increase in training data for task prediction with/without inferring families. Inferring families first reduces the training time (Fig. 3 (left)). On the other hand, predicting tasks directly significantly increases the training time for the machine learning methods along with the rule-based ACT-R approach (Fig. 3 (right)). Due to the issues with respect to performance and training time, we consider inferring families first for the rest of the experiments. An important point to note is that this has no effect on the Instance-based model for both performance and computation time.

Parameter Exploration

We now discuss two system parameters that control the performance of the ACT-R instance based model namely the stochastic noise parameter (s) and the activation threshold (τ). We use the mandiant dataset to perform this evaluation. The parameter s takes values between 0.1 and 1 (typical values range from 0.1 to 0.3). The value

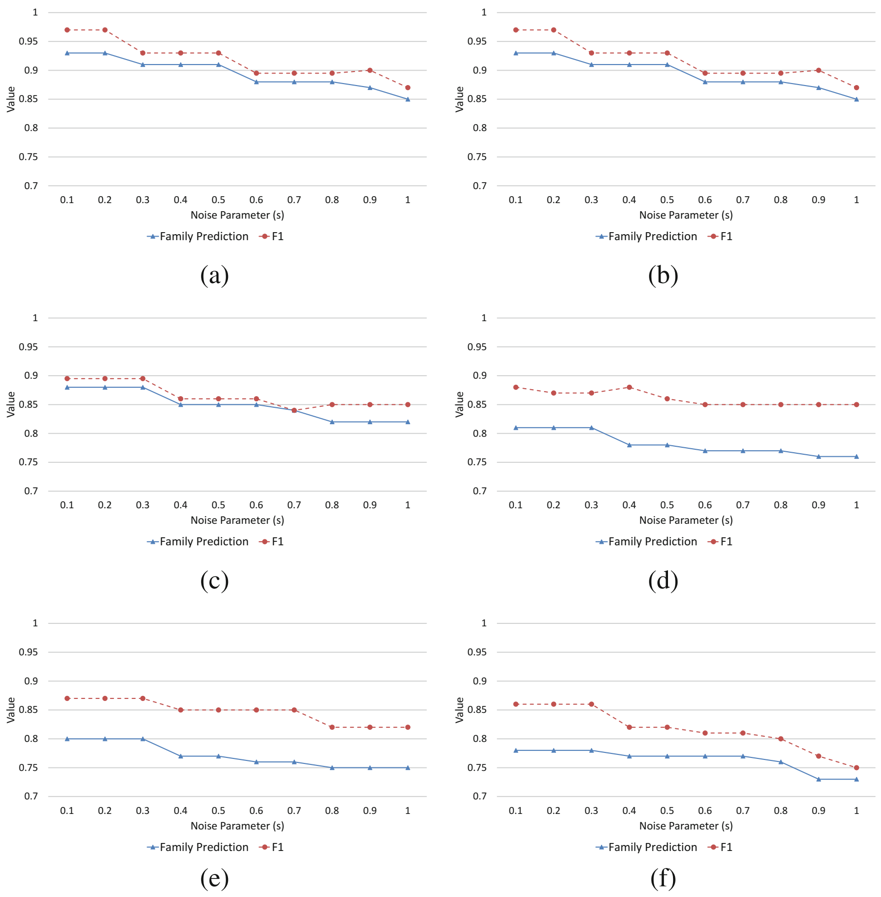


Fig. 4 Family prediction and F1 value for different threshold and noise parameters values. (**a**) $\tau = -20$. (**b**) $\tau = -10$. (**c**) $\tau = 0$. (**d**) $\tau = 5$. (**e**) $\tau = 10$. (**f**) $\tau = 15$

of the activation threshold depends on the application. Figure 4 shows the variation of family prediction accuracy and F1 score with respect to different noise parameter values and for different activation thresholds. The parameter s is used to compute the variance of the noise distribution and retrieval probability of sample in memory. Larger value of s triggers the retrieval of poor matching samples, which leads to lower family prediction and F1 scores. As seen in Fig. 4, as the value of s increases the performance decreases. On the other hand, the activation threshold dictates how many closely matched samples will be retrieved from memory. For high values of τ the performance decreases as many fewer samples are retrieved. For lower values of τ we end up retrieving almost all the samples in the training data, hence the performance does not decrease as τ decreases, but it adds to the computational cost of retrieving high number of samples which is not desirable. We get the best performance for $\tau = -10$ and $s = 0.1$. Even $s = 0.2$ is almost as good as 0.1 providing some advantages in terms of stochasticity ensuring robustness.

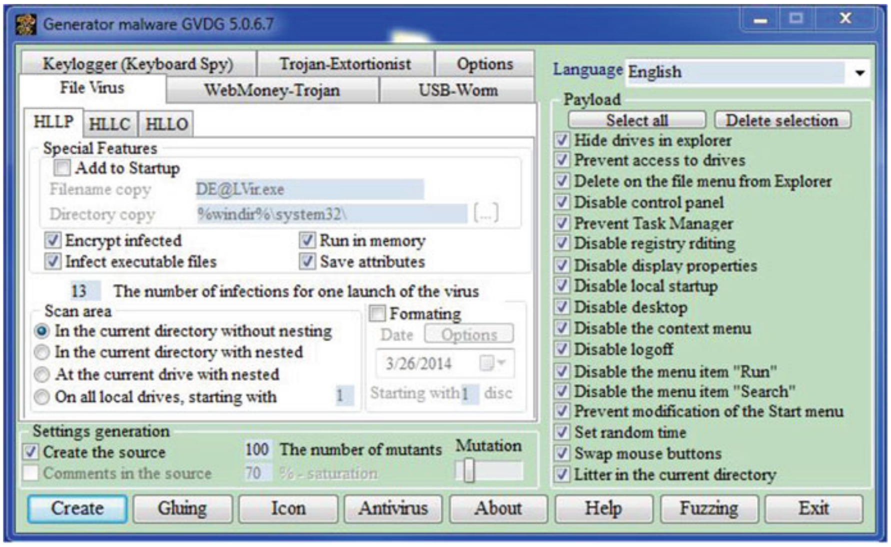
We keep the base-level constant (β) and mismatch penalty (mp) values constant. As explained earlier the base-level constant trades off directly against the retrieval threshold, and the mismatch penalty against the activation noise, respectively, so it makes sense to vary only one of the pair.

5.2 GVDG Dataset

GVDG is a malware generation tool designed for the study of computer threats [15]. It is capable of generating the following malware threats:

- File-virus
- Key-Logger
- Trojan-Extortionist
- USB-Worm
- Web Money-Trojan

Figure 5 shows the GVDG user interface used for the generation of malware samples. We can select the carrier type and the tasks that we want the malware sample to perform on the host machine. The tasks are represented as payloads, while carrier is a functional template which can be modified to execute the tasks desired by the user on the host system. In generating datasets with GVDG, we specify



Special Features
☐ Add to Startup
Filename copy DE@LVir.exe
Directory copy %windir%\system32\ [...]
✓ Encrypt infected
✓ Run in memory
✓ Infect executable files
✓ Save attributes
13 The number of infections for one launch of the virus

Scan area
● In the current directory without nesting
○ In the current directory with nested
○ At the current drive with nested
○ On all local drives, starting with 1

☐ Formating
Date Options
3/26/2014
Starting with 1 disc

Settings generation
✓ Create the source
☐ Comments in the source
100 The number of mutants
70 % - saturation
Mutation

Language English

Payload
✓ Hide drives in explorer
✓ Prevent access to drives
✓ Delete on the file menu from Explorer
✓ Disable control panel
✓ Prevent Task Manager
✓ Disable registry rditing
✓ Disable display properties
✓ Disable local startup
✓ Disable desktop
✓ Disable the context menu
✓ Disable logoff
✓ Disable the menu item "Run"
✓ Disable the menu item "Search"
✓ Prevent modification of the Start menu
✓ Set random time
✓ Swap mouse buttons
✓ Litter in the current directory

Fig. 5 GVDG user interface

families based on sets of malware with the same tasks. Whether or not a family consists of malware with the same carrier depends on the experiment. Further, GVDG also has an option to increase "mutation" or variance among the samples. We perform experiments analyzing the performance of the proposed methods when the generated samples belong to different carrier and same carrier types, as well as when the samples are encrypted and mutated making task prediction difficult. In all the experiments we consider 60% of the data for training and 40% for testing. The results are averaged across ten trials. The Cynomix tool from Invencia was unable to detect any tasks for the GVDG dataset, primarily due to its inability to find public source documents referencing GVDG samples and also unable to generalize from similar samples.

Different Carriers

In this experiment, we generated 1000 samples for each carrier type with low mutation. On average each carrier type performs seven tasks (payloads). Hence each carrier represents one family for this experiment. Both random forest and ACTR-IB model were able to predict the tasks and family with F1 measure of 1.0 outperforming LOG-REG 1 vs 0.91, SVM 1 vs 0.95 and ACTR-R 1 vs 0.95. All results are statistical significant with (t (1998) $\geq$ 8.93, $p < 0.001$) (Fig. 6). Also for family prediction ACTR-IB and RF outperformed LOG-REG 1 vs 0.92, SVM 1 vs 0.92 and ACTR-R 1 vs 0.95 (t (1998) $\geq$ 8.93, $p < 0.001$).

These results are not surprising given that different carrier(family) types have high dissimilarity between them. Also, samples belonging to the same carrier have on average 60% of similar attributes. Figure 7 shows the similarity between the carrier types. The similarity between families is calculated in the same way as ACTR-IB partial matching with 0 indicating complete match while -1 complete mismatch.

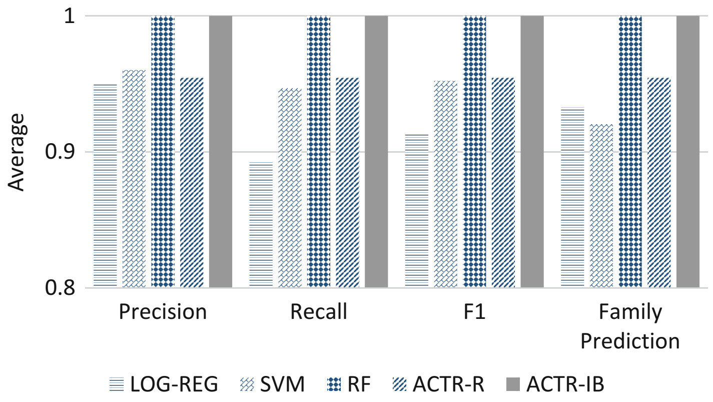


Fig. 6 Average precision, recall, F1 and family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for different carrier samples

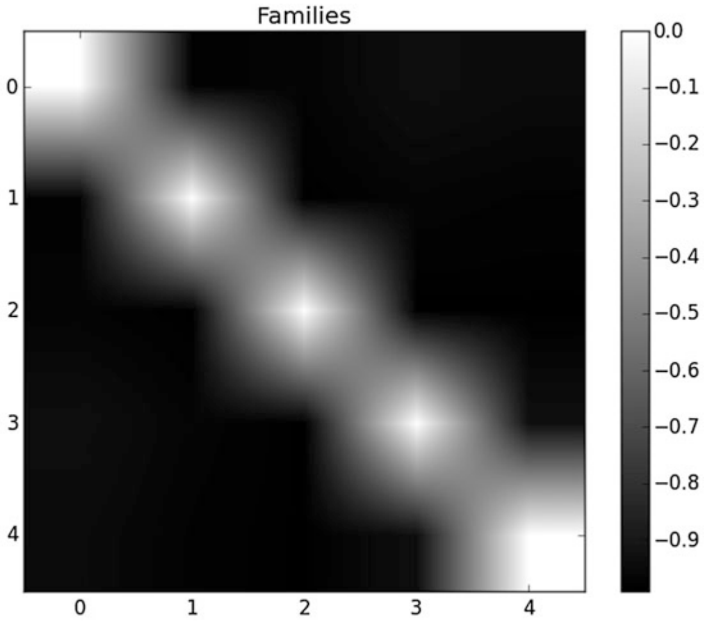


Fig. 7 Similarity matrix for five different carriers

Different Carriers-Mutation

For this case, we generate the same samples as in the previous experiment but with maximum mutation between samples belonging to the same carrier. We generated 1000 samples for each carrier with maximum mutation. In this case ACTR-IB had an average F1 of 1 outperforming LOG-REG 1 vs 0.83, SVM 1 vs 0.88, RF 1 vs 0.96 and ACTR-R 1 vs 0.92 (t (1998) $\geq$ 7, $p < 0.001$) (Fig. 8). Also for family prediction ACTR-IB outperformed LOG-REG 1 vs 0.85, SVM 1 vs 0.88, RF 1 vs 0.95 and ACTR-R 1 vs 0.92 (t (1998) $\geq$ 7, $p < 0.001$).

High mutation induces high variance between samples associated with the same carrier making the classification task difficult. High mutation samples belonging to same carrier have only 20% of common attributes as compared to 60% for low mutation.

Less Training Data

In order to see how the cognitive models perform with less training data, we repeated the different-carrier mutation experiment with 10% of the training data selected uniformly at random (300 samples). Even with less training data ACTR-IB had an average F1 of 0.93 outperforming LOG-REG 0.93 vs 0.71, SVM 0.93 vs 0.6, RF 0.93 vs 0.83 and ACTR-R 0.93 vs 0.88 (t (1998) $\geq$ 2.89, $p \leq 0.001$) (Fig. 9). Also for family prediction ACTR-IB outperformed LOG-REG 0.91 vs 0.73 (t (1998) = 19.3, $p < 0.001$), SVM 0.91 vs 0.58, RF 0.91 vs 0.79 and ACTR-R 0.91 vs 0.88 (t (1998) $\geq$ 2.05, $p \leq 0.04$).

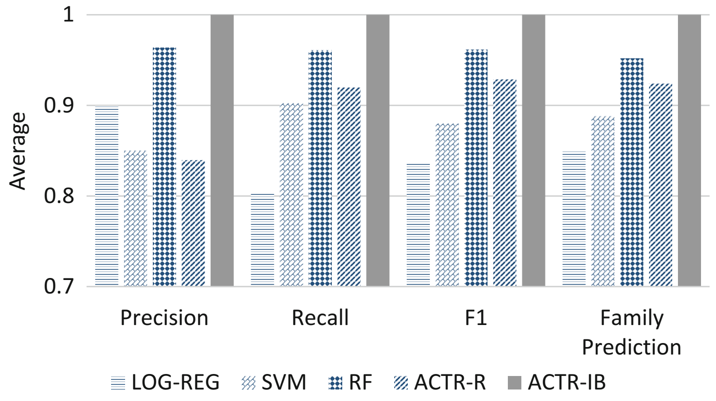


Fig. 8 Average precision, recall, F1 and family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for different carrier mutated samples

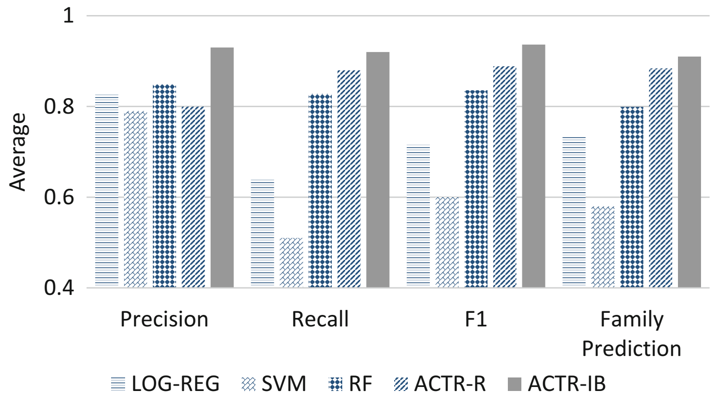


Fig. 9 Average precision, recall, F1 and family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for less training data

Different Carriers: Low-High Mutation

For this case, we consider the low mutation samples as training data and the high mutation samples as testing. Figure 10 shows the comparison results. ACTR-IB had an average F1 of 0.96 outperforming LOG-REG 0.96 vs 0.83, SVM 0.96 vs 0.92, RF 0.96 vs 0.93 and ACTR-R 0.96 vs 0.88 (t (2498) $\geq$ 15.7, $p < 0.001$) (Fig. 10). Also for family prediction ACTR-IB outperformed LOG-REG 0.96 vs 0.81, SVM 0.96 vs 0.92, RF 0.96 vs 0.94 and ACTR-R 0.96 vs 0.88 (t (2498) $\geq$ 7, $p < 0.001$).

Leave One Carrier Out Cross-Validation

To see how the models generalize to unseen malware family(carrier), we performed a leave-one-carrier-out comparison, where we test the models against one previously unseen malware carrier. ACTR-IB performs better or on par with all other baseline

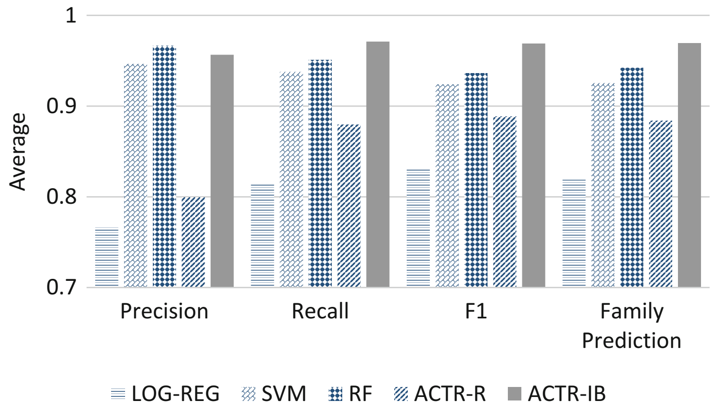


Fig. 10 Average precision, recall, F1 and family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for low-high mutated samples

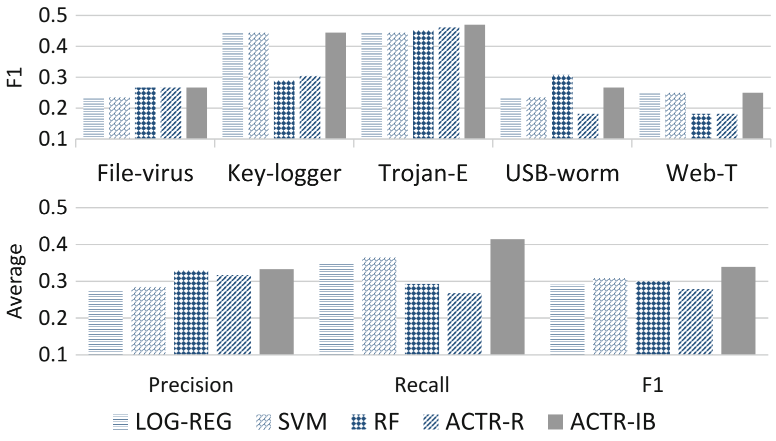


Fig. 11 Average F1 values for five malware carriers (above) and the average precision, recall and F1 across all carriers (below) for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for leave-one-carrier-out

approaches for all the carriers. It clearly outperforms all the approaches in recalling most of the actual tasks (40%) (Fig. 11). ACTR-IB has shown to generalize for unseen malware families [10]. This case is difficult given the fact that the test family is not represented during training, hence task prediction depends on associating the test family with the training families that perform similar tasks.

Same Carrier

As seen in the previous experiments, different carrier types makes the task easier because of less similarity between them. We now test the performance, on same carrier type performing exactly one task. Since there are 17 tasks in the GVDG

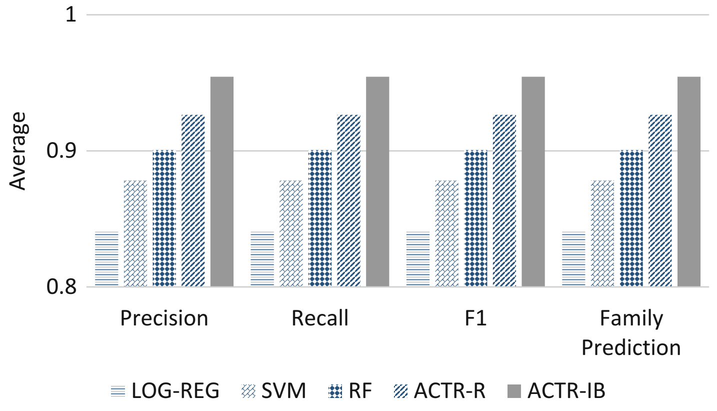


Fig. 12 Average precision, recall, F1 and family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for unencrypted same carrier samples

tool, we generate 100 samples for each task for carrier type File-virus. In this experiment each task represents one family. Thus in total we have 1700 samples. We do the 60–40 split experiment. From Fig. 12 ACTR-IB had an average F1 of 0.95 outperforming LOG-REG 0.95 vs 0.84, SVM 0.95 vs 0.87, RF 0.95 vs 0.90 and ACTR-R 0.95 vs 0.92 (t $(678) \geq 1.52$, $p \leq 0.13$). Since each family performs exactly one task the family prediction is similar to F1. Using the same carrier for each payload makes the task difficult as can be seen from the similarity matrix for the 17 payloads (Fig. 13).

Same Carrier-Encryption

The GVDG tool provides the option for encrypting the malware samples for the File-virus carrier type. We use this option to generate 100 encrypted malware samples for each task(payload) and use them as test data with the unencrypted versions from the same carrier experiment as training samples. From Fig. 14 ACTR-IB had an average F1 of 0.9 outperforming LOG-REG 0.9 vs 0.8, SVM 0.9 vs 0.8, RF 0.9 vs 0.74 and ACTR-R 0.9 vs 0.88 (t $(1698) \geq 2.36$, $p \leq 0.02$). Encrypting malware samples morphs the task during execution making it difficult to detect during analysis. Hence the drop in performance as compared to non-encrypted samples. We note that SVM performs better than RF likely because it looks to maximize generalization.

Runtime Analysis

Table 5 shows the classifier run times for the experiments. Machine learning techniques are faster but have large training times, which increase almost linearly with the size of the knowledge base. Hence updating the knowledge base is computationally expensive for these methods, as it has to re-estimate the parameters every time. The same notion holds true for ACTR-R, since computing the rules during training phase is expensive as can be seen from the large training times. ACTR-IB on the other hand, has no explicit training phase, so the only time cost

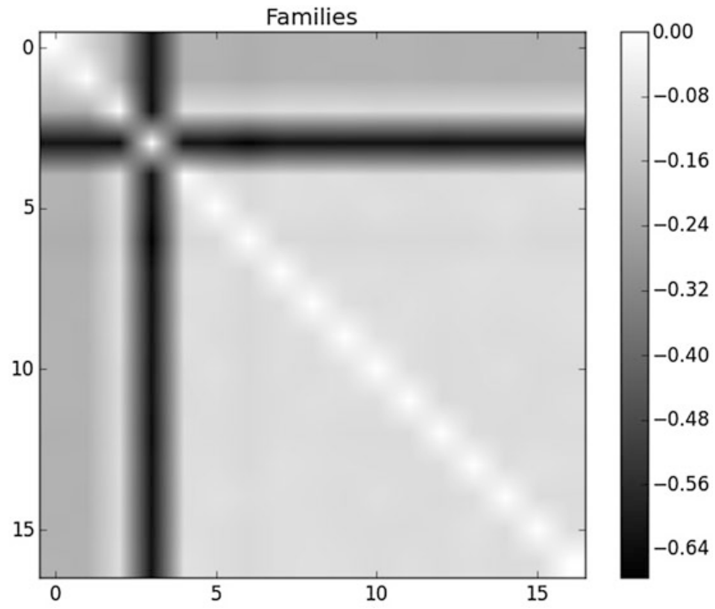


Fig. 13 Similarity matrix for 17 versions of the same carrier

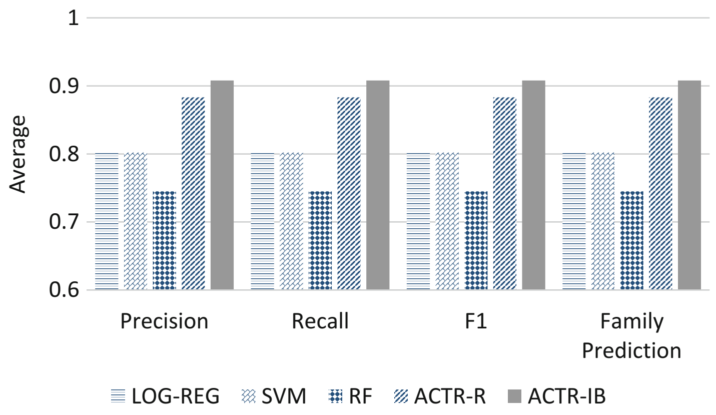


Fig. 14 Average precision, recall, F1 and family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for encrypted same carrier samples

is during testing. In fact ACTR-IB is faster than SVM and RF for same/encrypted carrier experiments.

Scaling of Instance-Based Model

Finally to conclude the GVDG experiments, we run ACTR-IB on a combination of all the above variations of dataset to highlight the space requirements for the learning model. The dataset comprises of five different carriers with low/high muta-

Table 5 Classifier run times

Experiment	Model	Train (s)	Test (s)
Different carriers	LOG-REG	202	7
	SVM	250	50
	RF	280	30
	ACTR-R	6443	143
	ACTR-IB	–	453
Mutated carriers	LOG-REG	214	18
	SVM	260	63
	RF	303	85
	ACTR-R	7223	185
	ACTR-IB	–	465
Same carriers	LOG-REG	152	4.22
	SVM	270	38
	RF	290	55
	ACTR-R	4339	120
	ACTR-IB	–	205
Encrypted carriers	LOG-REG	180	15
	SVM	300	80
	RF	353	110
	ACTR-R	6103	180
	ACTR-IB	–	365

tion (10,000 samples) and same carrier encrypted/non-encrypted (3400 samples). Based on the tasks they perform we have in total 22 families represented by 13,400 samples. The analysis reports generated by cuckoo take up 4 GB of disk space for the samples. We significantly reduce the size to 600 MB by parsing the analysis reports and extracting attributes. We set aside 10% of the samples for testing (1340) and iteratively add 10% of the remaining data for training. Table 6 gives a summary of the average F1 measure and testing time for ACTR-IB. The results are averaged across ten trials. There is a steady increase in performance till we reach 40% of the training data, after that the F1 measure remains almost constant. This experiment clearly indicates the ability of the ACTR-IB to learn from small amount of representation from each family, significantly reducing the size of the knowledge base required for training. We are also looking into techniques to reduce the time requirements of instance-based learning algorithm (e.g., Andrew Moore explored efficient tree-based storage). There are also techniques for reducing space requirements, [27] merged training instances in the ACT-R-Gammon model and obtained considerable space savings at little performance cost.

Table 6 Summary of ACTR-IB results

Fraction of training data	F1 measure	Test time (s)
0.1	0.77	418
0.2	0.82	839
0.3	0.90	1252
0.4	0.97	1676
0.5	0.97	2100
0.6	0.97	2525
0.7	0.97	2956
0.8	0.98	3368
0.9	0.98	3787
1.0	0.98	4213

5.3 *MetaSploit*

MetaSploit is a popular penetration testing tool used by security professionals to identify flaws in the security systems by creating attack vectors to exploit those flaws [28]. Penetration testing may also be defined as the methods an attacker would employ to gain access to security systems. Hence identifying the tasks the exploit was designed to perform is important to counter the exploit.

For this experiment we generate exploits that attacks windows operating systems. Each exploit has a set of tasks associated with it. The tasks include setting up tcp & udp back-door connections, adding unauthorized users to the system, modifying root privileges, download executables and execute them on the local machine, prevent writing of data to disk, deleting system folders, copying sensitive information etc. We generated 4 exploit families with 100 samples each performing on average 4 tasks. We induced mutation between samples belonging to the same family making the classification task difficult. We perform a 60–40 split training-testing experiment and average the results across ten trials. From Fig. 15, ACTR-IB had an average F1 of 0.86 outperforming LOG-REG 0.86 vs 0.62, SVM 0.86 vs 0.82, RF 0.86 vs 0.82, ACTR-R 0.86 vs 0.81 and INVINCEA 0.86 vs 0.8 (t (158) $\geq$ 1.94, $p \leq 0.05$). Also for family prediction ACTR-IB outperformed LOG-REG 0.8 vs 0.7, SVM 0.8 vs 0.72, RF 0.8 vs 0.72 and ACTR-R 0.8 vs 0.71 (t (158) $\geq$ 2.53, $p \leq 0.01$).

5.4 *Discussion*

We evaluated the two proposed cognitive models on three different datasets under various operational conditions (mutated and encrypted malware samples). The instance based model performs on par or better than the rule-based model and standard machine learning approaches. The performance improvement can be attributed to different ACT-R modules (partial matching and spreading activation) that model different aspects of the malware sample. Partial matching computes the

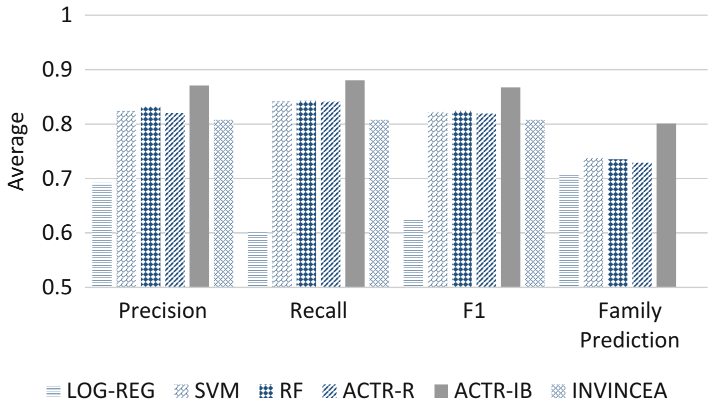


Fig. 15 Average precision, recall, F1 and family prediction comparisons for LOG-REG,SVM, RF, ACTR-R and ACTR-IB for metasploit samples

similarity between malware samples, while spreading activation identifies attributes that are indicative of a given malware family and the tasks that it is designed to perform. Hence, in cases where the training data is significantly less (as in one of the GVDG experiments), the proposed model is able to identify attributes representative of a particular malware family making the goal of correct task prediction better as compared to standard machine learning approaches which do require more training data for better generalization.

Experiments on the GVDG dataset under conditions of mutation and encryption provide further insights in the working of the cognitive models. For mutation, the malware samples used for training differ significantly from the ones used for testing. In this case the partial matching module does not contribute much towards the activation function but in turn the spreading activation is able to identify attributes that represent tasks that the malware sample is designed to perform thus making the correct task prediction in majority of the test cases. A similar behavior is observed for the encryption experiment as well. For the GVDG experiment with less training data shows how well the cognitive models are able to generalize with less training data which is difficult for standard machine learning approaches.

5.5 *Task Prediction from Hacker Activities*

In all the experiments discussed so far, the tasks associated with a given piece of malware are predefined and do not change with time. In this section, we try to map the tasks that a hacker is trying to achieve from the activities it performs on a compromised system. For the entire experiment only one malware is used whose sole purpose is to create a tcp backdoor connection to let the hacker have access to the system. We evaluate the test samples only using ACTR-IB and not other machine

Table 7 Summary of ACTR-IB results

Subject	Average precision	Average recall	Average F1
Hacker-1	0.8	0.85	0.83
Hacker-2	0.85	0.85	0.85

learning methods. The goal of this experiment is to demonstrate how the system can deal with real time hacker activities on a compromised system.

The experimental setup is as follows. We keep the Cuckoo sandbox running on the system by executing the malware. This will create a connection between the hacker and the system. Once the hacker gains control of the machine, he can perform operations in order to achieve his objectives. We treat these objectives as the tasks that the hacker wants to complete on the system. Once these tasks are completed, Cuckoo generates an analysis report detailing the behavioral analysis of the hacker. However, these analysis are too detailed and do not provide a clear picture of the main tasks of the hacker on the machine. Hence, traditionally, this will often require an expert security analyst to go through large analysis results to determine the task, which is often time consuming. But instead we can feed the analysis report to the ACTR-IB model to get a prediction of the hacker tasks. For this experiment we use the Metasploit dataset discussed in Sect. 5.3 as the knowledge base for the instance based approach. For the test set we generate samples in real time with hackers trying to achieve their goals (tasks) on the compromised system. This test also illustrates how well our model generalizes, as we are identifying hacker behavior using historical data that was not generated by the hacker—or even a human in this case. We consider two hackers, who are given a list of the payloads (tasks) to complete from the list mentioned in Sect. 5.3. They always perform a fraction of the tasks assigned to them at a given time instance and then the model is tested on predicting these tasks.

We generate ten such attacks, five from each hacker. Each attack consists of achieving five tasks on average. We note that for each of the test sample the malware used is the same. ACTR-IB results are presented in Table 7. The results are averaged for each hacker across test samples. Table 8 shows the actual and predicted tasks for Hacker-1 for five different attack instances. The results for Hacker-2 were analogous.

6 Related Work

Identification of Malicious Software The identification of whether or not binary is malicious [29, 30] is an important related, yet distinct problem from what we study in this chapter and can be regarded as a "first step" in the analysis of suspicious binaries in the aftermath of a cyber-attack. However, we note that as many pieces of malware are designed to perform multiple tasks, that successful identification of a

Table 8 Actual and predicted Hacker-1 attacks

Attack instance	Actual tasks	Predicted tasks
1	Setup backdoor connection modify root privileges uninstall program copy files	Setup backdoor connection modify root privileges uninstall program delete system files prevent access to drive
2	Setup backdoor connection modify root privileges download executables execute files copy files	Setup backdoor connection modify root privileges download executables execute files delete files
3	Setup backdoor connection modify root privileges add unauthorized users start keylogging uninstall program delete files prevent access to drives	Setup backdoor connection modify root privileges add unauthorized users start keylogging uninstall program delete files
4	Setup backdoor connection add unauthorized users prevent writing data to disk delete files copy files	Setup backdoor connection add unauthorized users prevent writing data to disk delete files modifying root privileges prevent access to drives
5	Setup backdoor connection download executables execute files start keylogging	Setup backdoor connection download executables execute files start keylogging

binary as malicious does not mean that the identification of its associated tasks will be a byproduct of the result.

Malware Family Classification There is a wealth of existing work on malware family identification [2–4, 6, 31–33]. The intuition here is that by identifying the family of a given piece of malware, an analyst can then more easily determine what it was designed to do based on previously studied samples from the same family. However, malware family classification has suffered from two primary drawbacks: (1) disagreement about malware family ground truth as different analysts (e.g. Symantec and McAfee) cluster malware into families differently; and (2) previous work has shown that some of these approaches mainly succeed in "easy to classify" samples [5, 6], where "easy to classify" is a family that is agreed upon by multiple malware firms. In this chapter, we infer the specific tasks a piece of malware was designed to carry out. While we do assign malware to a family as a component of our approach, it is not the focus of our comparison (though we show family prediction results as a side-result). Further, we also describe and evaluate a variant of our instance-based method that does not consider families and yields a comparable performance.

Malware Task Identification With regard to direct inference of malware tasks, the major related work includes the software created by the firm Invincea [7] for which we have included a performance comparison. Additionally, some of the ideas in this chapter were first introduced in [10–12]. However, these work primarily focused on describing the intuitions behind the cognitive modeling techniques and only included experimental evaluation on two datasets (the Mandiant APT1 and GVDG datasets). The experimental evaluation in this chapter includes additional experiments for the GVDG dataset to consolidate the previous experiments. Also algorithm analysis and parameter exploration are provided for the cognitive models. In addition we introduce a popular penetration tool used by security analyst Metasploit and present new results on this tool.

7 Conclusion

In this chapter, we introduced an automated method that combines dynamic malware analysis with cognitive modeling to identify malware tasks. This method obtains excellent precision and recall—often achieving an unbiased F1 score of over 0.9—in a wide variety of conditions over three different malware sample collections and two different sandbox environments—outperforming a variety of baseline methods.

Currently, our future work has three directions. First, we are looking to create a deployed version of our approach to aide cyber-security analysts in the field. Second, we look to enhance our malware analysis to also include network traffic resulting from the sample by extending the capabilities of the sandbox. Finally, we also look to address cases of highly-sophisticated malware that in addition to using encryption and packing to limit static analysis and employ methods to "shut down" when run in a sandbox environment [34]. We are exploring multiple methods to address this such as the recently introduced technique of "spatial analysis" [35] that involves direct analysis of a malware binary.

References

1. M. Sikorski, A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, 1st edn. (No Starch Press, San Francisco, 2012)
2. U. Bayer, P.M. Comparetti, C. Hlauschek, C. Kruegel, E. Kirda, Scalable, behavior-based malware clustering, in *NDSS*, vol. **9** (Citeseer, 2009), pp. 8-11
3. J. Kinable, O. Kostakis, Malware classification based on call graph clustering. J. Comput. Virol. **7**, 233–245 (2011)
4. D. Kong, G. Yan, Discriminant malware distance learning on structural information for automated malware classification, in *Proceedings of the 19th ACM SIGKDD. KDD '13, New York* (ACM, New York, 2013), pp. 1357–1365
5. P. Li, L. Liu, D. Gao, M.K. Reiter, On challenges in evaluating malware clustering, in *International Workshop on Recent Advances in Intrusion Detection* (Springer, Berlin, 2010), pp. 238–255
6. R. Perdisci, ManChon, Vamo: towards a fully automated malware clustering validity analysis, in *Proceedings of the 28th Annual Computer Security Applications Conference* (ACM, New York, 2012), pp. 329–338
7. Invencia, Crowdsource: crowd trained machine learning model for malware capability detection (2013). http://www.invincea.com/tag/cynomix/

8. Kaspersky, Gauss: abnormal distribution (2012). https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/20134940/kaspersky-lab-gauss.pdf
9. C. Lebiere, P. Pirolli, R. Thomson, J. Paik, M. Rutledge-Taylor, J. Staszewski, J.R. Anderson, A functional model of sensemaking in a neurocognitive architecture. Comput. Intell. Neurosci. 5:5–5:5 (2013)
10. C. Lebiere, S. Bennati, R. Thomson, P. Shakarian, E. Nunes, Functional cognitive models of malware identification, in *Proceedings of ICCM, ICCM 2015, Groningen, April 9–11* (2015)
11. R. Thomson, C. Lebiere, S. Bennati, P. Shakarian, E. Nunes, Malware identification using cognitively-inspired inference, in *Proceedings of BRIMS, BRIMS 2015, Washington DC, March 31–April 3* (2015)
12. E. Nunes, C. Buto, P. Shakarian, C. Lebiere, R. Thomson, S. Bennati, J. Holger, Malware task identification: a data driven approach, in *Proceedings of International Symposium on Foundation of Open Source Intelligence and Security Informatics (FOSINT-SI)* (IEEE, Piscataway, 2015)
13. Rapid7, Metasploit: penetration testing software (2003). http://www.metasploit.com/
14. D. McWhorter, APT1: exposing one of China's cyber espionage units (2013). http://Mandiant.com
15. GVDG, Generator malware GVDG (2011)
16. Mandiant, Mandiant APT1 samples categorized by malware families. Contagio Malware Dump (2013)
17. J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, Y. Qin, An integrated theory of mind. Psychol. Rev. **111**, 1036–1060 (2004)
18. C. Gonzalez, J.F. Lerch, C. Lebiere, Instance-based learning in dynamic decision making. Cogn. Sci. **27**(4), 591–635 (2003)
19. R.S. Sutton, A.G. Barto, *Introduction to Reinforcement Learning*, 1st edn. (MIT Press, Cambridge, 1998)
20. T.J. Wong, E.T. Cokely, L.J. Schooler, An online database of ACT-R parameters: towards a transparent community-based approach to model development, in *Proceedings of the 10th International Conference on Cognitive Modeling* (Citeseer, 2010), pp. 282–286
21. D. Bothell, Act-r 6.0 reference manual (2004). http://act-r.psy.cmu.edu/actr6/reference-manual.pdf
22. L. Breiman, Random forests. Mach. Learn. **45**(1), 5–32 (2001)
23. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**, 273–297 (1995)
24. C.C. Chang, C.J. Lin, Libsvm: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 27:1–27:27 (2011)
25. ISEC-Lab, Anubis: analyzing unknown binaries (2007). http://anubis.iseclab.org/
26. C. Guarnieri, A. Tanasi, J.B.M.S., Cuckoo sandbox (2012). http://www.cuckoosandbox.org/
27. S. Sanner, J.R. Anderson, C. Lebiere, M. Lovett, *Achieving Efficient and Cognitively Plausible Learning in Backgammon* (Carnegie Mellon University, Pittsburgh, 2000)
28. J. O'Gorman, D. Kearns, M. Aharoni, *Metasploit: The Penetration Tester's Guide* (No Starch Press, San Francisco, 2011)
29. I. Firdausi, C. Lim, A. Erwin, A.S. Nugroho, Analysis of machine learning techniques used in behavior-based malware detection, in *Proceedings of the 2010 Second International Conference on ACT. ACT '10, Washington, DC* (IEEE Computer Society, Philadelphia, 2010), pp. 201–203
30. A. Tamersoy, K. Roundy, D.H. Chau, Guilt by association: large scale malware detection by mining file-relation graphs, in *Proceedings of the 20th ACM SIGKDD. KDD '14* (ACM, New York, 2014), pp. 1524–1533
31. S.S. Hansen, T.M.T. Larsen, M. Stevanovic, J.M. Pedersen, An approach for detection and family classification of malware based on behavioral analysis, in *2016 International Conference on Computing, Networking and Communications (ICNC)* (IEEE, Piscataway, 2016), pp. 1–5
32. K. Sanders, X. Wang, Malware family identification using profile signatures. US Patent 9,165,142, 20 Oct 2015

33. C. Annachhatre, T.H. Austin, M. Stamp, Hidden Markov models for malware classification. J. Comput. Virol. Hacking Tech. **11**(2), 59–73 (2015)
34. M. Lindorfer, C. Kolbitsch, P. Milani Comparetti, Detecting environment-sensitive malware, in *Proceedings of the 14th International Conference on RAID. RAID'11* (Springer, Berlin, 2011), pp. 338–357
35. D. Giametta, A. Potter, There and back again: a critical analysis of spatial analysis (2014). https://archive.org/details/ShmooCon2014_A_Critical_Review_of_Spatial_Analysi