Malware Identification Using Cognitively-Inspired Inference

Robert Thomson, Christian Lebiere and Stefano Bennati
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
412-268-6028
(thomsonr, cl, sbennati @andrew.cmu.edu)

Paulo Shakarian, Eric Nunes
Arizona State University
699 S. Mill Avenue
Tempe, AZ 85382
(shak, eric.nunes @asu.edu)

Keywords:
Malware Analysis, Inference, Instance-Based Learning, Functional Modeling, Cognitive Architectures

ABSTRACT: *Malware reverse-engineering is an important type of analysis in cybersecurity. Rapidly identifying the tasks that a piece of malware is designed to perform is an important part of reverse engineering that is generally manually performed as it relies heavily on human intuition This paper describes how the use of cognitively-inspired inference can assist in automating some of malware task identification. Computational models derived from human-inspired inference were able to reach relatively higher asymptotic performance faster than traditional machine learning approaches such as decision trees and naïve Bayes classifiers. Using a real-world malware dataset, these cognitive models identified sets of tasks with an unbiased F1 measure of 0.94. Even when trained on historical datasets of malware samples from different families, the cognitive models still maintained the precision of decision tree and Bayes classifiers while providing a significant improvement to recall.*

1. Introduction

Malware reverse-engineering is an important task for cyber-security. While large amounts of data can be sorted and filtered using machine learning techniques, identifying the tasks that a piece of malware is designed to perform is manually performed as it relies heavily on human intuition (Sikorski & Honig, 2012). 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 cyber-security expert's own particular background.

Malware classification occurs in two stages: the first is determining whether a given binary is malicious (Tamersoy, Roundy & Horng 2014; Firdausi, Lim, Erwin, & Nugroho, 2010) and then classifying this malware by family (Bayer, Comparette, Hlauschek, et al., 2011; Kinable & Kostakis, 2011; Kong & Yan, 2013). Malware family classification has suffered from two primary draw-backs: (1) disagreement about malware family *ground truth* as different analysts (e.g. Symantec and MacAfee) cluster malware into families differently; and (2) previous work has shown that some of these approaches mainly succeed in "easy to classify" samples (Perdisci, 2012; Li, Liu, Gai & Reiter, 2010), where "easy to classify" is a family that is agreed upon by multiple malware firms. In this paper, we look to infer the specific tasks a piece of malware was designed to carry out. While we do assign malware to a family, to avoid the two aforementioned issues the family partition is done probabilistically and the ground truth compared to the tasks each malware performed rather than an assignment to a family.

The ability to stably and accurately sort substantial information from the environment is a key element of human cognition. It is generally accurate even with incomplete evidence and limited feedback. It thus seems beneficial to examine features of human cognition that may guide our development of algorithms to sort through the large amounts of data generated by malware analyses.

We argue that malware identification techniques can be improved using cognitively-inspired inference. Cognitive architectures such as ACT-R (Anderson, Bothell, Byrne, et al., 2004) have previously been shown to effectively model human cognition on a variety of decision-making (Lebiere, Gonzalez, & Martin, 2007) and general intelligences tasks (Lebiere, Gonzalez, & Warwick 2009), including complex domains such as intelligence analysis (Lebiere, Pirolli, Thomson, et al., 2013). Further, due to the ability of these models to mimic human cognition, they have been shown to perform well on reasoning tasks where historical knowledge is sparse, limited, or dissimilar to the current context (Taatgen, Lebiere, & Anderson, 2006).

An example of the efficiency gained through cognitive inference is the cognitive model of backgammon that is able to learn to perform at a highly skilled level after playing a few hundred games, as opposed to tens-of-thousands to millions of games for the equivalent machine learning algorithms to reach a comparable performance (Sanner et al., 2000). The key aspect of cognitive inference that is leveraged to achieve this efficiency and capability is the combination of symbolic problem decomposition with statistical learning, made possible by the tight integration of symbolic and subsymbolic representations in cognitive architectures such as ACT-R.

In this paper we leverage the cognitively-inspired inference mechanisms in ACT-R to identify the tasks associated with a piece of malware. Using a real-world malware dataset (Mandiant Corp, 2013), our cognitive models identified sets of tasks with an unbiased F1 measure of 0.94 – significantly out-performing baseline approaches including a decision-tree and naïve Bayes classifier while using only highly scalable online learning.

2. Cognitively-Inspired Inference

Machine learning algorithms like deep learning are massively parallel and can cope with large amounts of data; however they are limited because of their relatively primitive semantics and thus are best suited for the initial filtering and structuring of data. On the other end of the spectrum, while human inference has memory and attentional limitations, 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 cognitively-inspired 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.

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. Reitter & Lebiere (2010) introduced a modeling methodology called accountable modeling that recognizes that not every aspect of a cognitive model is reflected in measurable performance. 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 paper 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. The ACT-R Cognitive Architecture

We leveraged features of the declarative memory and production system of the ACT-R architecture to complete malware identification. These systems store and retrieve information that correspond to declarative and procedural knowledge, respectively. Declarative information is the kind of knowledge that a person can attend to, reflect upon, and usually articulate in some way. Conversely, procedural knowledge consists of the skills we display in our behavior, generally without conscious awareness. Modules are encapsulated and may process information in parallel within one another. However, there are two serial bottlenecks in processing: only one production may be executed at a time, and the contents of a module can only be accessed through a buffer that can only contain one chunk at a time.

3.1 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: $P_i = (e^{A_i/s}) \big/ (\sum_j e^{A_j/s})$ where P_i is the probability that chunk *i* will be recalled, A_i is the activation strength of chunk *i*, $\sum A_j$ is the activation strength of all of eligible chunks *j*, and s is momentary noise inducing stochasticity by simulating background neural activation.

The activation of a given chunk *i* (A_i) is governed by its summed base-level activation (B_i) reflecting its recency and frequency of occurrence, spreading activation (S_i) reflecting the effects that buffer contents have on the retrieval process, partial matching score (P_i) reflecting the degree to which the chunk matches the retrieval request, and finally a noise value ($\mathcal{E}_i$) including both transient and permanent noise: $A_i = B_i + S_i + P_i + \varepsilon_i$. Sub-symbolic activations approximate Bayesian inference by framing activation as log-likelihoods, with base-level activation (B_i) as the prior, the sum of spreading activation and partial matching as the likelihood adjustment factor(s), and the final chunk activation (A_i) as the posterior.

A chunk's base-level activation is computed by summing across the number of presentations *n* for chunk *i* the log of the time t_j since the j^{th} presentation discounted by decay rate *d*, with this an optional constant β_i added to this value: $B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) + \beta_i$. Base-level activation corresponds to the Bayesian prior of a chunk's activation. A benefit of base-level activation is that it provides an automated procedure for frequency-based strengthening as well as temporal discounting.

The spread of activation (S_i) is computed by the following equation: $S_i = \sum_k \sum_j W_{kj} S_{ji}$ where elements *k* being summed over are the set of buffers in the model, elements *j* being summed over are the chunks which are in the slots of the chunk in buffer *k* (these are referred to as the sources of activation), W_{kj} is the amount of activation from sources *j* in buffer *k* weighted by parameter *W*, and S_{ji} is the strength of activation from chunk *j* to *i*. Strengths of association correspond to the Bayesian log-likelihood of chunk *i* being relevant given context elements *j*. S_{ji} is therefore defined as $log(P(i|j)/log(P(i))$.These associations are built up from experience, and they reflect how chunks co-occur in cognitive processing. The spread of activation from one cognitive structure to another is determined by weighting values *W* on the associations among chunks, which determine the rate of activation flow.

Chunks are also compared to the desired retrieval pattern using a partial matching mechanism (P_i) that subtracts from the activation of a chunk *i* its degree of mismatch M_{ki} to the desired pattern *k*, additively for each component and chunk value: $P_i = \sum_k PM_{ki}$. Both the spreading activation and partial matching mechanisms serve to automate efficient contextual priming effects.

While the most active chunk is usually retrieved, a blending process (i.e., a *blended retrieval;* see Lebiere, 1999; Wallach & Lebiere, 2003) can also be applied that returns a derived output V reflecting the similarity S_{ij} between the values of the content of all chunks i and compromise value j, weighted by their retrieval probabilities P_i reflecting their activations and similarity scores: $V = argmin \sum_i P_i (1 - S_{ij})^2$. This process enables the generation of continuous values (e.g., probabilities) in a process akin to weighted interpolation.

3.2 Procedural Knowledge

Production rules are used to represent procedural knowledge. They represent and apply cognitive skill in the current context, including how to access and modify information in buffers and transfer it to other modules. Each production rule is a set of *conditions* and *actions* which are analogous to IF-THEN rules. Conditions specify structures that are matched in buffers, and correspond to information from the external world or other internal modules. Matching production rules effectively means: if the conditions of a given production match the current state then perform the following actions.

3.3 Instance-Based Learning

Instanced-based learning (IBL) is the theory that people have a general-purpose mechanism whereby situation-action-outcome observations are stored and retrieved from memory. IBL offers constraints on explanation by grounding implicit learning within the mechanisms of a cognitive architecture. The dynamics of an instance's sub-symbolic activations (e.g., frequency and recency in the base-level equation) provide a scientifically-justified mechanism for determining which instances are likely to be retrieved for a given situation, and also can explain *why* they were retrieved and what factors came into play.

These instances are represented with slots containing the conditions (contextual cues), the decision made (an action), and the outcome of the decision (the utility of the decision). Before sufficient task-relevant knowledge is available, alternatives are evaluated using heuristics (e.g., random choice, loss-minimization, maximizing gain). Once sufficient instances are learned, decision-makers retrieve and generalize from these instances to evaluate alternatives, make a decision, and execute the task. As for learning, the generalization process is constrained by mechanisms with the cognitive architecture, in this case partial matching and blending.

The process of feedback involves updating the outcome slot of the chunk according to the post-hoc generated utility of the decision. Thus, when decision-makers are confronted with similar situations while performing a task, they gradually abandon general heuristics in favor of improved instance-based decision-making processes (Gonzalez & Lebiere, 2005). IBL methodology has been used in a number of research applications including the AFRL 711 HPW/ RHA's model of Predator operators. It can also be used to represent individual differences in experience and capacity by providing and parameterizing content from a single individual (e.g., Sanner et al., 2000; Wallach & Lebiere, 2003).

4. Malware Identification Task

We created a dataset identified by the popular malware report by Mandiant Inc. (2013). Dynamic malware analysis was performed using the ANUBIS (2014) sandbox. From the ANUBIS data, a total of 1740 malware attributes were identified (see Table 1 for a select sample).

Table 1: Sample attributes from Anubis malware sandbox

ATTRIBUTES	**INTUITION**
hasDynAttrib	Has generic attribute in the analysis
usesDll(***X***)	Malware uses a library ***X***
regAct	Conducts an activity in the registry
fileAct	Conducts an activity on a certain file
proAct	Malware initiates or terminates a process

We studied all families where there were at least 5 samples successfully processed by ANUBIS, which provided 15 families and 137 samples (see Table 2).

Table 2. Samples of malware families.

FAMILY	**NUMBER OF SAMPLES**
BISCUIT	6
BOUNCER	5
COOKIEBAG	6
GOGGLES	5
GREENCAT	22
NEWSREELS	14
STARSYPOUND	21
TABMSGSQL	7
TARSIP-ECLIPSE	7
TARSIP-MOON	5
WEBC2-BOLID	5
WEBC2-CSON	8
WEBC2-GREENCAT	6
WEBC2-HEAD	9
WEBC2-YAHOO	11

Based on malware family description, 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 malwares (see Table 3). On average, each family performed 9 tasks.

Table 3. List of malware tasks.

TASK	**INTUITION**
beacon()	beacons back to adversary's system
bruteForceSqlLogin()	uses a brute-force technique
capturesKeystrokes()	Captures keystrokes from the target
createModifyFiles()	Designed to modify target's files
createProc()	Designed to create a new process
Download()	Download files to the target

encryptedComms()	Uses encrypted communication
enumFiles()	Enumerate files on the target
enumUsers()	Enumerate users on the target
exeArbitCmds()	Execution of arbitrary commands
gatherSysInfo()	Gathers system information
maintPersist()	Maintains persistence on the target
openListenPort()	Opens a listening port on the target
procEnum()	Enumerates running processes
procTerm()	Allows termination of processes
redirNwTraffic()	Re-directs target's network traffic
sendPwdInfo()	Sends target password information
serviceEnum()	Enumerates target's services
servieManip()	Manipulates target's services
shell()	Provides adversary remote shell
smartCardMonitor()	Monitors target for smart card use
sqlQueryToAttacker()	Conducts an SQL query
SSL()	Uses SSL for communication
sysEnum()	Enumerates systems on a network
takeScreenShots()	Takes screen shots
uninstall()	Includes an uninstall routine
updateMwCfg()	Update the malware's configuration
upload()	Designed to upload files from target
usesHttp()	Uses HTTP for communications
webC2()	Uses a web-based C&C
upload	Designed to upload files from target

4.1 Decision Tree

For baseline comparison to the cognitive models, we first implemented a decision tree. This hierarchical recursive partitioning algorithm is widely used for classification problems. The decision tree finds the attribute that maximizes information gain at each split. The total entropy is the weighted (fraction of samples in each split) sum of the two entropies. The attribute that minimizes this entropy (in turn maximizing information gain) is the best split attribute. We calculated the entropy for each split to be: $E = \sum_f -p(x) \times \log p(x)$. Each node in the tree is divided into two groups, one having the best split attribute and the other which does not have that attribute. In order to avoid over-fitting, the terminating criteria was set to less than 5% of total samples (i.e., the node with less than 5% of total samples is declared as a leaf node).

During the testing phase, for a new malware sample, we start from the root of the trained tree and for each node we see if the best split attribute is present in the test sample; if yes we assign the sample to Group 1 otherwise we assign it to Group 2. We continue this procedure iteratively until we reach a leaf node. Since labels are not used during training to build the tree, the leafs may or may not be pure, thus generating a probability distribution over the malware families. This family distribution is assigned to the test samples. Tasks are then determined by summing up the probability of the families associated with the task, with a threshold set at 50%.

4.2 Naïve Bayes Classifier

Due to its similarity to ACT-R's activation equation, we decided to use a Naïve Bayes classifier as a secondary baseline approach. Naïve Bayes is a probabilistic classifier that uses Bayes theorem with an 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., the fraction of the training data belonging to each family. More specifically, given a malware sample S with a set of attributes $(a_1, a_2, \ldots\ldots a_d)$, the probability that the given sample belongs to family (f) is calculated as $P\left(\frac{f}{S}\right) = \frac{P\left(\frac{S}{f}\right) \times P(f)}{P(S)}$. For a given sample the total probability $P(S)$ doesn't vary, so we can safely ignore it. Naïve Bayes assumes that the attributes are statistically independent hence the likelihood formula can be written in the simplified form $P\left(\frac{f}{S}\right) = P(f) \times \prod_{i=1}^{d} P\left(\frac{a_i}{f}\right)$. This generates a distribution over families for a given sample.

During testing, the probability of a malware sample belonging to a family is just the product of the individual attribute belonging to that family and its prior probability. The association of the test sample with each malware family is computed, generating a distribution over malware families and the tasks associated with the sample are determined in a similar way to that of decision tree.

5. Cognitive Models

Two distinct models were created that leveraged separate parts of the activation calculus. The models are built using the inferential mechanisms of the ACT-R cognitive architecture and learn to recognize malware samples based upon a limited training schedule similar to the actual experiences of a human analyst. Given a malware sample, the model generates a probability distribution over a set of malware families, then infers a set of likely malware intents based upon that distribution. The models primarily leverage the sub-symbolic mechanisms of the ACT-R architecture, especially the activation calculus underlying retrieval from declarative memory. Each sample is represented by its set of static and dynamic attributes. The model operates in two stages: first by family, then by intent. To assign family, the model generates a probability distribution over the set of possible malware families from the activation in long-term memory of the chunks representing instances of those families. To assign intent in a second pass the model uses a similar process to generate likely intents from a representation linking each malware family to known intents. How they accomplish these two stages, specifically which representations and mechanisms are used, distinguish the two models.

A rule-based model leverages the Bayesian memory activation mechanisms. Its representation is relatively compact, involving a single chunk for each family whose associations abstract the various instances belonging to that category, but whose associations need to be computed and do not involve temporal discounting and other adaptive features (see Thomson & Lebiere, 2013). The instance-based model is based on more direct, incremental learning that accumulates malware instances in memory and leverages neurally-plausible pattern matching processes such as partial matching and blending (Lebiere et al., 2013) but is less parsimonious with storage and thus has potential scalability issues for large data sets.

5.1 Rule-Based Model

This ACT-R model is not strictly rule-based because it does not in any way include a rule that determines its judgment, e.g., in the way that the decision tree is a representation of a hierarchical decision procedure that repeatedly partitions the attribute space in subcategories. Rather, this model is called rule-based because each family is represented as a single chunk, and the subsymbolic information associated with that chunk, specifically the base-level and strengths of associations, constitute an implicit definition of belonging to that family. Those parameters can be learned incrementally, or they can be set to reflect the aggregate statistics of an entire training corpus. We followed the latter procedure. Specifically, the base-level associated with family chunk *f*, representing the a priori probability of a malware sample belonging to that family, is set to the log of the Bayesian prior *ln(p(f))*.

Similarly, the strength of association from malware attribute *a* to family *f* is set to the log-likelihood ratio *ln(p(a|f)/p(a))*. These strengths of association are multiplied by the attentional focus *Wa* associated with each attribute to determine the total activation flowing from the set of attributes associated with the current malware to the family chunk. Together, base-level and spreading activation determine the total activation of the family chunk *f*, in turn determining the probability associated with that family through the Boltzmann (softmax) distribution. As for the baseline models, intent probabilities are then determined by summing up the probabilities of families associated with that intent, with the same threshold of 50% determining a positive intent identification.

5.2 Instance-Based Model

This model follows the instance-based learning theory (IBL; Gonzalez, Lerch, and Lebiere, 2003) that is particularly relevant to modeling naturalistic decision making in complex dynamic situations. The instance-based approach is an iterative learning method that reflects the cognitive process of accumulating experiences and using them to make decisions. In this case a chunk is created for each malware instance rather than each malware family. Thus it is a straightforward instance of online learning where each new experience results in a new memory chunk. Each chunk represents the set of attributes together with the family identification. The base-level activation of each chunk is learned by to the mechanism described in the introduction. The power law decay makes it sensitive to the recency of presentation, allowing both for old malware instances to quickly decay away as well as for new ones to rapidly reach prominence.

If the same instance (i.e., same attributes and family) is presented multiple times, the base-level will also reflect the frequency of presentation. Strengths of associations are not used: rather the effect of context, as represented by the set of attributes of the current malware, will be reflected through the partial matching mechanism. The match score of a chunk to the current context will reflect the similarity between the attribute sets of the current malware sample and each instance in memory, as measured by the dot product between the respective attribute vectors. The retrieval process then extracts from the chunk its family identification. The blending mechanism computes a probability distribution over all possible family values, reflecting the activation of each instance. Thus the same factors as in the rule-based model are reflected in the computation, specifically the frequency of each malware and overlap in attribute space, but using distinct architectural mechanisms.

The second part of the process, going from probability distribution over families to intents, is entirely different. Instead of simply summing up the probabilities of each families associated with a given intent, the instance-based learning approach is also applied for this second step. This time, the instances learn to associate the probability distribution over families computed for the given malware with its actual intents. Given a new malware instance, a retrieval process matches its family probability distribution against those of previous instances, and extracts the probability of each intent using the same blending process used for generating the family probabilities. Intents reaching the 50% threshold are again selected. The key aspect of this process is that it is now sensitive to the entire probability distribution over families rather than simply a sum of its values.

6. Results

We compared both the instance-based and rule-based models against both the decision tree and naïve Bayes classifiers for both leave-one-out cross-validation and leave-one-family-out cross-validation. In the leave-one-out cross-validation, for each of the 137 malware samples, we trained 136 samples and tested on the remaining one. This procedure was repeated for all samples and the results were averaged. Similarly, in the leave-one-family-out we trained on 14 families and tested on the 15th.

Pairwise t-tests (t(136)) for the leave-one-out cross-validation showed that ***instance-based*** > ***rule-based*** > ***decision tree*** > ***naïve Bayes model*** (all $p < .01$; see Figure 1). The instance-based model outperforms baseline approaches in detecting 9 of 15 families with an average F1 difference greater than 0.3 with at least 99% confidence ($t(136) = 4$, $p = .01$), while neither the decision tree nor naïve Bayes methods significantly outperformed the instance-based model on any family. We argue that instance-based model is superior because it uses the full pattern of the probability distribution over families rather than just a sum (as in the rule-based model).

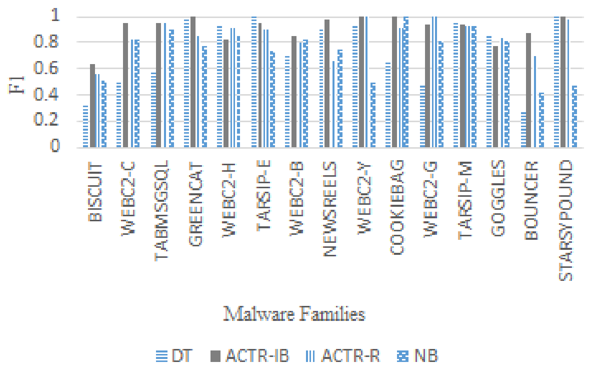

Figure 2. Leave one out cross-validation.

Similarly, pairwise t-tests for the leave one-family out cross-validation found that ***instance-based*** ~ ***rule-based*** > ***decision-tree*** ~ ***naïve Bayes*** ($p < .01$; see Figure 2).

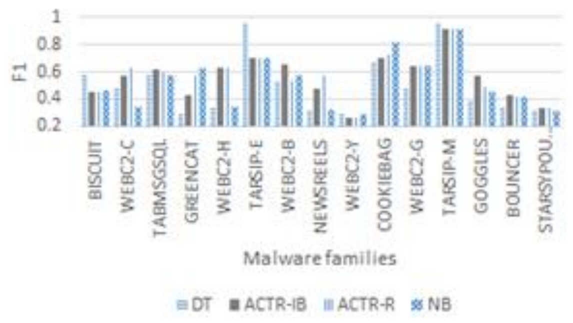

Figure 3. Leave one family out cross-validation.

The results of these analyses indicate that the cognitively-inspired models performed quantitatively better than the baseline machine learning approaches. Also, in the leave-one-out analyses, the iterative nature of the instance-based cognitive model allowed for it to reach near-asymptotic performance (and its best when compared against base-line approaches) after only 40% of the training instances.

We also investigated which features of families may cause the cognitive models to exhibit superior performance over baseline approaches. Figure 3 presents a similarity matrix between malware families. Similarity was computed by generating the Kullback-Leibler Divergence of intents' ground truth between families, averaged over all instances of a family. KLD is a non-symmetric measure of the difference between probability distributions. It is important to note that a family is not always maximally similar to itself: each cell contains the average of the similarities between each pair of malware in that family.

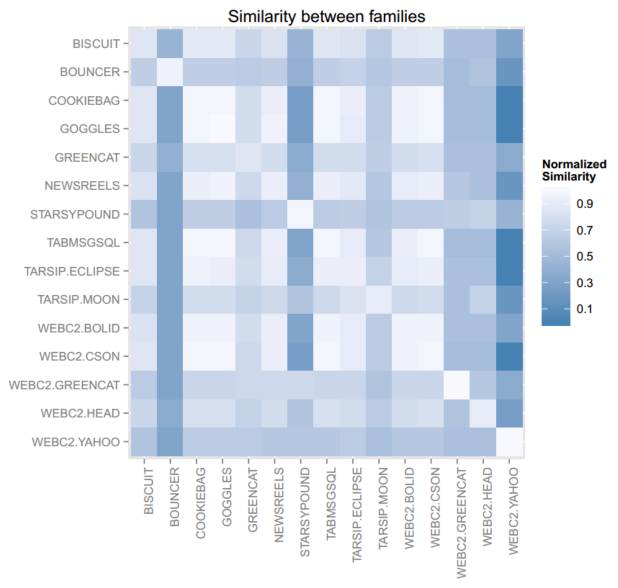

Figure 1. Similarity matrix for all 15 malware families.

By examining the family-wise performance for leave one out cross validation (see Figure 1) we determined that the decision tree has difficulty predicting malware tasks from BISCUIT, WEBC2-CSON, WEBC2-GREENCAT, TABMSGSQL, COOKIEBAG and BOUNCER. As seen in Figure 3, these families have similarity values to each other, leading to substantial confusion within the decision tree. For instance, BISCUIT (top row) is highly similar to 7 other families. It is thus likely that the decision tree will confuse one of the other families for BISCUIT, and potentially vice versa. In essence, the decision tree is a purely symbolic reasoned and will only work accurately if the categories are linearly separate in the dimensions of its representation (i.e., malware features). It fails in these families because there is no logical condition that can be designed to separate them.

The Naïve Bayes model does not fall prey to the same family confusability as the decision tree as it is an inherently statistical algorithm. Instead, from Figure 1 we determined that Naïve Bayes had difficulty predicting malware tasks from WEBC2-YAHOO, NEWSREELS, BOUNCER and STARSPOUND. These families are not as similar (as seen in Figure 3), so these families are in some sense uncorrelated. Naïve Bayes has one core assumption, namely that family attributes are statistically-independent. This assumption is incorrect for the given malware analyses. The families that the Naïve Bayes algorithm has difficulty judging, while not highly correlated, share a relatively large percentage of their attributes with all families when compared against other families from which it can correctly classify.

7. Discussion

We presented two models using cognitively-inspired inference scaled beyond traditional memory limitations in order to rapidly identify malware samples. When compared against baseline machine learning algorithms such as decision trees and Naïve Bayes, our cognitive models were better able to classify malware samples across all 15 sample families. The baseline machine learning algorithms each exhibited difficulties with several families that our cognitive models did not. This is because each is fundamentally limited to the feature set chosen by the modeler, a problem common to machine learning.

By leveraging the temporal dynamics (e.g., recency and frequency) of the cognitively-inspired base-level equation, we were better able to classify highly-similar families than the decision tree. In addition, while the base-level equation can be explained in Bayesian terms and generally reproduces behavior consistent with Bayesian reasoning, in the case of the instance-based model, it is not limited to assumptions of statistical independence. Instead, the instance-based model acts as a universal approximator that does not depend on any kind of linear separation: the more instances that the model perceives, the more accurately it will interpolate between them in a multidimensional space.

It is interesting that the rule-based model performs substantially better than the Naïve Bayes model because the spreading activation equations has the same independence assumption between sources as does the Naïve Bayes algorithm. We argue that the base-level learning component and weighting of the spreading activation equation that compensates, providing the benefits of both a decision-tree and Naïve Bayes classifier without all the drawbacks. For instance, the weight of the spreading activation is effectively forced to be 1 in the Naïve Bayes classifier. By varying this weight, the ACT-R model is capable of determining the relative importance between the prior and posterior likelihoods.

Furthermore, our models were able to reach peak performance when compared against baseline machine learning models after only 40% of stimuli were presented. This is not to argue that cognitively-inspired models are a panacea for malware identification or a clear improvement over machine learning techniques. While our models were able to learn with fewer samples, the processing overhead of situating our models within a cognitive architecture means that they do not necessarily operate faster than machine learning techniques, especially when scaled to larger datasets. That criticism aside, we have elsewhere argued that cognitive models work best when used to supplement human-in-the-loop operations whereby these models take input initially processed by machine learning algorithms (such as deep learning) to do some dimension reduction and provide the cognitive model with loosely structured data from which it may draw inferences (Thomson, Lebiere, & Bennati, 2014). Also, due to the rapid learning across sparse datasets, the ACT-R model has been used to provide top-down feedback to deep learning algorithms (Vinokurov et al., 2011).

Our work substantially differs from malware family classification as we look to infer the tasks a malware was created to perform directly whereas malware family classification is mainly used to help guide an analyst into identifying tasks by first identifying a family. It is noteworthy that we were able to train our classifiers on data of malware of *different* families than the malware we are attempting to classify and were still able to obtain a set of tasks with over 60% recall on the best-performing cognitive models. Further, as a side-effect, we created a probability distribution over malware families as part of an intermediate step – though the ultimate inference of malware tasks is independent of *how* the historical malware families are classified by family. This suggests that at this point machine learning, cognitive models and human experts provide complementary strengths that should be leveraged for the challenging problems facing us in the cyber security domain.

8. Acknowledgement

This work was sponsored by the Intelligence Advanced Research Projects Activity via DOI contract number D10PC20021. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI, or the U.S. Government.

9. References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. 2004. An integrated theory of mind. Psychological Review, 11(4), 1036-1060.

Bayer, U., Comparetti, P.M., Hlauschek, C., Krügel, C., Kirda, E. 2009. Scalable, behavior- based malware clustering. In *NDSS*.

Chase, W. G., & Simon, H. A. 1973. Perception in Chess. *Cognitive Psychology*, 4, 55-61.

Firdausi, I; Lim, C.; Erwin, A; & Nugroho, AS. 2010. Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection. *In Proceedings of Second Annual Conference of Advances in Computing, Control and Telecommunication Technologies*. 201-203.

Gonzalez, C., Lerch, J. F., & Lebiere, C. 2003. Instance-based learning in dynamic decision making. *Cognitive Science, 27*, 591-635.

Jilk, D. J., Lebiere, C., O'Reilly, R. C., & Anderson, J. R. 2008. SAL: An explicitly pluralistic cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 20(3), 197-218.

Kinable, J., Kostakis, O. 2011. Malware classification based on call graph clustering. *J. Comput. Virol*. 7(4), 233–245. DOI 10.1007/s11416-011-0151-y.

Kong, D., & Yan, G. 2013. Discriminant malware distance learning on structural information for automated malware classification. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining,* 1357–1365. ACM, New York, NY, USA. DOI 10.1145/2487575.2488219.

Lebiere, C. 1999. The dynamics of cognition: An ACT-R model of cognitive arithmetic. *Kognitionswissenschaft.* 8 (1), 5-19.

Lebiere, C., Gonzalez, C., & Martin, M. 2007. Instance-based decision making model of repeated binary choice. In *Proceedings of the 8th International Conference on Cognitive Modeling*. Ann Arbor, Michigan, USA.

Lebiere, C., Gonzalez, C., & Warwick, W. 2009. A Comparative Approach to Understanding General Intelligence: Predicting Cognitive Performance in an Open-ended Dynamic Task. In Proceedings of the Second Artificial General Intelligence Conference (AGI-09). Amsterdam-Paris: Atlantis Press.

Lebiere, C., Pirolli, P., Thomson, R., Paik, J., Rutledge-Taylor, M., Staszewski, J., & Anderson, J. R. 2013. A functional model of sensemaking in a neurocognitive architecture. *Computational Intelligence & Neuroscience.*

Lebiere, C, Bothell, D., Morrison, D., Oltramari, A., Martin, M., Romero, O., Thomson, R., & Vinokurov, J. 2015. Strong Cogsci: Guidance from cognitive science on the design of a test of Artificial Intelligence. Proceedings of the *Beyond the Turing Test Workshop, AAAI-2015*.

Li, P., Liu, L., Gao, D., & Reiter, M. K. 2010. On Challenges in Evaluating Malware Clustering. *Proceedings of the 13th International Symposium on Intrusion Detection*, Ottawa, Canada.

Mandiant. 2013. APT1: Exposing One of China's Cyber Espionage Units. Mandiant Corp. URL: http://intelreport.mandiant.com/ retrieved 1/21/2014.

Miller, G. A. 1956. "The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63 (2), 81–97.

Perdisci, P., & ManChon, U. 2012. VAMO: Towards a Fully Automated Malware Clustering Validity Analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference.*

Reitter, D., & Lebiere, C. (2010). Accountable Modeling in ACT-UP, a Scalable, Rapid-Prototyping ACT-R Implementation. In *Proceedings of the 2010 International Conference on Cognitive Modeling*.

Sanner, S., Anderson, J. R., Lebiere, C., & Lovett, M. 2000. Achieving efficient and cognitively plausible learning in backgammon. *In Proceedings of the Seventeenth International Conference on Machine Learning,* 823-830. San Francisco: Morgan Kaufmann.

Sikorski, M., Honig, A. 2012. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, San Francisco, CA, USA.

Taatgen, N., Lebiere, C. & Anderson, J.R. 2006. Modeling paradigms in ACT-R. In Sun, R. (Ed) *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation.* NY, NY: Cambridge Press.

Tamersoy, A., Roundy, K. A., & Horng, D. P. 2014. Guilt By Association: Large Scale Malware Detection by Mining File-Relation Graphs". In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2014.* New York City, NY.

Thomson, R. Lebiere, C., & Bennati, S. 2014. Human, Model, and Machine: A Complementary Approach to Big Data. In Association for Computing Machinery *Proceedings of the IARPA Workshop on Human Centered Big Data Research.* Raleigh, NC. doi:10.1145/2609876.2609883

Thomson, R. & Lebiere, C. 2013. Constraining Bayesian Inference with Cognitive Architectures: An Updated Associative Learning Mechanism in ACT-R. In *Proceedings of the 35th Annual Conference of the Cognitive Science Society*. Berlin, Germany.

Vinokurov, Y., Lebiere, C., Herd, S. A., & O'Reilly, R. C. 2011. A Metacognitive Classifier Using a Hybrid ACT-R/Leabra Architecture. *Lifelong Learning, AAAI Workshops*.

Wallach, D. & Lebiere, C. 2003. Conscious and unconscious knowledge: Mapping to the symbolic and subsymbolic levels of a hybrid architecture. In Jimenez, L. (Ed.*) Attention and Implicit Learning.* Amsterdam, Netherlands: John Benjamin Publishing.

Author Biographies

ROBERT THOMSON is research faculty at Carnegie Mellon University in Pittsburgh, PA and works with the Naval Research Laboratory in Washington, DC. His main research interests are learning theories and the structure of memory, particularly their application to computational cognitive architectures and robotics.

CHRISTIAN LEBIERE is research faculty in the Psychology Department at Carnegie Mellon University. His main research interests are computational cognitive architectures and their applications to psychology, artificial intelligence, human-computer interaction, decision-making, intelligent agents, and cognitive robotics.

PAULO SHAKARIAN Paulo Shakarian is an Assistant Professor at Arizona State University. His research areas include artificial intelligence, social network analysis, and cyber security. He has written two books, including Elsevier's "Introduction to Cyber-Warfare" and his social network analysis software is used by the Chicago Police.

STEFANO BENNATI is a researcher at Carnegie Mellon University in Pittsburgh, PA.

ERIC NUNES is a Ph.D. student at Arizona State University in Tempe, AZ.