Gaussian Mixture Models for Dynamic Malware Clustering

Alexander M. Interrante-Grant, *Student*, David Kaeli, *Professor*
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA

***Abstract* —**

As the number of unique malware samples grows at a rapidly increasing rate, analysts are having trouble tracking the evolution of existing malware and identifying new malware in an ever-changing threat landscape. It can take hours for a malware analyst to evaluate a single sample, so they are increasingly turning to methods of fast, automated malware analysis to identify trends across and attributes of newly observed malware. One goal is to identify the author of a new piece of malware. Various approaches have been proposed, applying machine learning to various concise program representations. Because ground truth labels for malware samples are notoriously difficult to find, most machine learning approaches rely on unsupervised learning (i.e. clustering) methods.

In this paper, a number of recently proposed clustering approaches using co-occurrence matrices of system calls are evaluated. In addition to applying previously-proposed clustering algorithms to this program representation, this work applies Gaussian mixture model (GMM) clustering – an approach that had not been evaluated by the research community for dynamic malware clustering. Our results show that GMM clustering outperforms other clustering approaches, achieving a Fowlkes-Mallows score two times better than the state-of-the-art on a dataset of real-world malware curated from the VirusShare malware corpus.

I. INTRODUCTION

Large-scale malware analysis continues to be increasingly challenging, given that the number of unique malware signatures grows exponentially. As a result of this growth, manual malware analysis is becoming infeasible, requiring an unbounded amount of an analyst's time. Instead, researchers have turned to automated analysis methods to characterize patterns in large corpora of malware to reduce analysis time. These automated analysis methods have been shown in previous work to be useful in identifying similarities and trends among malware samples that cannot feasibly be identified at by manual analysis. These insights can help to more quickly identify and mitigate newly emerging threats and potentially lead to attribution of new malware to the particular actors involved in creating it.

This project evaluates a number of recently published clustering approaches on a dynamic program representation derived from co-occurrence matrices of system calls which is discussed in more detail in later sections. Clustering approaches include k-means and agglomerative clustering using Jaccard distance, Euclidean distance, and hamming distance metrics as well as Gaussian Mixture Model (GMM) clustering. These approaches are evaluated on a corpus of malware from VirusShare, which are labeled using majority voting of various commercial antivirus solutions. This provides a side-by-side comparison of automated malware analysis methods, assessing their relative merits and detriments in assisting malware analysts on a single corpus of real-world malware.

II. DYNAMIC PROGRAM REPRESENTATION

A. Background and Previous Work

Behavior-based, dynamic program analysis has become a growing need in the malware research and program analysis communities. Dynamic program analysis is a comparatively rich data source when contrasted with static program analysis. Dynamic program analysis necessarily reveals program artifacts (files modified, system configuration changes, etc.) that may be obfuscated (as is often the case with malware) or difficult to discern with conventional static analysis methods. However, because of the scale and quantity of dynamic program artifacts, concise dynamic program representations and automated analysis techniques are required to efficiently compare and contrast different programs.

A number of dynamic program representations have been described by prior work – some of these approaches have shown promise in condensing dynamic program representations to a more concise format, while losing only minimal semantic meaning. Jang et al. proposed a method of hashing program features into a binary vector on which traditional mathematical similarity measures, like Jaccard distance, can be applied [1]. Shu et al. proposed a matrix representation of call (function call, syscall, or otherwise) ordering and, again, used simple mathematical matrix comparison operations to discern similarity [2]. Fredrikson et al. proposed an efficient graph-based representation of dynamic malware behavior [3]. Rieck et al. used a representation which they name the Malware Instruction Set (MIST) based on a computer instruction set [4].

B. Implementation

This project focuses on a representation of dynamic program behavior based on co-occurrence matrices of system calls introduced by Shu et al. [2]. This representation was used to

determine a measure of similarity between two malware samples behavior, expanding on Shu et al.'s original work which used this representation purely for the identification of malware from benign applications. Further, this program representation can be trivially converted to a set of features for a machine learning algorithms, and was used as input for GMM clustering.

Shu et al.'s program representation consists of two matrix primitives which can be computed from a trace of "calls". A "call", here could be at any layer of abstraction and could represent anything from specific functions called by the program, to specific operating system API functions, to system calls. Shu et al. define a co-occurrence matrix as an $m \times m$ binary matrix O, where

$$o_{i,j} = True \text{ if call } i \text{ occurred before call } j, \text{ else } False$$

Further, they then define an occurrence frequency matrix as an $m \times m$ matrix F where

$$f_{i,j} = \text{count(occurrences of call } i \text{ before call } j)$$

[2].

As an example, assume a program makes the following series of Windows system calls (representing a common file read operation):

NtOpenFile
NtSetInformationFile
NtReadFile
NtReadFile
NtReadFile
NtWriteFile
NtCloseFile

This program can be represented as the co-occurrence matrix depicted in Figure 1 or the occurrence frequency matrix shown in Figure 2.

	NtOpenFile	NtSetInformationFile	NtReadFile	NtWriteFile	NtCloseFile	⋱
NtOpenFile	0	1	0	0	0	
NtSetInformationFile	0	0	1	0	0	
NtReadFile	0	0	1	1	0	⋮
NtWriteFile	0	0	0	0	1	
NtCloseFile	0	0	0	0	0	
...			...			

Figure 1. Co-occurrence matrix for the given trace of system calls. Note, as an example, that the cell for NtOpenFile:NtSetInformationFile is a one because a call to NtOpenFile occurs before a call to NtSetInformationFile. Similarly, the cell for NtOpenFile:NtOpenFile is a zero because there does not exist a call to NtOpenFile immediately followed by another call to NtOpenFile in the call trace.

Using these representations, a trace of system calls can be converted into a matrix. Then, simple mathematical operations can be performed directly on the matrix for clustering. This paper used system calls as the atomic unit of co-occurrence and occurrence frequency matrices. Since system calls are the fundamental method by which unprivileged user-mode applications access critical machine resources through the kernel, a trace of system calls will necessarily contain all of the potentially security-related behavior that a program accomplishes. Additionally, system calls can be easily traced with minimal overhead for a running process.

	NtOpenFile	NtSetInformationFile	NtReadFile	NtWriteFile	NtCloseFile	⋱
NtOpenFile	0	1	0	0	0	
NtSetInformationFile	0	0	1	0	0	
NtReadFile	0	0	2	1	0	⋮
NtWriteFile	0	0	0	0	1	
NtCloseFile	0	0	0	0	0	
...			...			

Figure 2. Occurrence frequency matrix for a given trace of system calls. Note that the cell for NtReadFile:NtReadFile is a two in this case since a call to NtReadFile is immediately followed by another call to NtReadFile in the call trace.

III. Clustering Methods

A. Background and Previous Work

Machine learning concepts have been applied with some success to the problem of dynamic program similarity. Rieck et al. clustered similar malware and classified unknown malware by embedding n-grams of MIST instructions in a high-dimensional vector space [4]. In addition to simple mathematical analyses, Shu et al. applied a one-class support vector machine to their call frequency matrix program model to detect anomalous malware behaviors [2].

Although not strictly a machine learning approach, other projects have utilized traditional distance metrics in determining malware similarity. Jaccard distance, a measure of the similarity of two sets as the size of the intersection of those two sets divided by the size of their union, is very widely used in the malware analysis research community to determine similarity between malware samples [1, 5].

Shu et al. attempted to solve a classification problem in their work by using a Support Vector Machine (SVM) classifier. Instead, this work attempts to cluster malware by behavioral similarity. While malware classification can be useful for identifying malware and differentiating malware execution from expected system behavior, clustering can be useful for identifying commonalities between particular samples of malware and can be used for identification and attribution of new malware. To test the effectiveness and generalizability of

this program representation, this project examined three clustering algorithms along with three distance metrics to see which performed the best. Selected algorithms and metrics are discussed in the next section.

B. Implementation

The three clustering methods evaluated using the previously mentioned program representation as input data were:

K-Means Clustering clusters samples by alternating between assigning samples to clusters by computing the minimum distance from the cluster centroid (or mean) and recalculating cluster centroids based on the new clustering until convergence [6].

Agglomerative Clustering is a form of hierarchical clustering where clusters are recursively merged based on minimizing some distance metric until the desired number of clusters is achieved [7].

Gaussian Mixture Model (GMM) Clustering clusters samples by estimating the parameters of the distributions of each of the classes, assuming those distributions are normal [8].

For all of the above clustering methods, the number of clusters was given as the number of unique malware families in the test corpus.

While both k-means and agglomerative clustering have been widely used in the malware analysis literature, GMM clustering has not been applied to this type of data and, to the best of our knowledge, is a new contribution from this paper. With k-means and agglomerative clustering, the following distance metrics were evaluated:

Jaccard Distance measures the distance between any two sets and is the inverse of jaccard similarity which is defined as the cardinality of the sets' intersection divided by the cardinality of their union [9].

Hamming Distance measures the distance between any two ordered sets as the number of positions in each that differ [10].

Euclidian Distance measures the distance between two n-dimensional sets as square root of the sum of squared distances [11].

A table summarizing all selected algorithms and corresponding distance metrics is provided in Table 1. These algorithms were evaluated using Scikit Learn's implementations [12].

Table 1. Selected clustering algorithms and corresponding distance metrics.

Algorithm	Distance Metric
K-Means	Jaccard
	Hamming
	Euclidian
Agglomerative	Jaccard
	Hamming
	Euclidian
GMM	N/A

To evaluate each of the distance-based clustering algorithms (k-means and agglomerative clustering), the distances between co-occurrence matrices were taken to avoid the problem of normalizing occurrence frequency matrices across various samples. GMM, however, can be performed directly on the occurrence frequency matrix of a sample by treating each pair of system calls as a feature, effectively collapsing the occurrence frequency matrix into a vector of features. Feature selection is necessary here to reduce the requisite computational resources, so a simple minimum variance threshold was used.

IV. Evaluation and Results

A. Dataset

To evaluate the effectiveness of each algorithm and distance metric, a large corpus of malware was required. Most of the papers referred to previously did not specify what dataset they used for testing or referenced a private corpus of malware maintained directly by them. Given that no publicly available and well-labeled corpus of malware commonly used for research projects like this exists, a corpus of samples from some of the most recently uploaded Windows malware samples on VirusShare [13] was cultivated as a part of this project.

After removing samples that would not run due to compatibility issues with our target analysis environment, the remaining 159 samples were actuated in a sandbox environment. System call traces were gathered using DrStrace [14] and converted into co-occurrence and occurrence frequency matrix representations. Ground truth labels were generated by a majority voting scheme of popular commercial antivirus using VirusTotal [15], resulting in a total of ten unique malware families. The antivirus labels of this dataset and the relative representation of each are depicted in Table 2.

Table 2. Malware test corpus composition by antivirus label.

Antivirus Label	Count
Backdoor.Turkojan.DQ	47
Gen:Variant.FakeAV.21	22
Trojan.Jevafus.A	21
Trojan.Crypt.Delf.G	14
Trojan.Spy.BZub.NHN	14
Trojan.SMSHoax.X	13
Backdoor.PcClient.TEV	9
Backdoor.Optix.Pro.1.3.Dam.2	7
Gen:Variant.Graftor.18403	6
Gen:Variant.Graftor.Elzob.14614	6
Total	**159**

B. Results

Clustering was performed on the malware dataset depicted in Table 2 using the previously mentioned algorithms and distance metrics, contained in Table 1. Relative performance was calculated in terms of the following three metrics:

Adjusted Rand Index (ARI) measures similarity between two clusterings by counting the pairs of samples that are assigned to the same or different clusters, adjusted for chance [16]. Ranges from -1 to 1, higher values indicate greater similarity.

Adjusted Mutual Information Index (AMI) measures the similarity between two clusterings as the relative entropy of each clustering, adjusted for chance [17]. Ranges from -1 to 1, higher values indicate greater similarity.

Fowlkes-Mallows Index (FMI) measures the similarity between two clusterings as the geometric mean of precision and recall [18]. Ranges from 0 to 1, higher values indicate greater similarity.

Table 3. Clustering performance results.

Algorithm	Distance Metric	ARI	AMI	FMI
K-Means	Jaccard	-0.0037	-.0085	0.1236
	Hamming	0.0107	-0.0251	0.1724
	Euclidian	-0.0031	-0.0229	0.1527
Agglomerative	Jaccard	0.0007	-0.0240	0.1453
	Hamming	0.0107	-0.0251	0.1724
	Euclidian	-0.0033	-0.0222	0.1479
GMM	N/A	0.1953	0.4018	0.4054

The results are depicted in Table 3 along with Figure 3, Figure 4, and Figure 5.

C. Analysis

These results indicate that the clustering algorithms and distance metrics widely used in prior work for program similarity in malware analysis do not perform very well when tested on this VirusShare dataset. In contrast, a clustering technique which has not yet been widely used in this context, GMM clustering, performs significantly better in terms of all three metrics.

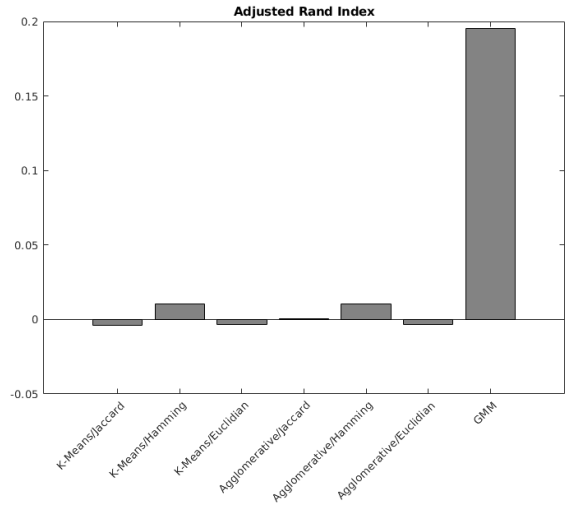

Figure 3. ARI performance of clustering algorithms. GMM clustering significantly outperforms all other techniques when considering the number of samples assigned to the same or different clusters between the produced clustering and the true labels, adjusted for chance.

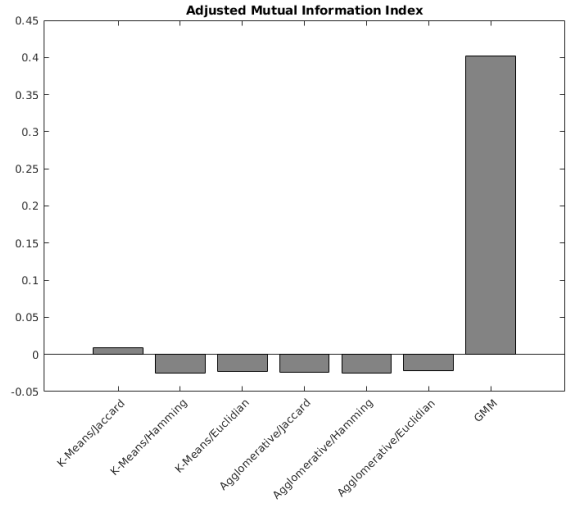

Figure 4. AMI performance of clustering algorithms. GMM clustering again significantly outperforms all other techniques when considering the relative entropy between the produced clustering and the true labels, adjusted for chance.

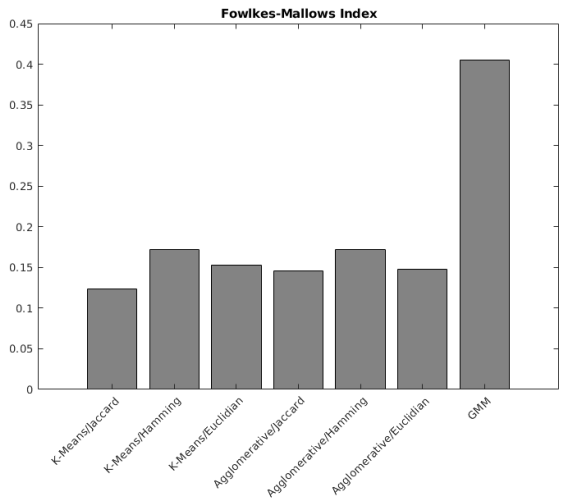

Figure 5. FMI performance of clustering algorithms. GMM clustering provides a more than two times performance increase when considering the precision and recall of the produced clustering as compared to the true labels.

It is worth noting that these results are constrained to this particular program representation (co-occurrence and occurrence frequency matrices of system calls) and when the mathematics behind each clustering approach are taken into consideration, this result is exactly as expected. Because of the nature of how system calls are used during the execution of a particular program, there are a number of system calls whose occurrence is likely very highly correlated with others. For example, if a program calls the NtOpenFile system call, it is very likely that it will then call NtReadFile or NtWriteFile immediately afterward, to operate on the file that they just opened. GMM clustering is well equipped to handle this covariance since it attempts to estimate the probability densities of each parameter directly. K-Means and agglomerative clustering, on the other hand, weigh each feature equally which is likely detrimental to clustering performance.

In terms of required computational resources, both k-means and agglomerative clustering approaches were significantly more performant than GMM clustering given the incredibly high dimensionality of the data. Across all actuated malware samples, 408 unique system calls were observed, resulting in 408^2 total features. In order to keep the dimensionality low, a simple minimum variance feature selection was used to decrease the dimensionality of the data in order for GMM clustering to be feasible. This relatively simple approach to feature selection could likely be improved by using a more complex feature selection algorithm (eg. principle component analysis). At a higher level of abstraction (e.g., Windows API calls, MIST instructions, or some other semantic interpretation of program behavior), distance-based clustering approaches may actually be more feasible as the dimensionality increases and the covariance between features decreases.

V. Limitations

A. Dataset Labels

Antivirus labels are notoriously lacking in substantive information about the behavior of individual malware samples and are often arbitrary, selected at the whim of the analyst dissecting the sample. In many cases the analyst assigns the malware to a class simply based on the suspected author of a piece of malware rather than its actual behavior. Previous research has evaluated the performance of antivirus labels in comparing malware families and found them to be significantly lacking [19, 20, 21]. Unfortunately, there is no better source of ground truth labels for real-world malware. While some well-labeled synthetic malware datasets do exist, many are not publicly available and the results using such a dataset are unrealistic. Gathering a corpus of malware with high-quality behavior-based labels remains a difficult problem. To some extent, this may account for the mediocre performance of most clustering approaches in this domain.

B. Cluster Poisoning

One major problem with all machine learning approaches to security problems is their susceptibility to an adversary with full knowledge of algorithm and training state. By inserting specifically designed behavioral chaff in the code of a piece of malware, the attacker can alter the clustering and classification decision, potentially evading detection or attribution entirely and reducing the effectiveness of malware analysts tools. This approach has been demonstrated by Biggio et al. [22].

VI. Conclusion

This work evaluated the effectiveness of Gaussian Mixture Models (GMMs) for behavioral malware clustering using occurrence frequency matrices of system calls. The results have shown that GMM clustering outperforms distance-based clustering algorithms like k-means and agglomerative clustering on a dataset of recent, real-world malware using antivirus labels as ground truth. The relative performance of these clustering algorithms shows that GMM clustering significantly outperforms widely-used distance-based clustering. This assessment of existing malware clustering approaches and proposal of GMM clustering as a novel contribution by this paper shows that automated analysis methods can be very effective in identifying trends and correlations in and between various families of malware. These types of automated analysis methods help malware analysts stay ahead of the ever-growing body of malware active on the internet and ultimately lead to a more secure internet.

Acknowledgements

Special thanks to Ryan Whelan for contributing to the initial research question and providing technical insight and support throughout this work.

Author Biographies

Alexander M. Interrante-Grant is a recent Northeastern EECE alumnus (class of 2018) and is presently at MIT Lincoln Laboratory, researching automated static and dynamic malware analysis. During his time at Northeastern, he participated in various cybersecurity-related research projects at the university and served as the captain of the Northeastern Cyber Defense Team. This work began through a directed study opportunity and was completed as a final project for Machine Learning (EECE 5644) during his final undergraduate semester at

Northeastern. The referenced work was performed in collaboration with MIT Lincoln Laboratory and the Northeastern TeSCASE group.

alexander.interrante-grant@ll.mit.edu

David Kaeli received a BS and PhD in Electrical Engineering from Rutgers University, and an MS in Computer Engineering from Syracuse University. He is the Associate Dean of Undergraduate Programs in the College of Engineering and a Full Processor on the ECE faculty at Northeastern University, Boston, MA. He is the director of the Northeastern University Computer Architecture Research Laboratory (NUCAR). Prior to joining Northeastern in 1993, Kaeli spent 12 years at IBM, the last 7 at T.J. Watson Research Center, Yorktown Heights, NY.

Dr. Kaeli has published over 300 critically reviewed publications, 7 books, and 11 patents. His research spans a range of areas including microarchitecture to back-end compilers and database systems. His current research topics include hardware security, graphics processors, virtualization, heterogeneous computing and multi-layer reliability. He serves as an Associate Editor of the IEEE Transactions on Parallel and Distributed Systems, ACM Transactions on Architecture and Code Optimization, and the Journal of Parallel and Distributed Computing. Dr. Kaeli an IEEE Fellow and a Distinguished Scientist of the ACM.

kaeli@ece.neu.edu

REFERENCES

[1] Jiyong Jang, David Brumley, and Shobha Venkataraman. "BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis." *Proceedings of the 19th ACM Conference on Computer and Communications Security.*

[2] Xiaokui Shu, Danfeng Yao, and Naren Ramakrishnan. "Unearthing Stealthy Program Attacks Buried in Extremely Long Execution Paths." *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.*

[3] Matt Fredrikson, et al. "Synthesizing Near-Optimal Malware Specification from Suspicious Behaviors." *Proceedings of the 2010 IEEE Symposium on Security and Privacy*.

[4] Konrad Rieck, et al. "Automatic Analysis of Malware Behavior Using Machine Learning." *Journal of Computer Security*. vol. 19, no. 4, Dec. 2011.

[5] Dhilung Kirat and Giovanni Vigna. "MalGene: Automatic Extraction of Malware Analysis Evasion Signature." *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.*

[6] J MacQueen. "Some methods for classification and analysis of multivariate observations." *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol 1, 1967.

[7] Lior Rokach, and Oded Maimon. "Clustering methods." *Data mining and Knowledge Discovery Handbook.* Springer US, 2005.

[8] A P Dempster, N M Laird, and D B Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". *Journal of the Royal Statistical Society*. vol. 39, no. 1, 1977.

[9] Elena Deza and Michel Marie Deza. "Encyclopedia of Distances." Springer US, 2009. p. 299.

[10] R W Hamming. "Error detecting and error correcting codes." *The Bell System Technical Journal*. vol. 29, no. 2, Apr. 1950.

[11] Elena Deza and Michel Marie Deza. "Encyclopedia of Distances." Springer US, 2009. p. 94.

[12] scikit-learn, "scikit-learn: Machine Learning in Python", Oct. 2017

[13] virusshare.com, "Virusshare.com", 2017. [Online]. Available: http://virusshare.com/. [Accessed: Dec- 2017].

[14] Dr. Memory, "Dr. Memory: System Call Tracer for Windows." Aug. 2016

[15] virustotal.com, "Virustotal.comhttps://www.virustotal.com/

[16] Lawrence Hubert and Phipps Arabie. "Comparing Partitions." *Journal of Classification*. vol. 2, no. 1, Dec. 1985.

[17] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance." *Jounral of Machine Learning Research.* vol. 11, Oct. 2010.

[18] E. B. Fowlkes and C. L Mallows. "A Method for Comparing Two Hierarchical Clusterings." *Journal of the American Statistical Association*. vol. 78, no. 383, Sep. 1983.

[19] Domhnall Carlin, et al. "The Effects of Traditional Anti-Virus Labels on Malware Detection using Dynamic Runtime Opcodes." *IEEE Access*, vol. 5, Sep. 2017

[20] Alex Kantchelian, et al. "Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels." *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security.*

[21] Aziz Mohaisen and Omar Alrawi. "AV-Meter: An Evaluation of Antivirus Scans and Labels." *Proceedings of the 2014 International Conference on Detection of Intrusions and Malware and Vulnerability Assessment.*

[22] Battista Biggio, et al. "Poisoning Behavioral Malware Clustering." *Proceedings of the 2014 Workshop on Artificial Intelligence and Security.*