The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Artificial Intelligence for Cyber Security: Technical Report WS-16-03

Discovering Human and Machine Readable Descriptions of Malware Families

Blake Anderson
blaander@cisco.com
Cisco Systems, Inc.

David McGrew
mcgrew@cisco.com
Cisco Systems, Inc.

Subharthi Paul
subharpa@cisco.com
Cisco Systems, Inc.

Abstract

While an immense amount of work has gone into novel clustering algorithms, little work has focused on developing compact, domain-specific explanations for the results of the clustering algorithms. Attaching semantic meaning to a cluster has numerous benefits, including the ability for such a description to be both human and machine readable. In this paper, we assume that the clusters are given to us, and find the minimal set of features that can differentiate one cluster from the remaining set of samples. We formulate this problem as an integer linear program. By using samples not belonging to the cluster in the optimization formulation, the resulting description will be minimal and contain no false positives.

The efficacy of this method is demonstrated on simulation data and real-world malware data run in a sandbox that collects behavioral characteristics. In the case of malware, once it has been clustered, it would have been sent to a reverse engineer who would have been tasked with creating the actual meaning of the clustering results and disseminating this information through signatures or indicators of compromise. This is a time-consuming process that can take hours to weeks depending on the complexity of the malware family. The methods presented in this paper automatically generate optimal signatures, which can then be quickly propagated to help contain the spread of a malware family.

Introduction

With more domains producing a seemingly unending amount of data, machine learning techniques to categorize and make sense of the data is of paramount importance. One simple machine learning concept, clustering, is heavily used in domains ranging from biology to cybersecurity (Bayer et al. 2009; Fielding 2007; Jain, Murty, and Flynn 1999).

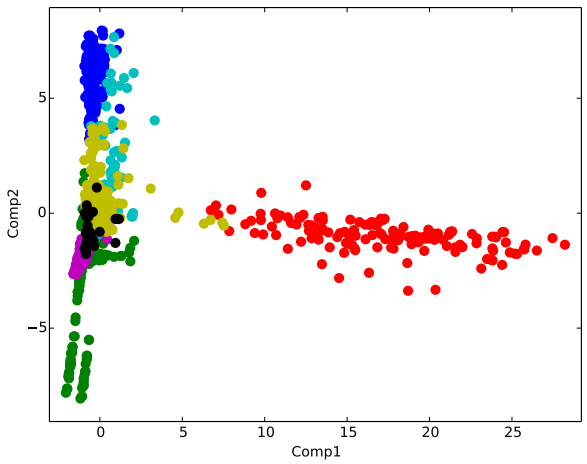
The goal of this paper is to give semantic meaning to the results of a clustering algorithm. We aim to answer the question, *What minimal set of features can differentiate one cluster from the remaining samples in the dataset?* For instance, we want to be able to say that a cluster is unique because it contains feature X, contains feature Y, and does not contain feature Z.

We solve this problem as follows. Assuming binary features, the centroid is computed for the cluster that is being analyzed. This is done by simply taking the average over all feature vectors for the given cluster. After this step, the centroid is thresholded to obtain a binary centroid vector. Then, an $n \times m$ matrix, $\mathbf{A}$, is created where n is the number of samples not in the cluster and m is the number of features. Each row of $\mathbf{A}$ is computed by taking the absolute value of the difference between the centroid and a sample not belonging to the cluster. By using samples not belonging to the cluster in the optimization formulation, the resulting description will be minimal and contain no false positives.

With $\mathbf{A}$ defined, this problem can be cast a standard integer linear program (Schrijver 1998) and solved. The solution will be a vector of 1's and 0's describing whether a specific feature is important to the description of the cluster or not. In this most general form, the features that make a cluster unique do not necessarily have to be positive features for that cluster, but it could be the case that lacking a feature(s) is what makes the cluster unique.

Although this method can be used for any clustered data, this paper is grounded by introducing a specific use case: malware signature generation. A malware signature is a set of simple rules about observable parameters such as files, registry entries, and network communication, that is characteristic of a particular malware type. A common theme in malware incident response is having the ability to quickly propagate threat intelligence to other sites within an organization or to sister organizations so that they can prevent or more quickly recover from an attack. This is typically done through the use of YARA rules (yar) or STIX indicators (sti). These methods give a standard format to report artifacts of a malware infection such as the strings contained within a malicious executable or registry modifications made by a malicious executable. Unfortunately, the generation of these indicators is a time consuming process and can take days to find a suitable signature with a low false positive rate that can properly capture the information of an attack. Furthermore, with the advent of polymorphic malware (Newsome, Karp, and Song 2005), developing a signature from a single sample is not a robust solution.

In the malware domain, clustering is a natural approach because malware instances can be grouped into malware families (Rieck et al. 2008). A malware family is a group of malicious programs that all contain some amount of similar code and/or functionality. While many authors have in-



Cluster	Description
Cluster 1	Cluster Has IP address *.*.*.*
Cluster 2	Cluster Has Registry Access * Cluster Has Registry Access * Cluster Does Not Have Registry Access *
Cluster 3	Cluster Has File Access * Cluster Does Not Have File Access * Cluster Does Not Have File Access *
Cluster 4	Cluster Does Not Have IP address *.*.*.* Cluster Does Not Have File Access * Cluster Does Not Have Registry Access *
Cluster 5	Cluster Has Registry Access * Cluster Has Registry Access *
Cluster 6	Cluster Has Registry Access *
Cluster 7	Cluster Does Not Have IP address *.*.*.* Cluster Does Not Have File Access *

Figure 1: This data consists of 3,623 instances of malware with 7,305 features (IP addresses, registry modifications, file system modifications) and is clustered by k-means with $k = 7$. On the left, the first two principle components of the clustered data is shown. On the right, the descriptions computed by the methods of this paper.

vestigated which feature representations and clustering algorithms are optimal for the malware domain (Bayer et al. 2009; Rieck et al. 2008; Anderson, Storlie, and Lane 2012), few have looked at generating actionable intelligence once the clusters are known.

The methods presented in this paper will address the problem of automatically finding the features of a malware infection that can be used to create high fidelity, low false positive YARA rules or STIX indicators to aid in propagating threat intelligence. We will show results using real-world data obtained from a sandbox that collects behavioral characteristics of the malware samples using dynamic analysis. We use a combination of k-means and anti-virus labels to perform the clustering. Alternatively, any clustering techniques previously examined could be substituted. Our methods will be able to describe malware families in ways that a malware analyst can actually understand, such as "malware family A connects to ip range $X.X.*.*$, modifies registry value Y, creates file Z, and does not modify file W".

Related Work

In the machine learning literature, explaining clusters has typically been done by projecting the data instances to a low dimensional feature space where it can be easily visualized (Kriegel, Kröger, and Zimek 2009), or by measuring the clusters with validity indices that help to score how well the clustering algorithm performed (Halkidi, Batistakis, and Vazirgiannis 2002). While the machine learning methods to validate the clusters' integrity, either visually or numerically, would be useful to perform to ensure a reasonable clustering, they do not provide a human and machine readable description of the cluster. For example, in Figure 1, the graph has visualized high-dimensional data by using the first two principle components. While this does give useful information with respect to the separability of the dataset, it does not give a compact, human readable description of the clusters. On the other hand, the table in Figure 1 displays this information clearly.

To our knowledge, no other attempt has been made to explain clusters using a minimal set of features that a cluster contains or does not contain. There have been approaches that are aimed at finding human-readable descriptions for important data mining problems. For instance, in (Wagstaff et al. 2013), the authors are solving the class discovery problem. They build a model with all known classes, and use per-feature residuals to rank the features that were most important in determining whether a data instance is novel relative to their current model. In (Strumbelj and Kononenko 2010), the authors construct per-feature explanations for why an instance was classified to a certain class.

There has been a class of methods to create signatures for families of polymorphic malware and their corresponding payloads (Newsome, Karp, and Song 2005; Wang, Cretu, and Stolfo 2006). These methods work by finding similar features or subsequences in all instances within a cluster. The two drawbacks to the these signature generating approaches are 1) they do not find the smallest set of features that uniquely define a cluster, and 2) they do not use the non-cluster instances in their signature definition. The signatures found can often be quite large, thus making it difficult for a human to reason about why a cluster is unique. By not comparing against instances belonging to other families, it is difficult to restrict the size of this set to just the features that are important. In this paper, the minimal set of features are found that uniquely define a cluster with respect to every other instance not in that cluster. This makes it both easier for a human to understand and guarantees that there will be no false positives among the members of the other

clusters. This is accomplished by an integer linear program using the members of the other clusters as part of its optimization formulation.

Methods

In this section, a novel method is proposed to find the minimal set of features that gives the semantic description for a given cluster. The data representation for a given cluster is illustrated. Then, the problem is posed as an integer linear program which can be solved using existing software. Finally, some practical details of the techniques and software used are considered.

Data Description

To begin, the centroid is computed for the cluster that is being analyzed. This is done by simply taking the average over all feature vectors for the given cluster. For instance, assume there are three samples in a cluster with the following feature vectors:

$$\begin{bmatrix} [0, & 0, & 1, & 0] \\ [1, & 0, & 1, & 1] \\ [1, & 0, & 1, & 0] \end{bmatrix}$$

Then the centroid for the cluster would be $[.66, 0.0, 1.0, .33]$. After this step, the centroid is thresholded to obtain a vector of just 1's and 0's. The threshold value helps to eliminate features that do not contain enough support within a cluster to be used. For a threshold value of t, any feature in the centroid, f_i, that is greater than t will be set to one, any feature less than $(1.0 - t)$ will be set to 0, and all other features in the centroid will be discarded:

$$f_i = \begin{cases} 1 & \text{if } f_i \geq t \\ 0 & \text{if } f_i \leq 1.0 - t \\ \text{discard} & \text{otherwise} \end{cases} \quad (1)$$

For example, with a threshold value of 0.9, $[.66, 0.0, 1.0, .33]$ would be converted to $[-, 0, 1, -]$ and only the second and third features would not be discarded.

ILP Formulation

The problem of finding the minimal set of distinguishing features can be cast as a standard integer linear program:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ \text{and} \quad & \mathbf{x} \in \{0, 1\} \end{aligned} \quad (2)$$

In this formulation, $\mathbf{b}$ is a length n vector of 1's where n is the number of samples not in the current cluster. $\mathbf{A}$ is an $n \times m$ matrix where m is the number of features in the centroid of the current cluster. Each row of $\mathbf{A}$ is computed by taking the absolute value of the difference between the centroid and a sample not belonging to the cluster. For example, assuming the previous centroid, $[-, 0, 1, -]$, and samples:

$$\begin{bmatrix} [1, & 1, & 1, & 1] \\ [0, & 0, & 0, & 1] \\ [0, & 1, & 1, & 0] \\ [1, & 1, & 1, & 0] \\ [0, & 0, & 0, & 1] \end{bmatrix}$$

The following $\mathbf{A}$ matrix would be produced:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Finally, $\mathbf{x}$ is the vector representing the different features in the centroid. If $x_i = 1$, this would be interpreted as the i'th feature having discriminatory power with respect to the current cluster. S, where $\forall i \in S, x_i = 1$, is the minimal set of features that uniquely defines the cluster. In this most general form, the features that make a cluster unique do not necessarily have to be positive features for that cluster, but it could be the case that lacking a feature(s) is what makes the cluster unique. In the previous example, the $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ constraint is equivalent to the following system of equations:

$$\begin{array}{rcrcl} 1.0 \cdot x_1 & + & 0.0 \cdot x_2 & \geq & 1.0 \\ 0.0 \cdot x_1 & + & 1.0 \cdot x_2 & \geq & 1.0 \\ 1.0 \cdot x_1 & + & 0.0 \cdot x_2 & \geq & 1.0 \\ 1.0 \cdot x_1 & + & 0.0 \cdot x_2 & \geq & 1.0 \\ 0.0 \cdot x_1 & + & 1.0 \cdot x_2 & \geq & 1.0 \end{array} \quad (3)$$

which is solved when both x_1 and x_2 are set to 1. While this is a trivial example, it does demonstrate the basic reasoning for making use of an integer linear program.

Practical Details

While using a linear program instead of an integer linear program would seem advantageous because solving an integer linear program is NP-complete, the solution to the linear program doesn't make sense in the context of deriving simple, human readable descriptions of the data. The meaning of a cluster having feature, x_i, is much more interpretable than a cluster having $2/3$ of feature x_i. Solving a linear program and thresholding the results was attempted, but led to poor results and was discarded.

There are many excellent optimization libraries to choose from, but for this work we used python and the CVXOPT optimization library (Andersen, Dahl, and Vandenberghe) with the GLPK (glp) bindings. GLPK uses a branch-and-cut algorithm (Padberg and Rinaldi 1991) for its integer linear programming optimization. These libraries proved to be fast and reliable.

The threshold in Equation 1 helps the integer linear program avoid infeasible solutions by relaxing the definition for what it means for a feature to be important to a cluster. All of the experiments in this paper used $t = 0.9$. This value could be automatically tuned with the presence of a holdout set to check for false positives and by checking for infeasible solutions.

As an aside, although this method was designed to work on clustered data, if the cluster centroid is replaced by a single instance, the algorithm would perform the same and the output would be a human and machine readable description of what makes a specific instance unique. In other words, it could be used to generate a signature for a specific sample.

Registry access: shellnoroam\muicache
Registry access: explorer\shell folders
Registry access: internet settings\zonemap
No file access: win.ini

(a) ILP

Registry access: explorer\user shell folders
Registry access: shellnoroam\muicache
Registry access: cryptography\rng
Registry access: explorer\mountpoints2
Registry access: explorer\shell folders
Registry access: internet settings\zonemap
File access: wkssvc
File access: lsarpc
File access: r00000000000b.clb
File access: desktop.ini

(b) Token

Figure 2: Two descriptions of Cluster 14 in the Zeus dataset. (a) is the description that resulted from the integer linear program, and (b) is the description from the token method with a threshold.

Malware Family	Number of Instances
Zeus	3620
Cybergate	3430
Shylock	433
Rovnix	1182

Table 1: The number of instances of each malicious family.

Results

In this section, the results of our method are demonstrated on two large malware datasets. Timing results for our method are demonstrated on a synthetic dataset. All results were obtained within a VM running Ubuntu 14.04 on top of a laptop with a Core i5-4300U CPU @ 1.9 GHz and 8 GB of memory.

The purpose of this work is not strictly to generate signatures, but rather to compactly describe clusters to an analyst. But, for the sake of completeness, the token enumeration method of (Newsome, Karp, and Song 2005) is compared against. This method is straightforward, and looks for common features among all instances within a cluster. To be fair, two versions of this method are used: one with the threshold set to 1.0 (all instances must have the feature) and 0.9.

Data

Table 1 lists the families used and the number of instances belonging to each family. The first set of experiments clusters the data within Zeus and generates human readable descriptions of the clusters. In the second set of experiments, all four families are used.

Zeus is a toolkit that is intended to facilitate botnets, and its primary purpose is to exfiltrate data. Cybergate is a malicious program that allows a threat actor to have remote access privileges. Shylock is a program whose primary purpose is to steal banking information. Finally, Rovnix is a program that hides information in the volume boot record with the intent to maintain persistence and exfiltrate data.

All of the malware samples were run in a virtual machine, which collected many behavioral characteristics of the sample. The sandbox runs only lasted 5 minutes, so that the observed behavior corresponds to the initial infection. Of the features collected, the registry modifications, file system modifications, and the IP addresses communicated with are used as input into the clustering algorithm and the methods presented in this paper. For just Zeus, there are 7,305 features. When all four families are considered, there are 17,683 features. It is important to note that these methods can easily be adapted to any features that the analyst deems interesting.

Describing Zeus Subfamilies

To begin, the 3,620 instances of Zeus are clustered using a standard k-means algorithm with $k = 15$. Figure 2 displays the resulting descriptions for the integer linear program and the token method with a threshold. The ILP method generates a more compact description than the token method. This is due to the ILP method being able to use negative relationships such as not having the file access `win.ini`. In this instance, the ILP was also able to find a higher fidelity description. The ILP description hit 91.94% of the instances within that cluster and had 0 false positives. The token method's description hit 77.01% of the instances within the cluster, but also hit 3.38% of the instances not in the cluster.

Table 2 lists the results for all 15 clusters. True positives is the percentage of files within the cluster that the rule hit, and false positives is the percentage of files in other clusters that the rule hit. The ILP method compares favorably in most clusters. The ILP method does fail to find descriptions in four cases because there are no feasible solutions. When a method fails to find a rule, a null rule is generated that will naturally have 100% false positives and negatives.

There are cases, e.g. cluster 6, cluster 13, and cluster 15, when the ILP method cannot find a feasible solution, but the token methods can find a rule. In all of these cases, the token methods will have false positives. In our experiments, when the ILP method does not find a solution, the false positive percentages of the token methods can be as high as 15-17%.

	Token		Token (threshold)		ILP	
Cluster	True Positives	False Positives	True Positives	False Positives	True Positives	False Positives
C1	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
C2	100.00%	84.60%	87.85%	26.32%	80.55%	0.00%
C3	100.00%	75.04%	100.00%	75.04%	95.40%	0.00%
C4	100.00%	0.42%	74.51%	0.08%	88.24%	0.00%
C5	100.00%	0.43%	82.52%	0.00%	93.71%	0.00%
C6	100.00%	1.20%	65.22%	0.06%	100.00%	100.00%
C7	100.00%	19.90%	93.81%	17.97%	87.63%	0.00%
C8	100.00%	85.59%	71.67%	0.53%	88.33%	0.00%
C9	100.00%	1.70%	95.70%	1.62%	94.62%	0.00%
C10	100.00%	100.00%	92.00%	0.00%	96.00%	0.00%
C11	100.00%	85.29%	91.67%	0.00%	91.67%	0.00%
C12	100.00%	100.00%	100.00%	100.00%	98.39%	0.00%
C13	100.00%	17.04%	84.00%	15.07%	100.00%	100.00%
C14	100.00%	25.67%	77.01%	3.38%	91.94%	0.00%
C15	100.00%	2.28%	93.65%	2.02%	100.00%	100.00%

Table 2: Results for the clusters of Zeus subfamilies. True positives is the percentage of instances within the cluster that the generated rule hit. False positives is the percentage of instances not belonging to the cluster that the generated rule hit.

		Number of features				
		100	500	1000	5000	10000
Number of samples	100	0.01	0.10	0.10	1.33	6.16
	500	0.02	0.15	0.44	6.81	20.87
	1000	0.04	0.42	0.91	9.20	42.02
	5000	0.28	4.28	6.43	38.80	272.02
	10000	0.59	8.89	15.71	102.03	489.16

Table 3: Timing results for various combinations of different feature vectors sizes and number of instances on randomly generated binary data. All times reported are in seconds.

On the other hand, the ILP method can find solutions when the token method cannot. For instance, the ILP method finds a description for cluster 12 that hits on 98.39% of the files within the cluster and has no false positives. The flexibility of the optimization problem and encoding of the data allows for negative relationships, and makes the ILP method more robust to situations where the cluster does not have many common features. For cluster 12, the lack of three IP addresses, a cryptography registry key, and the `winlogon` registry key is what makes it unique.

Four Family Descriptions

Using all four families from Table 1, there are 8,665 instances and 17,683 features. Each of the four families were first clustered with k-means with $k = 10$ for a total of 40 clusters. When the families were not first clustered into subfamilies, the ILP and token methods returned null rules. This could be a side effect of noisy labels and/or a feature space that is not expressive enough.

The results for all 40 clusters are shown in Table 4. Many of the same patterns emerged that were found in the Zeus subfamilies. For instance, cluster 5 is an example of the ILP method failing to find a proper description, but the token methods do find rules with high false positives. Cluster 3 is interesting because all the methods found the same description, which had 100% true positive rate and 0% false positive rate.

Timing Analysis

As mentioned in Section , even though integer linear programming is `NP`-complete, there are many good libraries that can efficiently approximate the solution. Again, for this work, python was used with the CVXOPT optimization library (Andersen, Dahl, and Vandenberghe) and the GLPK (glp) bindings. GLPK uses a branch-and-cut algorithm (Padberg and Rinaldi 1991) for its integer linear programming optimization.

Table 3 contains the timing results for a variety of different feature sizes and sample sizes. In these experiments, random binary vectors are generated with the probability of a 1 being set to .2 for each feature. It is important to note that ILP solvers typically run much more efficiently on

	Token		Token (threshold)		ILP	
Cluster	True Positives	False Positives	True Positives	False Positives	True Positives	False Positives
C1	100.0%	41.21%	76.58%	1.37%	73.87%	0.00%
C2	100.0%	88.05%	99.59%	87.71%	97.94%	0.00%
C3	100.0%	0.00%	100.00%	0.00%	100.00%	0.00%
C4	100.0%	7.72%	92.51%	7.25%	84.15%	0.00%
C5	100.0%	32.64%	90.63%	22.02%	100.00%	100.00%
C6	100.0%	18.40%	76.02%	0.00%	87.80%	0.00%
C7	100.0%	2.60%	76.57%	0.01%	77.14%	0.00%
C8	100.0%	100.00%	82.22%	17.31%	100.00%	100.00%
C9	100.0%	0.36%	37.50%	0.01%	68.75%	0.00%
C10	100.0%	2.37%	97.18%	1.99%	85.92%	0.00%
C11	100.0%	9.96%	75.78%	2.62%	100.00%	100.00%
C12	100.0%	10.83%	81.15%	0.52%	89.62%	0.00%
C13	100.0%	0.28%	67.39%	0.23%	95.65%	0.00%
C14	100.0%	0.06%	35.71%	0.02%	75.57%	0.00%
C15	100.0%	90.94%	84.00%	0.55%	85.00%	0.00%
C16	100.0%	0.00%	33.33%	0.00%	95.83%	0.00%
C17	100.0%	10.02%	92.42%	3.89%	100.00%	100.00%
C18	100.0%	13.04%	86.79%	2.24%	100.00%	100.00%
C19	100.0%	0.86%	77.27%	0.00%	84.09%	0.00%
C20	100.0%	100.00%	92.37%	51.29%	73.73%	0.00%
C21	100.0%	11.86%	87.67%	3.17%	100.00%	100.00%
C22	100.0%	90.95%	79.55%	12.42%	100.00%	100.00%
C23	100.0%	44.74%	76.75%	6.65%	76.20%	0.00%
C24	100.0%	91.03%	91.64%	25.46%	79.39%	0.00%
C25	100.0%	3.30%	80.65%	0.96%	100.00%	100.00%
C26	100.0%	12.77%	84.78%	2.94%	100.00%	100.00%
C27	100.0%	3.24%	52.38%	0.14%	100.00%	100.00%
C28	100.0%	0.00%	100.00%	0.00%	100.00%	0.00%
C29	100.0%	100.00%	99.33%	99.58%	97.48%	0.00%
C30	100.0%	91.36%	83.33%	3.79%	85.71%	0.00%
C31	100.0%	0.06%	74.14%	0.05%	94.83%	0.00%
C32	100.0%	0.00%	100.00%	0.00%	100.00%	0.00%
C33	100.0%	0.24%	55.55%	0.00%	94.44%	0.00%
C34	100.0%	52.30%	69.23%	0.00%	78.97%	0.00%
C35	100.0%	0.01%	95.83%	0.00%	95.83%	0.00%
C36	100.0%	0.00%	63.64%	0.00%	100.00%	0.00%
C37	100.0%	45.75%	93.09%	44.92%	100.00%	100.00%
C38	100.0%	0.73%	97.06%	0.50%	100.00%	100.00%
C39	100.0%	99.55%	96.52%	2.58%	100.00%	100.00%
C40	100.0%	13.03%	90.35%	0.00%	100.00%	100.00%

Table 4: Results for the clusters of the four families. True positives is the percentage of instances within the cluster that the generated rule hit. False positives is the percentage of instances not belonging to the cluster that the generated rule hit.

sparser data. The malware data presented in this paper had roughly 15-20% of their features being non-zero. As Table 3 demonstrates, the ILP solver scales favorably with the number of samples. But, the ILP method does perform worse when the number of features increases. In a real-world application of these techniques, it would be important to spend time considering what features are absolutely necessary to the problem, either manually or through dimensionality reduction methods.

Conclusion

This paper has demonstrated a simple and intuitive method for constructing compact, domain-specific explanations for clusters. While there have been many attempts to assess the validity of a set of clusters through different indices and visualizations, there has been little work in creating actual hu-

man readable descriptions. The strength of the integer linear programming formulation is that it uses the non-cluster instances. This allows for a minimal set of features and guarantees that the generated description does not hold for samples belonging to other clusters.

We demonstrated the results of our method on real-world malware data. The four malicious families that we collected were run in a sandbox where behavioral characteristics, such as connections to IP addresses and registry modifications, were collected. We showed that the integer linear program could find compact descriptions of the clusters behavior. This can potentially save reverse engineers weeks worth of effort. While the generated descriptions can be used for human consumption, they may also be converted into indicators of compromise using formats such as STIX (sti).

References

Andersen, M. S.; Dahl, J.; and Vandenberghe, L. CVXOPT: A Python Package for Convex Optimization, Version 1.1.6. [Online; accessed 12-May-2015].

Anderson, B.; Storlie, C.; and Lane, T. 2012. Multiple Kernel Learning Clustering with an Application to Malware. In *Proceedings of the International Conference on Data Mining*, ICDM, 804–809. IEEE.

Bayer, U.; Comparetti, P. M.; Hlauschek, C.; Kruegel, C.; and Kirda, E. 2009. Scalable, Behavior-Based Malware Clustering. In *Network and Distributed System Security (NDSS)*, volume 9, 8–11. Citeseer.

Fielding, A. 2007. *Cluster and Classification Techniques for the Biosciences*. Cambridge University Press Cambridge.

GNU Linear Programming Kit, Version 4.55. http://www.gnu.org/software/glpk/glpk.html. [Online; accessed 12-May-2015].

Halkidi, M.; Batistakis, Y.; and Vazirgiannis, M. 2002. Clustering Validity Checking Methods: Part II. *ACM Sigmod Record* 31(3):19–27.

Jain, A. K.; Murty, M. N.; and Flynn, P. J. 1999. Data Clustering: A Review. *ACM Computing Surveys (CSUR)* 31(3):264–323.

Kriegel, H.-P.; Kröger, P.; and Zimek, A. 2009. Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 3(1):1.

Newsome, J.; Karp, B.; and Song, D. 2005. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *Symposium on Security and Privacy (S&P)*, 226–241. IEEE.

Padberg, M., and Rinaldi, G. 1991. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM review* 33(1):60–100.

Rieck, K.; Holz, T.; Willems, C.; Düssel, P.; and Laskov, P. 2008. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 108–125. Springer.

Schrijver, A. 1998. *Theory of Linear and Integer Programming*. John Wiley & Sons.

STIX. https://stix.mitre.org/. [Online; accessed 12-May-2015].

Strumbelj, E., and Kononenko, I. 2010. An Efficient Explanation of Individual Classifications using Game Theory. *The Journal of Machine Learning Research* 11:1–18.

Wagstaff, K. L.; Lanza, N. L.; Thompson, D. R.; and Dietterich, T. G. 2013. Guiding Scientific Discovery with Explanations using DEMUD. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, AAAI.

Wang, K.; Cretu, G.; and Stolfo, S. J. 2006. Anomalous Payload-Based Worm Detection and Signature Generation. In *Recent Advances in Intrusion Detection (RAID)*, 227–246. Springer.

YARA. http://plusvic.github.io/yara/. [Online; accessed 12-May-2015].