Generative Malware Outbreak Detection

Sean Park, Iqbal Gondal, Joarder Kamruzzaman, Jon Oliver

TREND MICRO™ | research

Published by:
Trend Micro Research

Written by:
Sean Park
Trend Micro Research

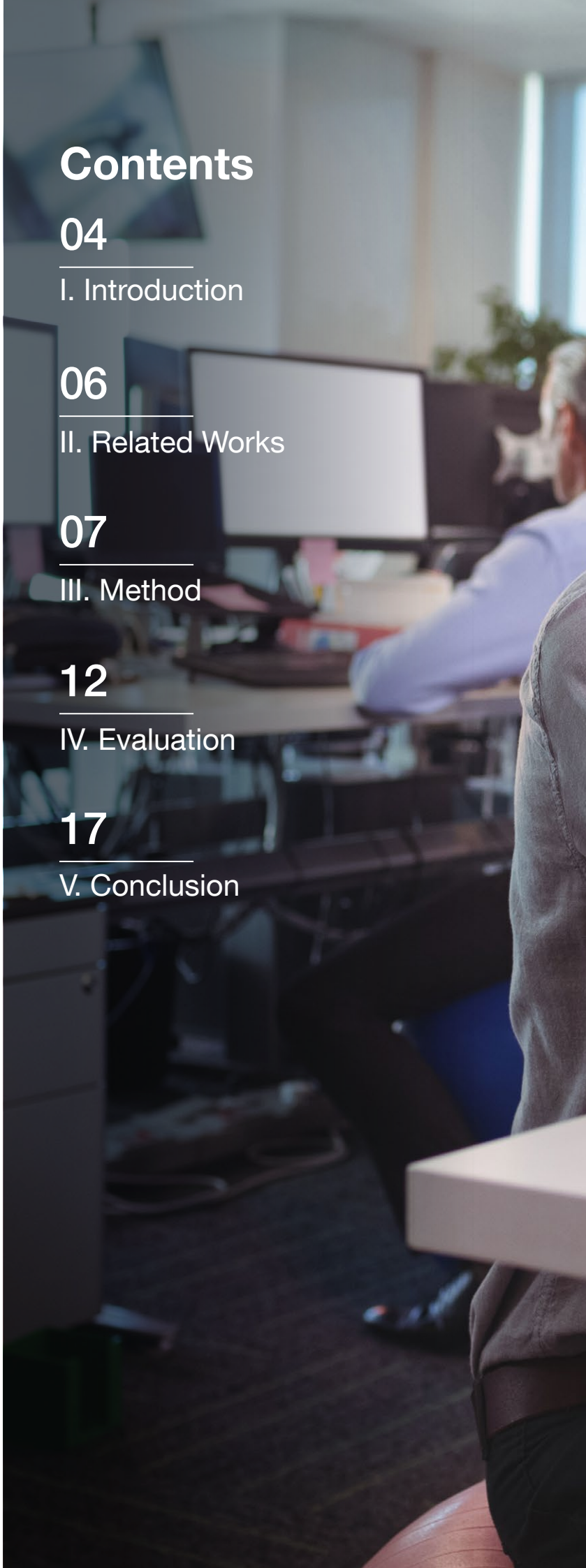
Iqbal Gondal
Federation University Australia

Joarder Kamruzzaman
Federation University Australia

Jon Oliver
Trend Micro Research

Stock images used under license from Shutterstock.com

Contents



Abstract

Recently, several deep learning approaches have been attempted to detect malware binaries using convolutional neural networks and stacked deep autoencoders. Although such approaches have shown satisfactory performance on a large corpus of dataset, practical defense systems require precise detection during malware outbreaks where only a handful of samples are available. This paper demonstrates the effectiveness of the latent representations obtained through the adversarial autoencoder for malware outbreak detection. Using instruction sequence distribution mapped to a semantic latent vector, this model provides a highly effective neural signature that helps detect variants of a previously identified malware within a campaign that have mutated with minor functional upgrades, underwent function shuffling, or have slightly modified obfuscations. This method demonstrates how adversarial autoencoder can turn a multiclass classification task into a clustering problem when the sample set size is limited and the distribution is biased. The model performance is evaluated on OS X malware dataset against traditional machine learning models.

I. Introduction

Statically identifying malware has been the most viable approach when timely detection is critical. This is especially true under a current threat environment where malware outbreaks have become part of the daily routine. This paper considers static features for malware detection. The malware packing problem[1] is crucial when the objective is to reverse engineer the detailed functional characteristics of a given malware. However, when the sole objective is to detect malware, its surface-level static features are sufficient to differentiate malware families from benign samples. Therefore, this paper does not attempt to propose a method to unpack malware samples.

An observation in the malware battlefront is that malware mutates over time to bypass static signature-based detection by either upgrading its functions or applying new metamorphic (or obfuscation) techniques. The downside for attackers is that malware mutation requires time and effort to do. Due to this developmental cost, minor tactical modification to the original malware code frequently occurs and arrives in the form of an outbreak, while a major strategic code change rarely occurs across a longer period of time. This inevitably causes a similar pattern of instruction sequence that is either generated by a metamorphic engine or upgraded from previous functions. As a result, there appears a phenomenon typically seen in the instruction sequence of malware samples from a campaign, as shown in Figure 1. The method used in this paper will exploit the presence of this unique pattern of instruction sequence in the malware samples of a campaign to determine whether or not it forms unpacking routines, metamorphic components, or pure functional modification.

Traditional machine learning algorithms such as Support Vector Machine (SVM), Random Forest, and Gradient Boosting commonly use metadata as features, such as executable file header fields, n-gram of raw binary file, and entropy of sections, because they are optimized to work with independent and sparse features. Meanwhile, encoded high-dimensional data — such as a sequence of program instructions — constitutes the substantial body of a sample in the context of malware detection, as this contains rich information of the sample's identity. Although several attempts utilized the instruction sequence as a feature,[2, 3, 4, 5] the n-gram reduction they performed simply throws away the sequence order information, leaving those approaches vulnerable to a trivial histogram matching attack.[6] However, the adversarial autoencoder used in this paper, like many other deep learning models, fully takes advantage of the input samples with the sequence order retained. In this paper, we use the sequence of program instructions as a feature.

One critical aspect of malware outbreak detection is the scarce number of samples we can train our systems with. The goal of this research is to introduce a method that not only detects malware variants but also detects them with an extremely small number of samples captured at the very early outbreak — as what occurs in a real-world malware detection scenario. This method, as well as various other machine learning models, will be evaluated with this scarce training dataset setting.

This paper presents a novel method that detects similar malware samples with high accuracy for malicious samples and low false positives for benign samples, using a single sample for training with adversarial autoencoder. The techniques described in this paper are applicable to other domains, such as the internet of things (IoT), that require well-generalized detection using a handful of malware samples.

II. Related Works

For binary malware classification, Joshua Saxe and Konstantin Berlin used fully connected layers with dropouts over handcrafted independent sparse features from an executable file.[7] The approach does not deal well with complex variations of metamorphism especially when the changes occur globally within the sample, let alone when the binary classification significantly overfits the limited number of training samples. Yuancheng Li et al recorded 92.1% accuracy over a binary classification problem with experimental data from KDDCUP '99 dataset using a stacked RBM (Restricted Boltzmann Machine) autoencoder combined with softmax regression.[8] Although the method did not use layer-wise noises for the autoencoder and the dataset consisted of handcrafted sparse features, the efficacy of using autoencoder was demonstrated. Eli (Omid) David and Nathan Netanyahu demonstrated the use of DBN (Deep Belief Network) for malware detection over a relatively concentrated number of target classes.[9] George Dahl et al tried to solve the malware classification problem using DBN with 136 malware family categories as output classes.[10] However, it still used a sparse feature set. Using convolutional neural network, Andrew Davis and Matt Wolff first adopted raw binary samples as features, which contain program codes inside.[11] However, the binary distribution of a malware outbreak can significantly vary depending on the packed code and data. The layout of sections can also change with little effort from the attacker. Using highly structured content such as an executable file as a sequence of raw bytes is less likely to generalize the distributions of a malware and its variants. Park introduced a method to detect malware metamorphism using a stacked de-noising autoencoder and semantic hashing on the features generated by Fourier transform applied to the program instructions.[12] Although the method used successfully captures intra-function metamorphism, no sample-wise similarity detection approach was suggested.

The majority of these approaches use a large dataset with the assumption that the dataset has unbiased distribution across different styles of samples. In addition, many approaches attempted to solve binary classification problems, where samples are labeled in one of two classes. The proposed approach in this paper attempts to solve multiclass classification problems by using unsupervised machine learning with no assumptions on the number of training samples and sample distribution.

III. Method

This section discusses our dataset, its features, the model architecture, and training methods.

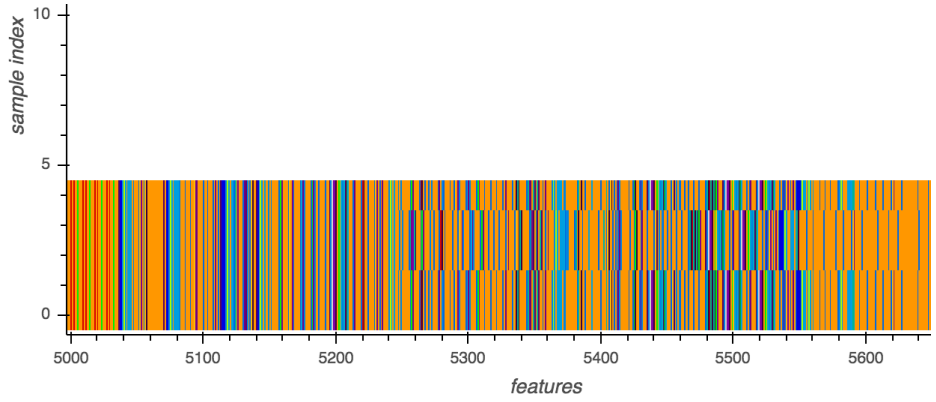


Figure 1. Visual analysis of three unique variants of *MAC.OSX.CallMe* family

Note: Each row represents a per-sample feature, which is a sequence of instructions of a malware sample. Each normalized instruction is rendered as a vertical bar with a unique color to differentiate between different instructions. The X-axis represents the feature while the Y-axis represents the sample number.

A. Features

In modern days, malware samples are automatically generated by a custom tool created by the attacker. It renders hard-coded static signature-based detection obsolete. A run of automatic malware generation tool essentially creates a batch of the functionally same malware in a different look, which can involve different obfuscations such as dead code insertion,[13, 14] register reassignment, code transposition, and integration and control flow obfuscation. Nonetheless, the observation is the distribution of the program instruction sequence remains relatively intact. Figure 1 shows three unique variants of *MAC.OSX.CallMe* family. The samples in Figure 1 are described below.

```
MAC.OSX.CallMe.A (3 samples)
MAC.OSX.CallMe.E (1 sample)
MAC.OSX.CallMe.F (1 sample)
```

As shown in Figure 1, *MAC.OSX.CallMe* variants have identical instruction sequences until a variation was introduced at approximately instruction 5250. Despite this variation, it is visually clear that parts of instructions of sample 2 and sample 3 merely shifted from the instructions of the rest of the samples. In short, the instruction sequences appear very similar to each other for these three *MAC.OSX.CallMe* variants. A visual analysis of the majority of malware families shows that program instruction sequence plays a significant role in identifying the variants during outbreaks.

Based on this observation, we therefore use the instruction sequence as the sole feature for the model proposed in this paper. We describe the steps in constructing a feature vector for a sample and show example values in Figure 2.

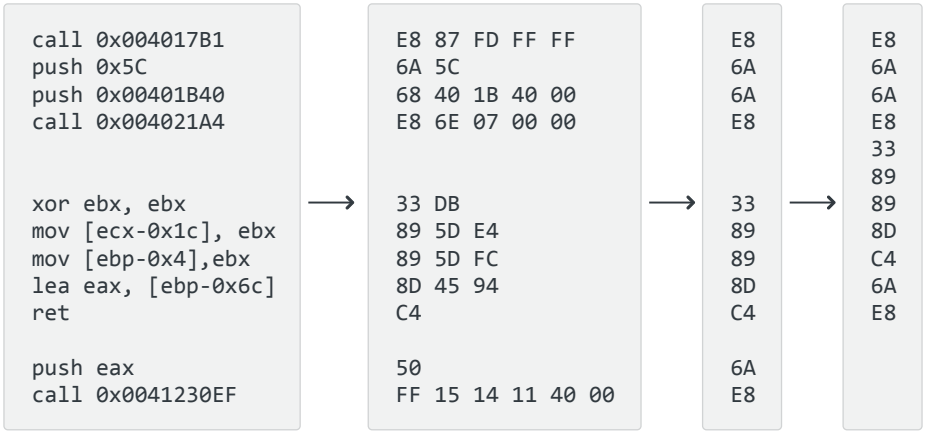


Figure 2. Example values for each step of the feature extraction

1. *Extract function-wise raw instruction bytes using IDA Pro*:[15] It is critical to extract the original raw features. Failing that, sample distribution will change, which will significantly affect the clustering result. In this paper, a list of function bytes are extracted from each malware sample using a custom IDA Python script.
    a. Create a sample by combining the extracted functions: Each individual data sample per malware sample needs to be created. This is done by concatenating the functions present in the executable file in the order they appear. A blind concatenation of functions is vulnerable to code transposition and integration metamorphism.[16] Overcoming this problem is later discussed in this section using a model that contains a translation invariant property.
2. *Map each instruction byte to a unique instruction ID*: An instruction's operands are ignored. For example, both push *0x5C* and *push eax* are mapped to a unique ID, *6A*. Note that this unique ID is computed using a custom table instead of being assigned directly from the instruction's opcode because the opcode in CPU architecture may include a portion of a byte or may span across multiple bytes. The rationale for this preprocessing is that it reduces noises in the distribution while remaining immune to several obfuscation techniques, such as register and memory reassignment (see Moser et al).

B. Adversarial Autoencoder (AAE)

The model consists of two independent modules. First, the latent representation for the instruction sequence feature that is resilient to metamorphism is acquired by adversarial autoencoder. Second, the class number for the latent representation is computed via HDBSCAN with a predefined threshold.

Over the past few years, GAN (Generative Adversarial Network) has successfully demonstrated its capability to understand the data distributions by generating realistic samples.[17] The power of GAN primarily comes from its generative nature by jointly training the generator and the discriminator in a tight competitive loop. In a situation like malware outbreaks where a handful of samples are available, adversarial autoencoder is a natural choice so that the scarce number of training samples produces smooth approximated nearby distributions. The core architecture for malware outbreak detection in this paper is borrowed from the original adversarial autoencoder, as seen in Figure 3.

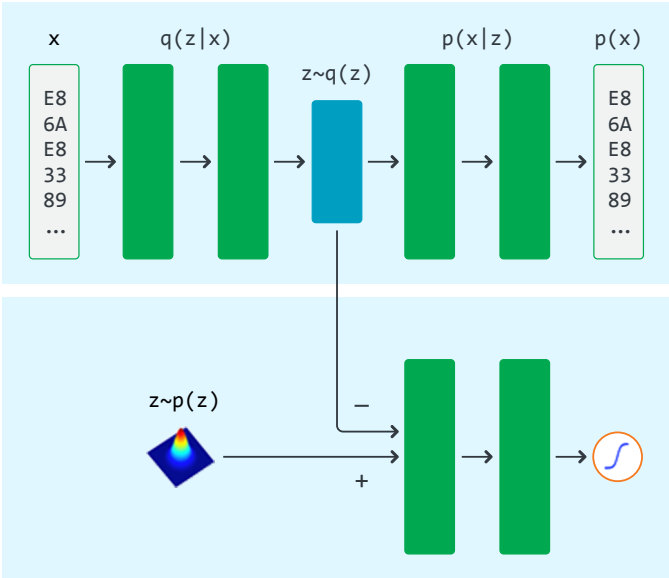


Figure 3. Adversarial autoencoder architecture used for malware outbreak detection

Note: The input, x, and the reconstructed input, p(x), have the instruction sequence feature.

Adversarial autoencoder essentially combines an arbitrary autoencoder with GAN. The autoencoder part within the model must have two properties:

- The stacked weights are symmetric and shared between encoder and decoder. This is a compulsory requirement to qualify as autoencoder.
- Encoder also functions as a generator; hence, it must have all of GAN's generator properties as well. Since encoder functions dually, it needs to conform to the training techniques used for the generator while maintaining the autoencoder property.

During malware outbreaks, one of the desired detection properties is the ability to identify relocated functions. Thus, the autoencoder used in the proposed model in this paper is a stacked convolutional autoencoder that aims to take advantage of the translation invariant property of the architecture.[18] This allows the model to capture the program instruction sequence in the presence of code transposition and integration metamorphism.[19]

The input vector consists of sparse discrete symbols, which are difficult to train with a stochastic gradient descent. Therefore, we create an embedding lookup for the symbols and let the model find the best representations for them during the training. This embedding layer nicely transforms a 1D input vector into a 2D array, which can be fed as an input to this convolutional autoencoder. The reconstruction method based on cosine similarity cross entropy is used to deal with sparse discrete input symbols.[20]

As outlined in a paper written by Alireza Makhzani et al,[21] both the adversarial network and the autoencoder are trained jointly with stochastic gradient descent in two phases — the reconstruction phase and the regularization phase — and are then executed on each mini-batch. Specifically, in the reconstruction phase, the model is trained by minimizing the cross entropy loss between the input symbol and the decoder output via sigmoid activation. During the regularization phase, binary cross entropy is used for the discriminator loss, which is computed by summing the loss between positive samples from the Gaussian normal distribution and negative samples from the encoder output. Binary cross entropy is also used for the generator loss. Let x be the one-hot encoded representation of input data distribution and z be the latent code vector of autoencoder. Let $p(z)$ be the prior distribution imposed on the codes, $q(z|x)$ be the encoding distribution, $p(x|z)$ be the decoding distribution, and $p(x)$ be the model's reconstructed distribution. Reconstruction loss is described in Formula (1), and discriminator and generator loss are defined by Formula (2).

1
$$E_x[E_{q(z|x)}[-\log p(x|z)]]$$

2
$$\min_G \max_D \; E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1-D(G(z))]$$

Consensus optimization has been adopted to mitigate the instability caused by standard adversarial autoencoder training. Mescheder showed that the simultaneous gradient descent used in GAN does not generally converge to a Nash equilibrium in a non-cooperative minmax game.[22] Mescheder proposed to solve this by constructing a conservative vector field from the original using consensus optimization.[23, 24]

There are a number of hyperparameters that can be tuned in the network architecture, for example, how much of diverse clusters you want to detect and what level of performance you need during prediction. In terms of clustering behavior, the standard deviation of GAN's Gaussian noise input generally affects the total number of clusters that are detectable with accuracy. The wider the Gaussian normal distribution is, the larger the number of clusters that the model will spread evenly. A large size latent vector increases the accuracy of clustering. On the contrary, a large embedding size for an input symbol does not have a significant impact on clustering accuracy.

In general, the convolutional autoencoder part of the network does not have much impact on accuracy, but the increased number of convolutional layers can significantly reduce both training and prediction speeds. A large convolutional filter window tends to produce less optimal results.

From a training perspective, batch normalization within the GAN generator is necessary to help generate a consistent latent representation. The Adam optimizer was used for both the reconstruction and regularization phases,[25] while consensus optimization was performed with RMSProp.[26] Some of the key hyperparameters are shown below.

Latent representation dimension: 100
Input Gaussian noise standard deviation: 5.0
Embedding dimension: 4
Number of channels for each convolution layer: [1, 20, 20, 1]
Filter sizes: [3, 3, 3, 3]
Strides: [1, 2, 2, 1]
Maximum epochs: 100
Learning rate: 0.0001
Batch size: 20

When training is complete, the encoder output is taken as a latent vector that represents the input sample, which will be used as an input used for semantic hashing.

C. Semantic Hashing

The latent representation obtained through adversarial autoencoder needs to be transformed into a class number for prediction. First, the latent vector represented by real valued numbers are binarized using the bitwise mean value of the training samples. The hamming distance is then used to compute the distance for the two given latent vectors.[27] Finally, a test sample is assigned a class with the closest training sample.

IV. Evaluation

A. Dataset

3,254 in-the-wild OS X malware samples collected from a proprietary source and 9,981 randomly chosen benign OS X Mach-O samples were used for evaluation. A snapshot of the family distribution of 3,254 in-the-wild malware samples is shown in Figure 4. In order to simulate the outbreak situation, 175 out of 3,254 malicious samples that exhibited unique instruction sequence patterns were manually selected by a human malware expert as core malicious training samples and were assigned a unique label for each sample. Note that no benign samples were included in the training set.

Since there is no generic evaluation metric available in finding a core sample of a malware family that's based on instruction sequence, the instruction sequences of all 3,254 malicious samples were visually explored to obtain the core sample of each family. The properties that are used to put samples in the same category are summarized as follows:

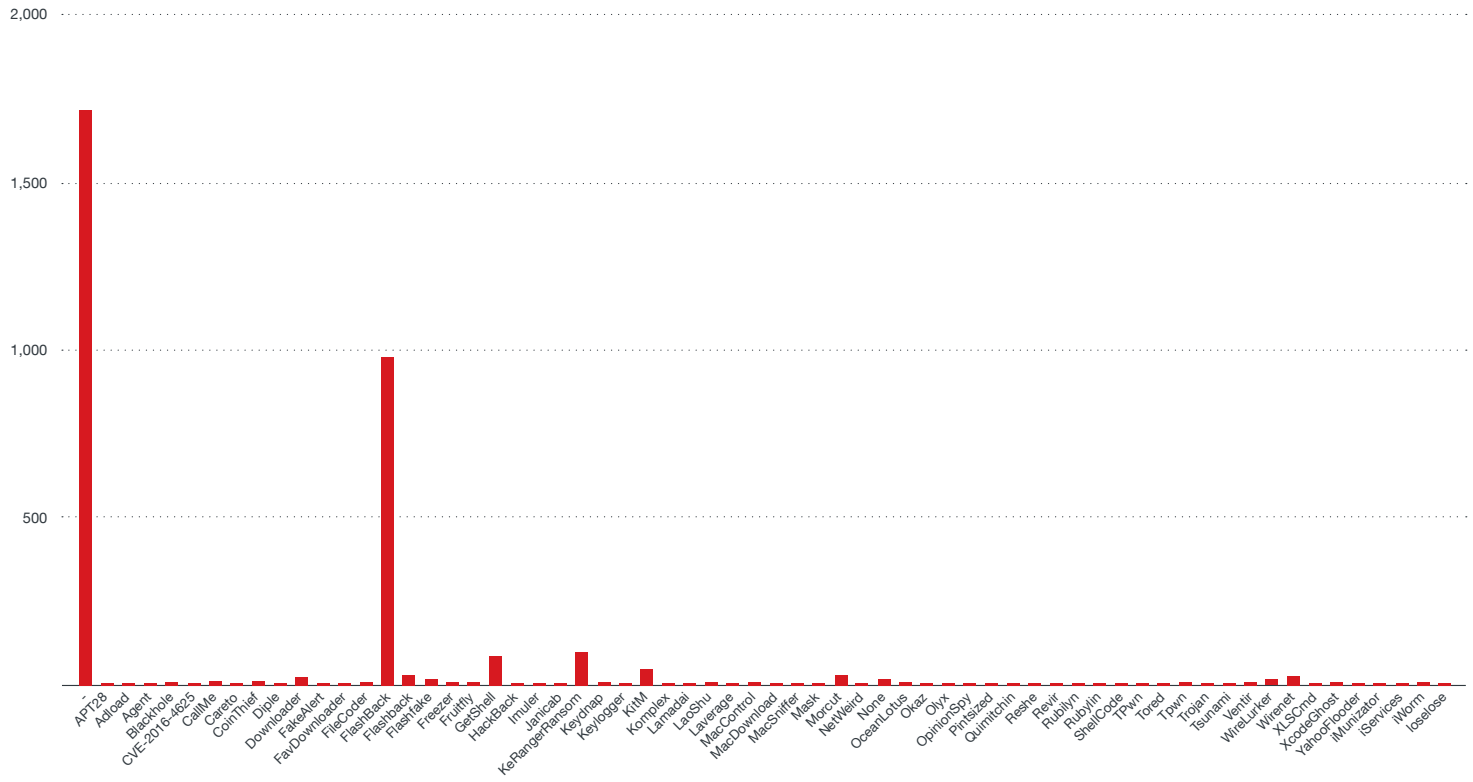


Figure 4. Malicious sample distribution by VirusTotal detection names.

Note: The biggest bar at the far left hand side indicates the samples with no detection or only has a generic name.

- Similar instruction distribution: The samples have similar instruction distribution statistics.
- Minor local variations: The modification of a sample's instruction sequence is restricted to one or more local areas.
- Translation invariant: Most part of the sample's code distribution is identical to the rest of them within the same cluster when the code is translated or shuffled.

The properties for unqualified malware families are summarized as follows:

- Mismatched function-wise distribution: Neither similar sample length nor similar statistical distribution qualifies a sample to become a member of a cluster. Samples must also match function-wise statistical distributions.
- Substantial difference in code distribution size: Although a partial match suggests a variant, it is desirable to have the size of similar code distribution significantly larger than the variations. It's because the clusters can drift over repetitive trainings across longer periods of time, which can potentially cause false positives due to mixed distributions.

In order to find out the category for the latent representation, HDBSCAN clustering algorithm was used, which shows the most appealing performance against unknown numbers of clusters.[28] The samples categorized as a noise by HDBSCAN are classified as benign because this indicates that no similar cluster was found in the test sample. The overall flow chart is illustrated in Figure 5.

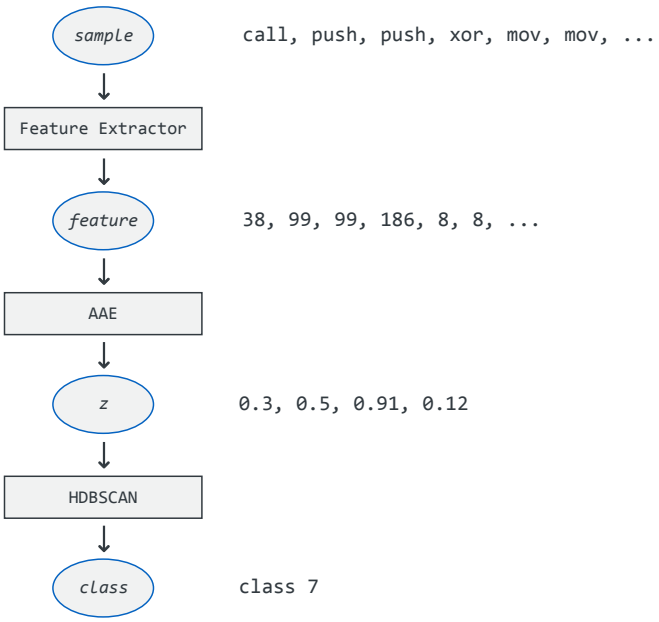


Figure 5. Overall pipeline from a sample to its predicted class number (Examples are shown on the right)

B. Result

We found that using raw instruction sequence for classification models significantly reduces accuracy. Gradient boosting, Support Vector Machine, and Random Forest models were chosen as baseline with the feature implemented using n-gram[29] over the instruction sequence. Clustering models such as KNN were not included in the baseline model since they need a decent number of training samples to work, which is different to the problem setting put forward in this research.

As shown in Table 1, traditional classifiers perform reasonably well even for a training dataset that consists of a single sample for each class. The proposed model, *aae-sh*, which is adversarial autoencoder combined with semantic hashing, shows reasonably high detection accuracy against malicious samples. However, we found that all traditional classification models catastrophically fail on benign samples, recording 100% false positives. With the training set of only core malicious samples by which outbreaks are simulated, the traditional classification methods do not work at all. On the contrary, *aae-sh*, records a 91% accuracy over benign samples with this training setting.

Model	Malicious (3,254)	Benign (9,981)
gradient-boosting-1gram	0.935	0.000
gradient-boosting-2gram	0.936	0.000
gradient-boosting-3gram	0.931	0.000
svm-1gram	0.934	0.000
svm-2gram	0.944	0.000
svm-3gram	0.968	0.000
randomforest-1gram	0.983	0.000
randomforest-2gram	0.987	0.000
randomforest-3gram	**0.989**	0.000
aae-sh	0.959	**0.910**

Table 1. Detection rate against malicious and benign samples for various models.

C. Analysis

Visual analysis of the families detected by *aae-sh* not only shows similar instruction sequences with variations within the family but it also does not exhibit undesirable properties described in the previous subsection. Figure 6 shows *aae-sh* correctly identifying malware variants whose major feature mass is identical across all samples in the cluster while variations occur in many different ways. These samples are the variants of malware named *Blackhole* or *Freezer*. It becomes clear that the names from VirusTotal[30] do not necessarily match the malware clusters produced by *aae-sh* because human analysts tag detection names based on analyst-specific heuristics, whereas the proposed approach in this paper derives the detection purely from the instruction sequence pattern.

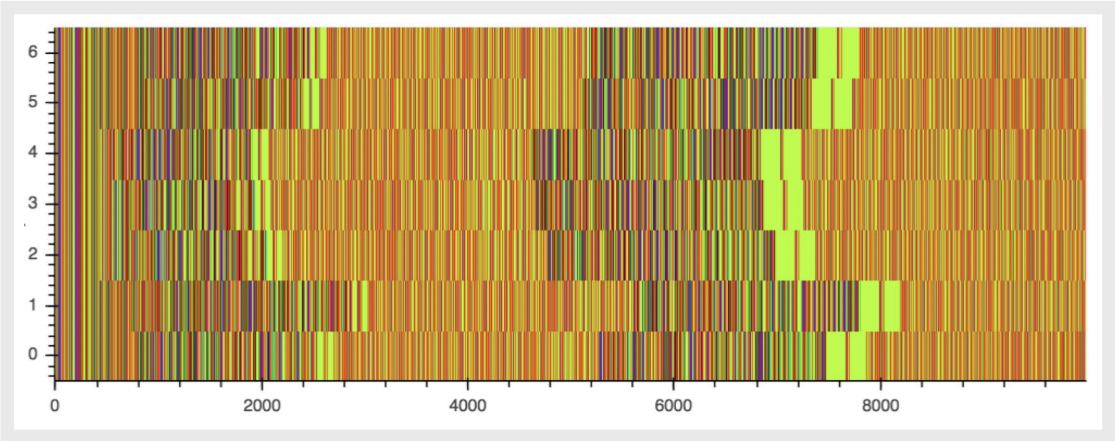


Figure 6. Visualization of the instruction sequences of Blackhole or Freezer samples identified by *aae-sh*

Note: The X-axis represents the feature while theY-axis represents the sample number.

Figure 7 shows the detected cluster 49 that contains many *Flashback* variants. Note that *aae-sh* detected the samples of different lengths as long as the instruction sequences are similar.

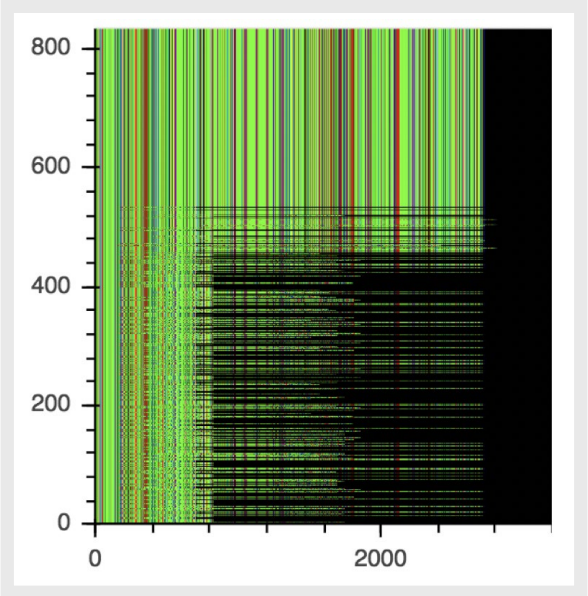


Figure 7. Visualization of the instruction sequences of malware samples within cluster 49 identified by *aae-sh*

Note: The X-axis represents the feature while the Y-axis represents the sample number.

```
Cluster 49 (nsamples=836)
     FlashBack.AF (2 samples)
     FlashBack.L (344 samples)
     FlashBack.M (5 samples)
     FlashBack.Q (2 samples)
     Flashback.E (1 sample)
     Flashback.J (1 sample)
     Flashback.K (1 sample)
     Flashback.L (1 sample)
     Flashback.M (10 samples)
     Flashback.N (7 samples)
     Flashback.O (1 sample)
     Flashback.P (1 sample)
     Flashback.Q (1 sample)
     Trojan-Downloader.Flashfake.ab (12 samples)
     Unknown (447 samples)
```

Figure 8. VirusTotal detection names for the samples in cluster 49 that are visualized in Figure 6

Figure 8 shows redacted VirusTotal detection names for the samples in Figure 7. It is notable that 447 samples either had no names or had generic names.

V. Conclusion

The research shows that the generative power of adversarial autoencoder creates latent representations that can be used to identify similar samples with minimal number of training samples. It turned out that some malware families such as Flashback reuse the same piece of code repeatedly across their variants, and this subsequently enables the adversarial autoencoder to identify the family effectively. In addition, the model was found to be effective in discovering multiple variants across heterogeneous malware families that share similar instruction-wise characteristics.

References

1. Fanglu Guo, Peter Ferrie, and Tzi-Cker Chuiueh. (2008). “A study of the packer problem and its solutions.” Heidelberg, DE: Springer-Verlag Berlin.

2. Igor Santos, et al. (2009). *ICEIS*. “N-grams-based File Signatures for Malware Detection.” Last accessed on 16 January 2019 at https://pdfs.semanticscholar.org/1ba4/d1666d6d9ab784063202d78fba1838ca03cf.pdf.

3. Abdurrahman Pektaş, Mehmet Eriş, and Tankut Acarman. (2011). *The Fifth International Conference on Emerging Security Information, Systems and Technologies*. “Proposal of n-gram based algorithm for malware classification.” Last accessed on 16 January 2019 at https://www.thinkmind.org/download.php?articleid=securware_2011_1_30_30099.

4. Nwokedi Idika and Aditya P. Mathur. (2007). “A survey of malware detection techniques.” Last accessed on 16 January 2019 at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.4594&rep=rep1&type=pdf.

5. Sachin Jain and Yogesh Kumar Meena. (2011). *5th International Conference on Information Processing, ICIP 2011 Bangalore, India, August 5-7, 2011 Proceedings*. “Byte level n-gram analysis for malware detection.” Heidelberg, DE: Springer-Verlag Berlin.

6. Wikimedia Foundation Inc. (n.d.) *Wikipedia*. “Histogram Matching.” Last accessed on 16 January 2019 at https://en.wikipedia.org/wiki/Histogram_matching.

7. Joshua Saxe and Konstantin Berlin. “Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features.” In *10th International Conference on Malicious and Unwanted Software (MALWARE)*, 20 October 2015.

8. Yuancheng Li, Rong Ma, and Runhai Jiao. (2015). “A Hybrid Malicious Code Detection Method based on Deep Learning.” Last accessed on 28 January 2019 at https://pdfs.semanticscholar.org/45ba/f042f5184d856b04040f14dd8e04aa7c11f6.pdf.

9. Eli (Omid) David and Nathan Netanyahu. “DeepSign: Deep learning for automatic malware signature generation and classification.” In *10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 1-8, 12 July 2015.

10. George Dahl, Jack Stokes, Li Deng, and Dong Yu. (2013). “Large-Scale Malware Classification Using Random Projections and Neural Networks.” In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422-3426, 26 May 2013.

11. Andrew Davis and Matt Wolff. (2015). *Blackhat*. “Deep Learning on Disassembly Data.” Last accessed on 28 January 2019 at https://www.blackhat.com/docs/us-15/materials/us-15-Davis-Deep-Learning-On-Disassembly.pdf.

12. Sean Park. (2016). “Fighting Metamorphism using Deep Neural Network with Fourier.” Last accessed on 28 January 2019 at https://ruxcon.org.au/assets/2016/slides/Fighting%20Metamorphism%20using%20Deep%20Learning%20with%20Fourier%20v1.4.pdf.

13. Ilsun You and Kangbin Yim. “Malware Obfuscation Techniques: A Brief Survey.” In *2010 International Conference on Broadband, Wireless Computing, Communication and Application*, pages 297-300, 4 November 2010.

14. Andreas Moser, Christopher Kruegel, and Engin Kirda. “Limits of Static Analysis for Malware Detection.” In *Limits of Static Analysis for Malware Detection*, pages 421-430, 10 December 2007.

15. Chris Eagle. (2011). The IDA Pro Book: The Unofficial Guide to the World’s Most Popular Disassembler. United States, CA: No Starch Press.

16. Ilsun You and Kangbin Yim. “Malware Obfuscation Techniques: A Brief Survey.” In *2010 International Conference on Broadband, Wireless Computing, Communication and Application*, pages 297-300, 4 November 2010.

17. Ian Goodfellow, et al. “Generative Adversarial Nets.” In *Advances in Neural Information Processing Systems 2014*, pages 2672-2680, June 2014.

18. Jonathan Masci, Ueli Meier, Dan Cireşan, Jürgen Schmidhuber. “Stacked Convolutional Auto-encoders for Hierarchical Feature Extraction.” In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52-59, September 2014.

19. Ilsun You and Kangbin Yim. “Malware Obfuscation Techniques: A Brief Survey.” In *2010 International Conference on Broadband, Wireless Computing, Communication and Application*, pages 297-300, 4 November 2010.

20. Yizhe Zhang, et al. “Deconvolutional Paragraph Representation Learning.” In *Advances in Neural Information Processing Systems 2017*, August 2017.

21. Alireza Makhzani, et al. (18 November 2015). *Arxiv*. “Adversarial Autoencoders.” Last accessed on 29 January 2019 at https://arxiv.org/abs/1511.05644.

22. Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. (30 May 2017). *Arxiv*. “The Numerics of GANs.” Last accessed on 1 February 2019 at https://arxiv.org/abs/1705.10461.

23. Ibid.

24. Ferenc Huszár. (5 October 2017). *Inference*. “GANs are Broken in More than One Way: The Numerics of GANs.” Last accessed on 1 February 2019 at http://www.inference.vc/my-notes-on-the-numerics-of-gans/.

25. Diederik Kingma and Jimmy Ba. (22 December 2014). *Arxiv*. “Adam: A Method for Stochastic Optimization.” Last accessed on 1 February 2019 at https://arxiv.org/abs/1412.6980.

26. Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. (n.d.) *Computer Science University of Toronto*. “Neural Networks for Machine Learning Lecture 6” Last accessed on 1 February 2019 at http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

27. Ruslan Salakhutdinov and Geoffrey Hinton. “Semantic Hashing.” In *International Journal of Approximate Reasoning*, pages 969-978, 7 July 2009.

28. Leland McInnes and John Healy. “Accelerated Hierarchical Density Based Clustering.” In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33-42, 1 November 2017.

29. Abdurrahman Pektaş, Mehmet Eriş, and Tankut Acarman. (2011). The Fifth International Conference on Emerging Security Information, Systems and Technologies. “Proposal of n-gram based algorithm for malware classification.” Last accessed on 16 January 2019 at https://www.thinkmind.org/download.php?articleid=securware_2011_1_30_30099.

30. Google Inc. *Virus Total*. Last accessed on 1 February 2019 at https://virustotal.com.

31. Razvan Pascanu, Jack Stokes, Hermineh Sanossian, Mady Marinescu, Anil Thomas. “Malware classification with recurrent networks.” In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1916-1920, 19 April 2015.

TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threats techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com

TREND MICRO™ research