Adversarial Variational Modality Reconstruction and Regularization for Zero-Day Malware Variants Similarity Detection

Christopher Molloy*, Jeremy Banks*, Steven H. H. Ding*, Philippe Charland†, Andrew Walenstein‡, and Litao Li*
*School of Computing, Queen's University, Kingston, Canada.
Emails: {chris.molloy, steven.ding, jeremy.banks, litao.li}@queensu.ca
†Mission Critical Cyber Security Section, Defence R&D Canada - Valcartier, Quebec, QC, Canada.
Email: philippe.charland@drdc-rddc.gc.ca
‡BlackBerry Limited, Waterloo, ON, Canada.
Email: awalenstein@blackberry.com

***Abstract*—Matching malware variants in the same malware family has always been a significant challenge for Cyber Threat Intelligence (CTI). For zero-day malware that does not belong to an existing family, a timely matching of its variants is essential for effective threat tracing and prompt response to the cyber incident. However, malware variants are of diverse forms that make them difficult to match. Additionally, the information extracted from a given malware sample is inaccurate, especially on zero-day malware. Existing malware solutions only focus on detecting known malware or find if two samples are similar without creating any reusable representation of the samples. In this paper, we propose the first practical and efficient solution for zero-day malware variant matching with reconstruction. By combining multi-modality learning and a Siamese-based structure, our model can navigate across different modalities and match zero-day variants. To address the missing or noisy modality issue, we propose a Conditional Variable Autoencoder with a Generative Adversarial Network for heightened resolution. We trained the model on 100,000 malware triplet pairs. Our experiments on real-world noisy samples show that the model out-performs the state-of-the-art and can accurately match not only zero-day malware, but also out-of-sample benign binaries of the same category.**

***Index Terms*—Cyber-security, Deep Learning, Mining heterogeneous data.**

I. Introduction

The cyber world has seen a flood of exponentially increasing malware attacks. According to the AV-TEST Institute, there were 90.77 million new malware samples observed on Windows-based systems in 2020[1]. Many of these malware samples are unique, as variants to existing malware families or new malware strains. Matching and finding similar variants of an existing malware family and grouping malware variants from a new (zero-day) malware family have become increasingly critical for a prompt cyber malware incident response and effective threat tracing.

Malware samples are grouped into families based on their observed behaviors and revealed functionalities, and similarity-based solutions are robust and have the potential to handle unknown new families without retraining. However, existing state-of-the-art solutions often overfits into the known families due to limited information used, resulting in low performance in unseen families [1]. Additionally, all the existing solutions in this direction requires pair-wise comparisons for each incoming sample, which is not scalable [1]–[3].

To address these limitations, MA This ensures the searching process can scale with the amount of indexed malware. Rather than looking at one set of features, our network utilizes multi-modality learning to diversify the input information. To address the noise introduced by malware anti-analysis techniques such as custom packing and code obfuscation, we assume modality inaccuracy and incompleteness. We propose a modality reconstruction mechanism leveraging conditional variable autoencoder and generative minmax learning to improve the model robustness over unseen families. This reconstruction mechanism allows our system to learn from malware that is corrupt or has been modified to evade detection. Our system has shown to successfully classify zero-day variants of malware families unknown to our network. It takes on average only 10 seconds for information extraction and 1 millisecond (in batch mode) for inference. The main contributions of this paper are as follows:

- We combine multimodal learning on binary executables and a Siamese-based neural network architecture as the first practical zero-day malware similarity detection model.
- We propose a conditional variable autoencoder for modality reconstruction and regularisation to handle missing or noisy modality issues in the zero-day malware samples.
- We train the model on real-life malware samples and evaluate it on out-of-sample malware families and benignware under different categories. The experiment shows that our model outperforms current state-of-the-art methods for malware variant similarity analysis, and that the modality

[1]AV-TEST Institute Statistics - Malware Sample Distribution

reconstruction method is robust against the noisy modalities in the testing samples.

II. Heterogeneous Malware Information

Malware Image. With each malware sample being a raw byte sequence, the malware sample can be reshaped into a square image. Each byte is in the range of 0 to 255, which can be interpreted as a gray-scale pixel. The pixels are padded with 0 to the length of the closest square, and are then arranged into a $1:1$ image. The image is down-sampled to the resolution of a by a. The a value was 256 to match [4]. The image modality gives our network a very high-level overview of each malware sample that may otherwise not be possible from the other input modalities.

Malware Image Signature. We use an image-based signature as the second modality. We follow the method proposed by Wong *et al.* [5] to generate a signature derived from relative brightness in regions of the image. We choose a grid size of $b = 10$, and a vector length $c = 4$.

String Information. Variants of the same family may not have the same code appearance due to polymorphism. However, since their high-level functions are the same, string values can aid in malware analysis [6]. We remove a list of common strings including API names, date format strings, and file section titles.

Executable Imports. Binary executable malware samples also leverage system API calls for certain functionalities such as process injection, file encryption, and communications. Having completely different objectives than benignware, the list of APIs that a malware sample imports may contain critical information about the objective of the malware [7].

Assembly Code. Additionally, we model the extracted assembly code from the malware program. We use the open source disassembler Ghidra to extract the assembly code[2]. To reduce file-specific information, address-specific assembly code instructions were removed during the modality extraction process.

III. Multi-Modality Similarity Learning

Each malware modality is passed through a separate portion of the network and collected together at the end. We denote these sections of the network as subnetworks. A random non-zero modality from the output of the subnetworks is chosen to be reconstructed. The modalities are then sent through the reconstruction subnetwork to predict the real values for the noised modality tensor. The Mean Squared Error (MSE) from the reconstruction is also incorporated into the total loss of the network. The output of the network is then flattened, and appended to the output of the reconstruction subnetwork. This embedding is used for similarity analysis to determine the angular loss of the network. An overview of the processes is shown in Figure 1

[2]The Ghidra Disassembler

A. Modality Encoding

All three of the text modalities are separately tokenized into integer vectors. We choose a vocabulary size of $d = 1,000$ for the strings and import modalities. For the assembly code we choose a vocabulary size of $f = 30,000$ and a sequence length of $e = 1,000$ for all three text modalities.

Our network takes the five previously discussed modalities as input and outputs an embedding to be used for angular similarity analysis. Before the denoising and reconstruction portion of our network, it learns from each modality separately. We will denote the image, image signature, strings, imports, and code subnetworks as $I(\mathsf{Y}_{\text{image}}) : \mathbb{R}^{a\times a} \to \mathbb{R}^g$, $S(\mathsf{Y}_{\text{signature}}) : \mathbb{Z}^{b^2*c} \to \mathbb{R}^g$, $T(\mathsf{Y}_{\text{strings}}) : \mathbb{R}^e \to \mathbb{R}^g$, $P(_{\text{imports}}) : \mathbb{R}^e \to \mathbb{R}^g$, and $E(_{\text{code}}) : \mathbb{R}^e \to \mathbb{R}^g$ respectively, where Y is the incoming malware sample. The output of the subnetworks are then collected in a stack, $X \in \mathbb{R}^{5\times g}$. This stack is used for reconstruction, and later is a part of the final output of the network. We choose the output dimension of each subnetwork to be $g = 32$.

Our image processing subnetwork $I(\mathsf{Y}_{\text{image}})$ is defined as follows. We will let $C_r : \mathbb{R}^{a\times a} \to \mathbb{R}^{h\times h\times i}$ and $C_s : \mathbb{R}^{a\times a} \to \mathbb{R}^{h\times h\times i}$ denote two-dimensional convolution functions.The two convolution results are then multiplied to create a gate. This matrix is then reduced by $P : \mathbb{R}^{h\times h\times i} \to \mathbb{R}^q$, a max pooling function. The resulting matrix is then flattened to a single real valued vector of length q and sent through a linear transformation $I(\mathsf{Y}_{\text{image}}) = W^{\text{image}}P(C_r \times C_s) + b^{\text{image}}$. W^{image} and b^{image} are trainable parameters. We choose a convolution output dimension of $h = 253$ and $i = 8$. We choose the pooling reduction to be done on an $r = 16$ sized grid, thus the output of the pooling and flattening is a vector of length $q = 1800$.

Next we describe the image signature subnetwork, $S(\mathsf{Y}_{\text{signature}})$. First, our signature is mapped from its integer values to a real valued representation by mapping function $\zeta^{\text{signature}} : \mathbb{Z}^{b^2\times c} \to -1 < \mathbb{R}^{b^2\times c} < 1$. This real mapping allows for a better representation of the signature. This mapping is then sent through a linear transformation to find the embedding of the image signature. The embedding calculation is defined as $S(\mathsf{Y}_{\text{signature}}) = W^{\text{signature}}\zeta^{\text{signature}}(\mathsf{Y}_{\text{signature}}) + b^{\text{signature}}$. $W^{\text{signature}}$ and $b^{\text{signature}}$ are trainable parameters. These parameters are trained though a perceptron in our network.

The three text modalities all pass through subnetworks with the same structure, so we describe the general text modality subnetwork. Initially, the text modalities are mapped from a 1-dimensional integer vector to a r dimensional real valued vector. This mapping is done by $\zeta^{\text{text}} : \mathbb{Z}^e \to -1 < \mathbb{R}^{e\times r} < 1$. We choose the expanded dimension for the mapping $r = 16$. This mapping is then transformed to the subnetwork embedding by a Gated Recurrent Unit (GRU). A recurrent unit was chosen for processing the text modalities due to the importance of the sequence at which these modalities appear in malware samples. The output of the GRU is a vector of length g that is the embedding of the text subnetwork. The calculation for the GRU output is found in [8], and we denote it as s_g^{text}. The

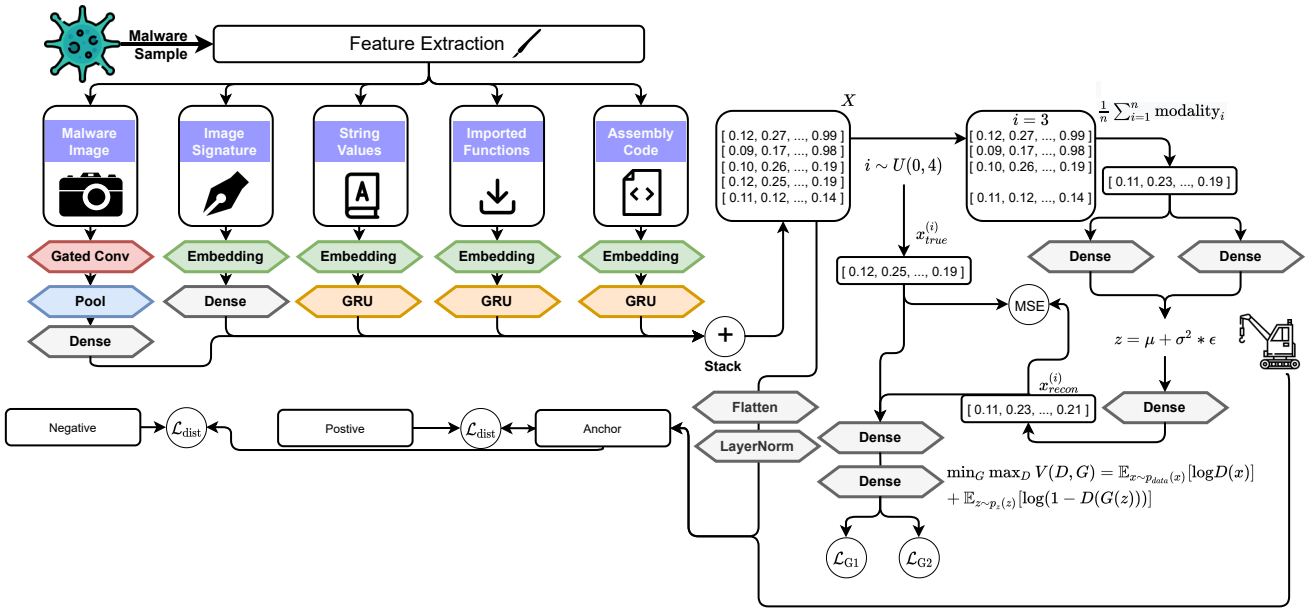

Fig. 1: After feature extraction the five modalities go through their respective subnetwork. The subnetwork outputs are grouped and a random modality is chosen for reconstruction. The reconstruction is done by a Conditional Variable Autoencoder subnetwork that is aided by a Generative Adversarial subnetwork. The result of the reconstruction is used to calculate the MSE for further learning. All five modalities, as well as the five reconstructed modalities are collected together to create the embedding of the sample

final outputs of the three text inputs are: $T(Y_{\text{strings}}) = s_g^{\text{strings}}$, $P(Y_{\text{imports}}) = s_g^{\text{imports}}$, and $E(Y_{\text{code}}) = s_g^{\text{code}}$.

B. Conditional Cross-Modality Reconstruction

In real-world malware triage environments, it is common for the features extracted from incoming samples to be missing or have some noise. To address this issue, our network has been implemented with a Conditional Variable Autoencoder (C-VAE) for modality reconstruction [9]. A C-VAE is an extension of a typical variable autoencoder. Variable autoencoders are systems for generating encoded data from some latent space. Autoencoders are systems that take some input x from the space $\mathbb{R}^n$ and map it to a latent space with dimensionality $\mathbb{R}^m$ where $m < n$. This mapping in latent space can then be decoded back to the original data with error. Whereas a traditional autoencoder encoding mapping to the latent space is deterministic, variable autoencoders will map a distribution based on the incoming sample to some latent space. This difference in mapping allows variable autoencoders to be more robust against overfitting, as well as perform well against noisy or unseen data. This success against previously unseen data is of great use to the domain of malware modality reconstruction due to newly emerging variants and malware families. The initial assumption of the C-VAE is that our input values x have been generated from an unknown continuous random variable β, and that there exists a posterior distribution for $\beta, q(\beta|x) = N(\mu, \sigma)$. Where μ and σ are some mean and standard deviation. Due to the reparameterization trick, we are able to express β through a deterministic variable $\beta = g_\phi(\epsilon, x)$ where ϵ is some noise with a standard normal distribution, and ϕ are the parameters of g. Since we are in the Gaussian case for β, the transformation of β can be reparametrized as $g_\phi(\epsilon, x) = \mu + \sigma * \epsilon$, where $\epsilon \sim \text{N}(0, 1)$.

In the case of neural networks, differentiable multi-layer perceptron's (MLP) are used for approximating the probabilistic encoder $q_\phi(\beta|x)$, which is the posterior to the generative model $p_\theta(x, \beta)$. Here, θ and ϕ are optimized through gradient descent. With the assumption that β is Gaussian from above, we know the posterior has a Gaussian distribution with an approximately Gaussian covariance matrix [9]. The mean and variance are approximated through MLPs. Through the transformation described above, we estimate $z^{(i,l)}$ [9]. A third MLP estimates the input value $x^{(i)}$ based on the estimation function $g_\phi(\epsilon^{(l)}, x^{(i)})$

The network design of our C-VAE is as follows. Once a sample has been parsed through the modality subnetworks, it is collected as a matrix X. The input to the C-VAE is X. One of the five output vectors is randomly chosen to be reconstructed. We will denote this chosen modality vector as $x^{(i)}_{true}$. The four other modalities are then averaged together to create a single one-dimensional vector of length g. The condition of the specific training cycle is then appended to the end averaged vector. This condition is a one dimensional one-hot vector describing which modality is to be reconstructed. The length of the one-dimensional vector is now $g + 5$. We denote this vector as ψ. The Normal distribution of ψ is then approximated by two MLPs. The first MLP is trained to approximate the mean of the distribution, and the second is trained to

approximate the variance. These two linear transformations are $\mu = W^{\text{mean}}\psi + b^{\text{mean}}$, and $\sigma^2 = W^{\text{variance}}\psi + b^{\text{variance}}$. Where $W^{\text{mean}}, b^{\text{mean}}, W^{\text{variance}}$, and b^{variance} are trainable parameters. Initially, to reduce the possibility of an exploding gradient, W^{variance} and b^{variance} are initialized to 0. The latent space of the encoder, γ sets the dimensionality of the MLPs, so $\mu, \sigma^2 \in \mathbb{R}^{\gamma}$. For our network we choose a latent space of $\gamma = 4$. $\beta = \mu + \sigma^2 * \epsilon$ is the latent space calculation where μ and σ^2 are the approximated distributions mean and variance, and ϵ is an error vector generated from the standard normal distribution.

The latent space vector β is decoded through an MLP. The ouptut of the decoder, which is the final output of the C-VAE, $x^{(i)}_{recon} \in \mathbb{R}^{32}$, is $x^{(i)}_{\text{recon}} = W^{\text{decode}}z + b^{\text{decode}}$. Where W^{decode} and b^{decode} are trainable parameters of the decoder MLP. The reconstruction of all modalities is then appended with the other modalities to the end of X. This stack of 10 modalities is flattened, and the final output of the network. This embedding can be used for similarity analysis and stored as a representation of a malware sample. This embedding is a single real valued vector with length 320.

We will now describe how we find the loss of the C-VAE. The loss for the C-VAE is shown in Equation 1 where μ_i and σ_i^2 are each of the means and variances in the latent space.

$$\mathcal{L}_{\text{C-VAE}} = -\frac{1}{2}\sum_{i=1}^{\gamma}(1 + \sigma_i^2 - \mu_i^2 - \sigma_i^2) \tag{1}$$

On top of this, the MSE is also calculated between the true modality and the reconstructed modality. The MSE is calculated as $\text{MSE} = (x^{(i)}_{\text{recon}} - x^{(i)}_{\text{true}})^2$.

C. Reconstruction through Minmax Learning

We have also implemented a Generative Adversarial Network (GAN) for assisting reconstruction [10]. The purpose of the GAN is to heighten the resolution of the modalities reconstructed by the C-VAE process. This is done by predicting whether an incoming modality is reconstructed or original. For describing our GAN, we define our reconstruction subnetwork as $G(\beta)$ where the output is a reconstructed modality vector based on the latent space vector β. We will now define a second subnetwork $D(x)$ where the output is a single scalar representing the probability that the input value x is an original embedding and not a reconstructed embedding. D is trained on correctly assigning the reconstructed label to the reconstructed data. Along with the original loss propagated through $G(\beta)$, $G(\beta)$ is also trained to minimize the success of $D(x)$ as $\log(1 - D(G(\beta))$.These two trainings can be thought of as a game between two parties. We describe this minimax game with value function $V(G, D)$ in Equation 2.

$$\begin{aligned}\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] \\ +\mathbb{E}_{\beta\sim p_\beta(\beta)}[\log(1 - D(G(\beta)))]\end{aligned} \tag{2}$$

In terms of neural networks, the input to the GAN is either $x^{(i)}_{\text{true}}$ or $x^{(i)}_{\text{recon}}$ and the GAN is tasked with classifying the incoming vector as either original or reconstructed. The subnetwork $GAN : \mathbb{R}^g \rightarrow 0 \leq \mathbb{R} \leq$ is two MLPs with trainable parameters.

The GAN subnetwork is trained on the gradient generated by binary cross entropy loss of classifying $x^{(i)}_{\text{true}}$ and $x^{(i)}_{\text{recon}}$ as either true or reconstructed, $\mathcal{L}_{\text{G1}} = -0.01 * \log(\text{GAN}(x^{(i)}_{\text{recon}}))$. The opposite of this loss is propagated through the rest of the network as $\mathcal{L}_{\text{G2}}$. The reconstruction losses are then totaled together for adjusting the network after each training cycle. This total loss is denoted as $\mathcal{L}_{\text{recon}}$ and is calculated in Equation 3.

$$\mathcal{L}_{\text{recon}}(X) = \mathcal{L}_{\text{C-VAE}} + \mathcal{L}_{\text{G2}} + 0.01 * \text{MSE} \tag{3}$$

Through imperical testing, it was found that multipling the MSE by the scalar 0.01 leads to the best network results. $\mathcal{L}_{\text{recon}}$ is then added into the total loss of the network after each training cycle.

D. Angular Loss with Cross-Entropy

Our Siamese network is trained with the triplet loss method [11]. For our Siamese network, we have designed an angular similarity loss technique. This loss is derived from the cosine similarity between the three samples. We will denote each anchor, positive, and negative trained through the network as X_a, X_p, and X_n. First, the angular distance between the anchor and the positive, and the angular distance between the anchor and the negative are calculated. The calculation for angular distance is shown in Equation 4.

$$D(\mathrm{u}, \mathrm{v}) = \arccos\left(\frac{\mathrm{u}\cdot\mathrm{v}}{\|\mathrm{u}\|\,\|\mathrm{v}\|}\right)/\pi \pm \epsilon \tag{4}$$

Where ϵ is a very small real constant added to our angular distance to ensure differentiation. By the range of arccos, the distance function is bounded between $(0 + \epsilon)$ and $(1 - \epsilon)$. The ideal result for our network is if there is no distance between our anchor and positive, and the maximal distance of $(1 - 2\epsilon)$ between our anchor and negative. The goal of our loss is to maximize the difference between the negative and positive distance ($D_{\text{negative}} - D_{\text{positive}}$). We have defined $\mathcal{L}_{dist}$ to be a minimizing function that maximizes the previously discussed distance. $\mathcal{L}_{dist}$ is defined in Equation 5.

$$\begin{aligned}\mathcal{L}_{\text{dist}} = -\frac{1}{n}\sum_{i=1}^{n}\log([D(X_{a_i}, X_{n_i}) - D(X_{a_i}, X_{p_i})]) \\ -\frac{1}{n}\sum_{i=1}^{n}\log(D(X_{a_i}, X_{n_i})) + \frac{1}{n}\sum_{i=1}^{n}\log(D(X_{a_i}, X_{p_i}))\end{aligned} \tag{5}$$

We utilize log loss for our distance loss equation to speed up network convergence. First, we find the loss in the total distance between the negative and positive distance calculations. We then find the loss between the negative distance and 0, and between the positive distance and 1. The second two calculations are added to our loss to ensure that the movement of distances is equally weighted in our loss calculation.

Finally, the total loss of the network is calculated from the reconstruction loss and the triplet loss. For malware

Model	AUC	Accuracy	F1 Score	Precision	Recall	μ Positive Similarity $\pm$ σ^2	μ Negative Similarity $\pm$ σ^2	Optimal Threshold
Image	0.868	0.798	0.813	0.757	0.877	0.441±0.044	0.153±0.026	0.194
Image signature	0.785	0.720	0.773	0.650	**0.952**	0.468±0.039	0.244±0.043	0.157
Strings	0.831	0.770	0.791	0.724	0.873	0.465±0.047	0.190±0.041	0.200
Imports	0.738	0.718	0.756	0.666	0.874	0.356±0.042	0.199±0.053	0.111
Code	0.870	0.786	0.800	0.755	0.850	0.467±0.072	0.129±0.031	0.031
Hsiao et al. [1]	0.668	0.704	0.751	0.647	0.896	0.599±0.074	0.373±0.164	0.162
Zhu et al. [3] (single loss)	0.878	0.810	0.797	0.856	0.747	0.724±0.096	0.245±0.044	0.531
Zhu et al. [3] (dual loss)	0.881	0.832	0.815	**0.904**	0.742	0.701±0.100	0.209±0.035	0.521
Khandhar [2]	0.875	0.790	0.791	0.788	0.795	0.545±0.035	0.274±0.019	0.378
Viz*	**0.917**	**0.835**	**0.843**	0.803	0.888	0.475±0.044	0.127±0.024	0.192

TABLE I: Results of different models on the zero-day set. (* indicates proposed model)

Model	AUC	Accuracy	F1 Score	Precision	Recall	μ Positive Similarity $\pm$ σ^2	μ Negative Similarity $\pm$ σ^2	Optimal Threshold
Image	0.888	0.834	0.857	0.768	**0.971**	0.477±0.048	0.141±0.031	0.151
Image signature	0.835	0.796	0.822	0.728	0.945	0.360±0.025	0.171±0.031	0.151
Strings	0.857	0.806	0.807	0.801	0.814	0.389±0.035	0.147±0.033	0.189
Imports	0.782	0.778	0.810	0.707	0.948	0.439±0.035	0.214±0.066	0.154
Code	0.865	0.821	0.838	0.762	0.931	0.454±0.076	0.130±0.043	0.071
Hsiao et al. [1]	0.560	0.628	0.634	0.624	0.645	0.485±0.082	0.420±0.155	0.490
Zhu et al. [3] (single loss)	0.827	0.824	0.805	0.897	0.731	0.792±0.087	0.380±0.041	0.690
Zhu et al. [3] (dual loss)	0.859	0.794	0.798	0.783	0.815	0.745±0.081	0.337±0.043	0.451
Khandhar [2]	0.863	0.795	0.811	0.751	0.883	0.525±0.019	0.327±0.015	0.375
Viz*	**0.958**	**0.898**	**0.895**	**0.923**	0.869	0.292±0.025	0.050±0.003	0.136

TABLE II: Results of different models on the benign set. (* indicates the proposed model)

embeddings X_a, X_p, and X_n, the total loss of the network is calculated in Equation 6.

$$\mathcal{L}_{\text{total}} = \frac{\mathcal{L}_{\text{recon}}(X_a) + \mathcal{L}_{\text{recon}}(X_p) + \mathcal{L}_{\text{recon}}(X_n)}{3} + \mathcal{L}_{\text{distance}} \tag{6}$$

The loss $\mathcal{L}_{total}$ is then averaged by the batch and propagated back into the model. In total, our model has 597,605 trainable parameters. For model training, we used the Adam optimization algorithm with a learning rate of 0.0001.

IV. Experiments

A. Experiment Setup

There were two experiments that were conducted on our proposed network. These experiments were to evaluate the similarity of zero-day malware samples and the similarity of benignware samples. The first experiment showed whether our network was able to accurately match variants of malware families that it had never seen before. This simulated a real world environment where incoming malware samples may be variants of families that have never been seen by the network. The second experiment evaluated the similarity analysis performance against benignware, further evaluating the out of sample similarity detection ability of the network. No families that have been tested on the network were part of families during the training process, so the experiments were zero-shot testing.

Three datasets were needed to conduct the two experiments. The first dataset was a training set. It was comprised of 100,000 triplet pairs sampled from 20 common malware families available from MalwareBazaar[3]. The second dataset was the zero-day set. It was a set of 10,000 triplet pairs of malware that were all variants of families not used in the training set. An extra 20,000 triplet pairs from the training families were used as a validation set.

There were 9 benchmarks that were compared against our network in the two experiments. The first five benchmarks were the separate subnetworks of our network. This was performed as an ablation test to show how the performance of our networks is raised from multi-modality learning and show which modalities give our network more insight towards the best embedding of the malware. These networks do not perform reconstruction. We denoted each of these benchmarks by the single modality used in our results. As well, we evaluated our model against four state-of-the-art malware similarity networks proposed by Khandhar, Hsiao *et al*. and Zhu *et al*. [1]–[3]. Each of these four benchmarks are denoted by their authors. For our experiments, we denoted our proposed network as Viz. All of the networks discussed were trained for 10 epochs. A single epoch of training takes 1256 seconds (0.0126 seconds per triplet pair). Evaluating data took 0.0014 seconds per triplet pair. The computer used for experiments was a Linux machine with 16 cores Xeon Gold 2.3/3.9 GHz, a single RTX6000 with 24GB of VRAM, and 200GB of memory. The metrics used for the experiments were Area Under the ROC Curve (AUC), accuracy, precision, recall, mean positive similarity, mean negative similarity, and F1 Score. As well, an optimal threshold was derived from the ROC curve for determining the prediction labels.

B. Evaluation Results

The results for the zero-day malware experiment can be seen in Table I. The model with the best performance on the zero-day set was Viz with an AUC of 0.917 and an accuracy of 0.835.The results from the benign set experiment can be seen in Table II. The model with the highest performance

[3]A malware samples sharing platform

on the benign set was Viz with an AUC of 0.958, followed by the image subnetwork with an AUC of 0.888. This high performance in the image subnetwork on the benign set may be caused by the overall structure of benign software rarely being changed.

V. Related Work

Malware Similarity Analysis. Malware similarity analysis is the domain of studying methods for finding similarities between malware samples. Similarity analysis has been shown to be effective using many different input spaces, such as API sequence alignments and feature hashing [12]–[14]. One method proposed by Jang *et al*. used hashed feature vectors and co-clustering techniques for malware detection and family classification [13]. Similar to feature hashing, our system generates an embedding that can be used for similarity analysis, but our work does not require the full information on an incoming malware sample. Also, our system utilizes machine learning for the embedding generation.

Siamese Networks for Binary Similarity Analysis. The Siamese network is a neural network architecture originally proposed by Bromley *et al.* for handwritten signature verification [15]. Siamese networks have been further applied to the task of binary code similarity analysis. Stokes *et al*. proposed a siamese network for classifying a small set of malware families based on family specific n-grams [16]. Khandhar, Hsiao *et al*. and Zhu *et al*. have all employed Siamese networks for performing binary similarity analysis in images formed from raw binary content [1]–[3]. As well, none of these systems implemented a triplet loss function. The system proposed by Hsiao *et al*. is the only of the four to attempt to classify zero-day malware samples. Ji *et al*. proposed a system for source-binary code similarity analysis [17]. Siamese networks have also recently been shown to work in similarity analysis of Android binaries [18].

VI. Conclusion

In this study, we present a real-world malware similarity analysis tool that has the ability to accurately match variants of zero-day malware. Our experiments showed that multi-modality learning can greatly improve model performance for malware similarity analysis. Future work involve training similarity analysis systems against adversarial malware.

VII. Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number ALLRP 561035-20], BlackBerry Limited, and Defence R&D Canada.

References

[1] S. Hsiao, D. Kao, Z. Liu, and R. Tso, "Malware image classification using one-shot learning with siamese networks," in *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES-2019, Budapest, Hungary, 4-6 September 2019*, ser. Procedia Computer Science, I. J. Rudas, J. Csirik, C. Toro, J. Botzheim, R. J. Howlett, and L. C. Jain, Eds., vol. 159. Elsevier, 2019.

[2] S. Khandhar, "A few-shot malware classification approach for unknown family recognition using malware feature visualization," 2021.

[3] J. Zhu, J. Jang-Jaccard, and P. A. Watters, "Multi-loss siamese neural network with batch normalization layer for malware detection," *IEEE Access*, vol. 8, 2020.

[4] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *J. Comput. Virol. Hacking Tech.*, vol. 15, no. 1, 2019.

[5] C. Wong, M. W. Bern, and D. Goldberg, "An image signature for any kind of image," in *Proceedings of the 2002 International Conference on Image Processing, ICIP 2002, Rochester, New York, USA, September 22-25, 2002*. IEEE, 2002.

[6] J. Lee, C. Im, and H. Jeong, "A study of malware detection and classification by comparing extracted strings," in *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, ICUIMC 2011, Seoul, Republic of Korea, February 21 - 23, 2011*, S. Lee, L. Hanzo, M. Y. Chung, S. Lee, and K. Cho, Eds. ACM, 2011.

[7] J. Bai, J. Wang, and G. Zou, "A malware detection scheme based on mining format information," *The Scientific World Journal*, vol. 2014, 2014.

[8] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. J. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *CoRR*, vol. abs/1902.06705, 2019.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014.

[11] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015.

[12] J. D. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digit. Investig.*, vol. 3, no. Supplement-1, 2006.

[13] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011.

[14] I. K. Cho, T. Kim, Y. J. Shim, H. Park, B. Choi, and E. G. Im, "Malware similarity analysis using API sequence alignments," *J. Internet Serv. Inf. Secur.*, vol. 4, no. 4, 2014.

[15] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a siamese time delay neural network," in *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan Kaufmann, 1993.

[16] J. W. Stokes, C. Seifert, J. Li, and N. Hejazi, "Detection of prevalent malware families with deep learning," in *2019 IEEE Military Communications Conference, MILCOM 2019, Norfolk, VA, USA, November 12-14, 2019*. IEEE, 2019.

[17] Y. Ji, L. Cui, and H. H. Huang, "Buggraph: Differentiating source-binary code similarity with graph triplet-loss network," in *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, J. Cao, M. H. Au, Z. Lin, and M. Yung, Eds. ACM, 2021.

[18] J. Zhu, J. Jang-Jaccard, A. Singh, P. A. Watters, and S. Camtepe, "Task-aware meta learning-based siamese neural network for classifying obfuscated malware," *CoRR*, vol. abs/2110.13409, 2021.