ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation

Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard Harang, *Sophos*

https://www.usenix.org/conference/usenixsecurity19/presentation/rudd

This paper is included in the Proceedings of the
28th USENIX Security Symposium.
August 14–16, 2019 • Santa Clara, CA, USA
978-1-939133-06-9

ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation

Ethan M. Rudd*, Felipe N. Ducau*, Cody Wild, Konstantin Berlin, and Richard Harang*
Sophos PLC

Abstract

Malware detection is a popular application of Machine Learning for Information Security (ML-Sec), in which an ML classifier is trained to predict whether a given file is malware or benignware. Parameters of this classifier are typically optimized such that outputs from the model over a set of input samples most closely match the samples true malicious/benign (1/0) target labels. However, there are often a number of other sources of contextual metadata for each malware sample, beyond an aggregate malicious/benign label, including multiple labeling sources and malware type information (e.g. *ransomware*, *trojan*, etc.), which we can feed to the classifier as auxiliary prediction targets. In this work, we fit deep neural networks to multiple additional targets derived from metadata in a threat intelligence feed for Portable Executable (PE) malware and benignware, including a multi-source malicious/benign loss, a count loss on multi-source detections, and a semantic malware attribute tag loss. We find that incorporating multiple auxiliary loss terms yields a marked improvement in performance on the main detection task. We also demonstrate that these gains likely stem from a more informed neural network representation and are not due to a regularization artifact of multi-target learning. Our auxiliary loss architecture yields a significant reduction in detection error rate (false negatives) of 42.6% at a false positive rate (FPR) of 10^{-3} when compared to a similar model with only one target, and a decrease of 53.8% at 10^{-5} FPR.

1 Introduction

Machine learning (ML) for computer security (ML-Sec) has proven to be a powerful tool for malware detection. ML models are now integral parts of commercial anti-malware engines and multiple vendors in the industry have dedicated ML-Sec teams. For the malware detection problem, these models are typically tuned to predict a binary label (malicious or benign) using features extracted from sample files. Unlike signature engines, where the aim is to reactively blacklist/whitelist malicious/benign samples that hard-match manually-defined patterns (signatures), ML engines employ numerical optimization on parameters of highly parameterized models that aim to learn more general concepts of *malware* and *benignware*. This allows some degree of proactive detection of previously unseen malware samples that is not typically provided by signature-only engines.

Frequently, malware classification is framed as a binary classification task using a simple binary cross-entropy or two-class softmax loss function. However, there often exist substantial metadata available at training time that contain more information about each input sample than just an aggregate label of whether it is malicious or benign. Such metadata might include malicious/benign labels from multiple sources (e.g., from various security vendors), malware family information, file attributes, temporal information, geographical location information, counts of affected endpoints, and associated tags. In many cases this metadata will not be available when the model is deployed, and so in general it is difficult to include this data as features in the model (although see Vapnik et al. [28, 29] for one approach to doing so with Support Vector Machines).

It is a popular practice in the domain of malware analysis to derive binary malicious/benign labels based on a heuristic combination of multiple detection sources for a given file, and then use these noisy labels for training ML models [9]. However, there is nothing that precludes training a classifier to predict each of these source labels simultaneously optimizing classifier parameters over these predictions + labels. In fact, one might argue intuitively that guiding a model to develop representations capable of predicting multiple targets simultaneously may have a smoothing or regularizing effect conducive to generalization, particularly if the auxiliary targets are related to the main target of interest. These auxiliary targets can be ignored during test time if they are ancillary to the task at hand (and in many cases the extra

∗ Equal contribution.
Contact: `<firstname>.<lastname>@sophos.com`

weights required to produce them can be removed from the model prior to deployment), but nevertheless, there is much reason to believe that forcing the model to fit multiple targets simultaneously can improve performance on the key output of interest. In this work, we take advantage of multi-target learning [2] by exploring the use of metadata from threat intelligence feeds as auxiliary targets for model training.

Research in other domains of applied machine learning supports this intuition [14, 19, 12, 31, 1, 22], however outside of the work of Huang et al. [11], multi-target learning has not been widely applied in anti-malware literature. In this paper, we present a wide-ranging study applying auxiliary loss functions to anti-malware classifiers. In contrast to [11], which studies the addition of a single auxiliary label for a fundamentally different task, i.e., malware family classification – we study both the addition of multiple label sources for the same task and multiple label sources for multiple separate tasks. Also, in contrast to [11], we do not presume the presence of all labels from all sources, and introduce a per-sample weighting scheme on each loss term to accommodate missing labels in the metadata. We further explore the use of multi-objective training as a way to expand the number of trainable samples in cases where the aggregate malicious/benign label is unclear, and where those samples would otherwise be excluded from purely binary training.

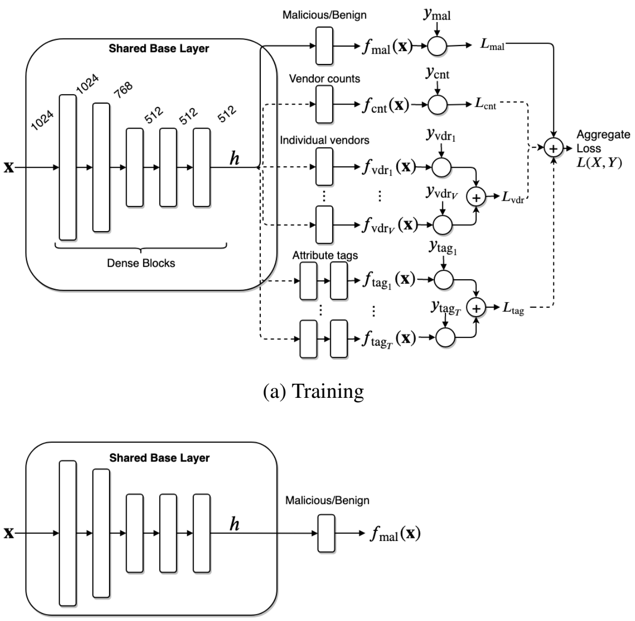
Having established for which loss types and in which contexts auxiliary loss optimization works well, we then explore *why it works well*, via experiments designed to test whether performance gains are a result of a regularization effect from multi-objective training or information from the content of the target labels that the network has learned to correlate.

In summary, this paper makes the following contributions:

- A demonstration that including auxiliary losses yields improved performance on the main task. When all of our auxiliary loss terms are included, we see a reduction of 53.8% in detection error (false negative) rate at 10^{-5} false positive rate (FPR) and a 42.6% reduction in detection error rate at 10^{-3} FPR compared to our baseline model. We also see a consistently better and lower-variance ROC curve across all false positive rates.

- A breakdown of performance improvements from different auxiliary loss types. We find that an auxiliary Poisson loss on detection counts tends to yield improved detection rates at higher FPR areas ($\geq 10^{-3}$) of the ROC curve, while multiple binary auxiliary losses tend to yield improved detection performance in lower FPR areas of the ROC curve ($< 10^{-3}$). When combined we see a net improvement across the entire ROC curve over using any single auxiliary loss type.

- An investigation into the mechanism by which multi-objective optimization yields enhanced performance, including experiments to assess possible regularization effects.



(a) Training

(b) Deployment

Figure 1: Schematic overview of our neural network architecture. (a) During training multiple output layers with corresponding loss functions are optionally connected to a common base topology consisting of five dense blocks (see Section 3) of sizes 1024, 768, 512, 512, and 512. This base, connected to our main malicious/benign output (solid line in the figure) with a loss on the aggregate label, constitutes our baseline architecture. Auxiliary outputs and their respective losses are represented as dashed lines. The auxiliary losses fall into three types: count loss, multi-label vendor loss, and multi-label attribute tag loss. The formulation of each of these auxiliary loss types is explained in Section 3. (b) At deployment time, these auxiliary outputs are removed and we predict only the main label.

We see our auxiliary loss approach as specifically useful for both endpoint and cloud deployed models in cases when the auxiliary information cannot be directly used as input into the model at prediction time, but can be collected for a training dataset. This could be due to high cost, performance issues, latency concerns, or a multitude of other constraints during prediction time. For example, it is not feasible to scan every new file executed on an endpoint via a threat intelligence feed, because of prohibitive licensing fees, endpoint latency and bandwidth limitations, as well as customer privacy concerns. However, procuring reports from such a feed for large training sets might be feasible offline.

The remainder of this paper is laid out as follows: First, in Section 2 we discuss some of the metadata available for use as auxiliary targets, and feature extraction methods for portable executable (PE) files. We then provide details on how we converted the available metadata into auxiliary targets and losses, as well as how the individual losses are weighted and combined on a per-sample basis in Section 3. We finish that Section with a description of how our dataset was collected and provide summary statistics. In Section 4 we describe our experimental evaluations across a number of combinations of auxiliary targets, and demonstrate the impact of fitting these targets on the performance of the main malware detection task. Section 5 presents discussion of our results, as well as a set of experiments on synthetic targets to explore potential explanations for the observed improvements. Section 6 presents related work and Section 7 concludes.

2 ML-Sec Detection Pipelines: From Single Objective to Multi-Objective

In the following, we describe a simplified ML-Sec pipeline for training a malicious file classifier, and propose a simple extension that allows the use of metadata available during training (but not at test time) and improves performance on the classification task.

ML-Sec detection pipelines use powerful machine learning models that require many labeled malicious and benign samples to train. Data is typically gathered from several sources, including deployed anti-malware products and vendor aggregation services, which run uploaded samples through vendor products and provide reports containing per-vendor detections and metadata. The exact nature of the metadata varies, but typically, malicious and benign scores are provided for each of M individual samples on a per-vendor basis, such that, given V vendors, between 0 and V of them will designate a sample malicious. For a given sample, some vendors may choose not to answer, resulting in a missing label for that vendor. Furthermore, many vendors also provide a detection name (or malware family name) when they issue a detection for a given file. Additional information may also be available, but crucially, the following metadata are presumed present for the models presented in this paper: i) per-vendor labels for each sample, either malicious/benign (mapped to binary $1/0$, respectively) or NULL; ii) textual per-vendor labels on the sample describing the family and variant of the malware (an empty string if benign or NULL); and iii) time at which the sample was first seen.

Using the individual vendor detections, an aggregate label can be derived either by a voting mechanism or by thresholding the net number of vendors that identify a given sample as malicious. The use of aggregated anti-malware vendor detections as a noisy labeling source presumes that the vendor diagnoses are generally accurate. While this is not necessarily a valid assumption, e.g., for novel malware and benignware, this is typically accounted for by using samples and metadata that are slightly dated so that vendors can correct their respective mistakes (e.g., by blacklisting samples in their signature databases).

Each malware/benignware sample must also be converted to a numerical vector to be able to train a classifier, a process called *feature extraction*. In this work we focus on static malware detection, meaning that we assume only access to the binary file, as opposed to dynamic detection, in which the features used predominantly come from the execution of the file. The feature extraction mechanism varies depending on the format type at hand, but consists of some numerical transformation that preserves aggregate and fine-grained information throughout each sample, for example, the feature extraction proposed by Saxe et al. [25] – which we use in this work – uses windowed byte statistics, 2D histograms of delimited string hash vs. length, and histograms of hashes of PE-format specific metadata – e.g., imports from the import address table (IAT).

Given extracted features and derived labels, a classifier is then trained, tuning parameters to minimize misclassification as measured by some loss criterion, which under the constraints of a statistical noise model measures the deviation of predictions from their ground truth. For both neural networks and ensemble methods a logistic sigmoid is commonly used to constrain predictions to a [0,1] range, and a cross-entropy loss between predictions and labels is used as the minimization criterion under a Bernoulli noise model per-label.

While the prior description roughly characterizes ML-Sec pipelines discussed in literature to date, note that much information in the metadata, which is often not used to determine the sample label but *is* correlated to the aggregate classification, is not used in training, e.g., the individual vendor classifications, the combined number of detections across all vendors, and information related to malware type that could be derived from the detection names. In this work, we augment a deep neural network malicious/benign classifier with additional output predictions of vendor counts, individual vendor labels, and/or attribute tags. These separate prediction arms were given their own loss functions which we jointly minimized through backpropagation. The difference between a conventional malware detection pipeline and our model can be visualized by considering Figure 1 in the absence and presence of auxiliary outputs (and their associated losses) connected by the dashed lines. In the next section, we shall explore the precise formulation and implementation of these different loss functions.

3 Implementation Details

In this section we describe our implementation of the experiments sketched above. We first introduce our model immediately below, followed by the various loss functions – denoted by $L_{\text{loss type}}(X,Y)$ for some input features X and targets Y – associated with the various outputs of the model, as well as how the labels Y representing the targets of these outputs are constructed. Finally we discuss how our data set of M samples associated with V vendor targets is collected. We use the same feature representation as well as the same general model class and topology for all experiments. Each portable executable file is converted into a feature vector as described in [25].

The base for our model (see Figure 1) is a feedforward neural network incorporating multiple blocks composed of Dropout [27], a dense layer, batch normalization [13], and an exponential linear unit (ELU) activation [7]. The core of the model contains five such blocks with 1024, 768, 512, 512, and 512 hidden units, respectively. This base topology applies the function $f(\cdot)$ to the input vector to produce an intermediate 512 dimensional representation of the input file $\mathbf{h} = f(\mathbf{x})$. We then append to this model an additional block, consisting of one or more dense layers and activation functions, for each output of the model. We denote the composition of our base topology and our target-specific final dense layers and activations applied to features $\mathbf{x}$ by $f_{\text{target}}(\mathbf{x})$. The output for the main malware/benign prediction task – $f_{\text{mal}}(\mathbf{x})$ – is always present and consists of a single dense layer followed by a sigmoid activation on top of the base shared network that aims to estimate the probability of the input sample being malicious. A network architecture with only this malware/benign output serves as the baseline for our evaluations. To this baseline model we add auxiliary outputs with similar structure as described above: one fully connected layer (two for the *tag* prediction task in Section 3.4) which produces some task-specific number of outputs (a single output, with the exception of the restricted generalized Poisson distribution output, which uses two) and some task-specific activation described in the associated sections below.

Except where noted otherwise, all multi-task losses were produced by computing the sum, across all tasks, of the per-task loss multiplied by a task-specific weight (1.0 for the malware/benign task and 0.1 for all other tasks; see Section 4). Training was standardized at 10 epochs; for all experiments we used a training set of 9 million samples and a test set of approximately 7 million samples. Additional details about the training and test data are reported in Section 3.6. Additionally, we used a validation set of 100,000 samples to ensure that each network had converged to an approximate minimum on validation loss after 10 epochs. All of our models were implemented in Keras [6] and optimized using the Adam optimizer [15] with Keras's default parameters.

3.1 Malware Loss

As explained in Section 2, for the task of predicting if a given binary file, represented by its features $\mathbf{x}^{(i)}$, is malicious or benign we used a binary cross-entropy loss between the malware/benign output of the network $\hat{y}^{(i)} = f_{\text{mal}}(\mathbf{x}^{(i)})$ and the malicious label $y^{(i)}$. This results in the following loss for a dataset with M samples:

$$\begin{aligned} L_{\text{mal}}(X,Y) &= \frac{1}{M}\sum_{i=1}^{M} \ell_{\text{mal}}(f_{\text{mal}}(\mathbf{x}^{(i)}), y^{(i)}) \\ &= -\frac{1}{M}\sum_{i=1}^{M} y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)}). \end{aligned} \tag{1}$$

In this paper, we use a "1-/5+" criterion for labeling a given file as malicious or benign: if a file has one or fewer vendors reporting it as malicious, we label the file as 'benign' and use a weight of 1.0 for the malware loss for that sample. Similarly, if a sample has five or more vendors reporting it as malicious, we label the file as 'malicious' and use a weight of 1.0 for the malware loss for that sample.

3.2 Vendor Count Loss

To more finely distinguish between positive results, we investigate the use of the total number of 'malicious' reports for a given sample from the vendor aggregation service as an additional target; the rationale being that a sample with a higher number of malicious vendor reports should, all things being equal, be more likely to be malicious. In order to properly model this target, we require a suitable noise model for count data. A popular candidate is a Poisson noise model, parameterized by a single parameter μ, which assumes that counts follow a Poisson process, where μ is the mean and variance of the Poisson distribution. The probability of an observation of y counts conditional on μ is

$$P(y|\mu) = \mu^y e^{-\mu}/y!. \tag{2}$$

In our problem, as we expect the mean number of positive results for a given sample to be related to the file itself, we attempt to learn to *estimate* μ conditional on each sample $\mathbf{x}^{(i)}$ in such a way that the likelihood of $y^{(i)}|\mu^{(i)}$ is maximized (or, equivalently, the negative log-likelihood is minimized). Denote the output of the neural network with which we are attempting to estimate the mean count of vendor positives for sample i as $f_{\text{cnt}}(\mathbf{x}^{(i)})$. Note that under a non-distributional loss, this would be denoted by $\hat{y}^{(i)}$, however since we are fitting a parameter of a distribution, and not the sample label y directly, we use different notation in this section. By taking some appropriate activation function $a(\cdot)$ that maps $f_{\text{cnt}}(\mathbf{x}^{(i)})$ to the non-negative real numbers, we can write $\mu^{(i)} = a\left(f_{\text{cnt}}(\mathbf{x}^{(i)})\right)$. Consistent with generalized linear model (GLM) literature [18], we use an exponential activation for a, though one could equally well employ some other transformation with the correct output range, for instance the ReLU function.

Letting $y^{(i)}$ here denote the actual number of vendors that recognized sample $\mathbf{x}^{(i)}$ as malicious, the corresponding negative log-likelihood loss over the dataset is

$$
\begin{aligned}
L_{\mathrm{p}}(X,Y) &= \frac{1}{M}\sum_{i=1}^{M} \ell_{\mathrm{p}}\left(a\left(f_{\mathrm{cnt}}(\mathbf{x}^{(i)})\right), y^{(i)}\right) \\
&= \frac{1}{M}\sum_{i=1}^{M} \mu^{(i)} - y^{(i)}\log(\mu^{(i)}) + \log(y^{(i)}!), \qquad (3)
\end{aligned}
$$

which we will refer to as the *Poisson* or *vendor count* loss. In practice, we ignore the $\log(y^{(i)}!)$ term when minimizing this loss since it does not depend on the parameters of the network.

A Poisson loss is more intuitive for dealing with count data than other common loss functions, even for count data not generated by a Poisson process. This is partly due to the discrete nature of the distribution and partly because the assumption of increased variance with predicted mean is more accurate than a homoscedastic – i.e., constant variance – noise model.

While the assumption of increasing variance with predicted count value seems reasonable, it is very unlikely that vendor counts perfectly follow a Poisson process – where the mean *is* the variance – due to correlations between vendors, which might occur from modeling choice and licensing/OEM between vendor products. The variance might increase at a higher or lower rate than the count and might not even be directly proportional to or increase monotonically with the count. Therefore, we also implemented a Restricted Generalized Poisson distribution [10] – a slightly more intricate noise model that accommodates dispersion in the variance of vendor counts. Given dispersion parameter α, the Restricted Generalized Poisson distribution has a probability mass function (pmf):

$$
P(y|\alpha,\mu) = \left(\frac{\mu}{1+\alpha\mu}\right)^{y} (1+\alpha y)^{y-1} \exp\left(\frac{-\mu(1+\alpha y)}{1+\alpha\mu}\right) / y!. \qquad (4)
$$

When $\alpha = 0$, this reduces to Eq. 2. $\alpha > 0$ accounts for over-dispersion, while $\alpha < 0$ accounts for under-dispersion. Note that in our use case α, like μ, is estimated by the neural network and conditioned on the feature vector, allowing varying dispersion per-sample. Given the density function in Eq. 4, the resultant log-likelihood loss for a dataset with M samples is defined as:

$$
\begin{aligned}
L_{\mathrm{gp}}(X,Y) = -\frac{1}{M}\sum_{i=1}^{M} \Bigg[& y^{(i)}\left(\log \mu^{(i)} - \log(1+\alpha^{(i)}\mu^{(i)})\right) \\
& + (y^{(i)}-1)\log(1+\alpha^{(i)}y^{(i)}) \\
& - \frac{\mu^{(i)}(1+\alpha^{(i)}y^{(i)})}{1+\alpha^{(i)}\mu^{(i)}} + \log(y^{(i)}!) \Bigg], \qquad (5)
\end{aligned}
$$

where $\alpha^{(i)}$ and $\mu^{(i)}$ are obtained as transformed outputs of the neural network in a similar fashion as we obtain $\mu^{(i)}$ for the Poisson loss. In practice, as for the Poisson loss, we dropped the term related to $y!$ since it does not affect the optimization of the network parameters.

Note also that restrictions must be placed on the negative value of the $\alpha^{(i)}$ term to keep the arguments of the logarithm positive. For numerical convenience, we used an exponential activation over the dense layer for our $\alpha^{(i)}$ estimator, which accommodates over-dispersion but not under-dispersion. Results from experiments comparing the use of Poisson and Generalized Poisson auxiliary losses are presented in Section 4.1.

While the Poisson distribution is a widely used model for count data, other discrete probability distributions could also be used to model the count of vendor positive results. During early experimentation we also examined the binomial, geometric, and negative binomial distributions as models for vendor counts, but found that they produced unsatisfactory results and so do not discuss them further.

3.3 Per-Vendor Malware Loss

The aggregation service from which we collected our data sets contains a breakout of individual vendor results per sample. We identified a subset $\mathscr{V} = \{v_1, \ldots, v_V\}$ of 9 vendors that each produced a result for (nearly) every sample in our data. Each vendor result was added as a target in addition to the malware target by adding an extra fully connected layer per vendor followed by a sigmoid activation function to the end of the shared architecture. We employed a binary cross-entropy loss per vendor during training. Note that this differs from the vendor count loss presented above in that each high-coverage vendor is used as an individual binary target, rather than being aggregated into a count. The aggregate *vendors* loss L_{vdr} for the $V = 9$ selected vendors is simply the sum of the individual vendor losses:

$$
\begin{aligned}
L_{\mathrm{vdr}}(X,Y) &= \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{V} \ell_{\mathrm{vdr}}\left(f_{\mathrm{vdr}_j}(\mathbf{x}^{(i)}), y_{v_j}^{(i)}\right) \\
&= -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{V} y_{v_j}^{(i)}\log(\hat{y}_{v_j}^{(i)}) + (1-y_{v_j}^{(i)})\log(1-\hat{y}_{v_j}^{(i)}), \qquad (6)
\end{aligned}
$$

where ℓ_{vdr} is the per-sample binary cross-entropy function and $f_{\mathrm{vdr}_j}(\mathbf{x}^{(i)}) = \hat{y}_{v_j}^{(i)}$ is the output of the network that is trained to predict the label $y_{v_j}^{(i)}$ assigned by vendor j to input sample $\mathbf{x}^{(i)}$.

Results from experiments exploring the use of individual vendor targets in addition to malware label targets are presented in Section 4.2.

3.4 Malicious Tags Loss

In this experiment we attempt exploit information contained in family detection names provided by different vendors in the form of malicious tags. We define each tag as a high level description of the purpose of a given malicious sample. The tags used as auxiliary targets in our experiments are: *flooder*, *downloader*, *dropper*, *ransomware*, *crypto-miner*, *worm*, *adware*, *spyware*, *packed*, *file-infector*, and *installer*.

We create these tags from a parse of individual vendor detection names, using a set of 10 vendors which from our experience provide high quality detection names. Once we have extracted the most common tokens, we filter them to keep only tokens related to well-known malware family names or tokens that could easily be associated with one or more of our tags, for example, the token *xmrig* – even though it is not a malware family – can be recognized as referring to a crypto-currency mining software and therefore can be associated with the *crytpo-miner* tag. We then create a mapping from tokens to tags based on prior knowledge. We label a sample as associated with tag t_i if any of the tokens associated with t_i are present in any of the detection names assigned to the sample by the set of trusted vendors.

Annotating our dataset with these tags, allows us to define the tag prediction task as multi-label binary classification, since zero or more tags from the set of possible tags $\mathscr{T} = \{t_1, \ldots, t_T\}$ can be present at the same time for a given sample. We introduce this prediction task in order to have targets in our loss function that are not not directly related to the number of vendors that recognize the sample as malicious. The vendor counts and the individual vendor labels are closely related with the definition of our main target, i.e. the malicious label, which classifies a sample as malicious if 5 or more vendors identify the sample as malware (see Section 3.1). In the case of the tag targets, this information is not present. For instance, if all the vendors recognize a given sample as coming from the *WannaCry* family in their detection names, the sample will be associated only once with the *ransomware* tag. On the converse, because of our tagging mechanism, if only one vendor considers that a given sample is malicious and classifies it as coming from the *WannaCry* family, the *ransomware* tag will be present (although our malicious label will be 0).

In order to predict these tags, we use a *multi-headed* architecture in which we add two additional layers per tag to the end of the shared base architecture, a fully connected layer of size 512-to-256, followed by a fully connected layer of size 256-to-1, followed by a sigmoid activation function, as shown in Figure 1. Each tag t_j out of the possible $T = 11$ tags has its own loss term computed with binary cross-entropy. Like the per-vendor malware loss, the aggregate tag loss is the sum of the individual tag losses. For the dataset with M samples it becomes:

$$
\begin{aligned}
L_{\text{tag}}(X,Y) &= \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{T}\ell_{\text{tag}}\left(f_{\text{tag}_j}\left(\mathbf{x}^{(i)}\right), y_{t_j}^{(i)}\right) \\
&= -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{T} y_{t_j}^{(i)}\log(\hat{y}_{t_j}^{(i)}) + (1-y_{t_j}^{(i)})\log(1-\hat{y}_{t_j}^{(i)}),
\end{aligned}
\tag{7}
$$

where $y_{t_j}^{(i)}$ indicates if sample i is annotated with tag j, and $\hat{y}_{t_j}^{(i)} = f_{\text{tag}_j}\left(\mathbf{x}^{(i)}\right)$ is the prediction issued by the network for that value.

3.5 Sample Weights

While our multi-objective network has the advantage that multiple labels and loss functions serve as additional sources of information, this introduces an additional complexity: given many (potentially missing) labels for each sample, we cannot rely on having all labels for a large quantity of the samples. Moreover, this problem gets worse as more labels are added. To address this, we incorporated per-sample weights, depending on the presence and absence of each label. For labels that are missing, we assign them to a default value and then set the associated weights to zero in the loss computation so a sample with a missing target label will not add to the loss computation for that target. Though this introduces slight implementation overhead, it allows us to train our network, even in the presence of partially labeled samples (e.g., when a vendor decides not to answer).

3.6 Dataset

We collected two datasets of PE files and associated metadata from a threat intelligence feed: a set for training/validation and a test set. For the training/validation set, we pulled 20M PE files and associated metadata, randomly sub-selecting over a year – from September 6, 2017 to September 6, 2018. For the test set, we pulled files from October 6, 2018 to December 6, 2018. Note also that we indexed files based on unique SHA for first seen time, so every PE in the test set comes temporally after the ones in the training set. We do not use a randomized cross-validation training/test split as is common in other fields, because that would allow the set on which the classifier was trained to contain files "from the future", leading to spuriously optimistic results. The reason for the one month gap between the end of the training/validation set and the start of the test set is to simulate a realistic worst-case deployment scenario where the detection model of interest is updated on a monthly basis. All files used in the following experiments – both malicious and benign – were obtained from the threat intelligence feed.

We then extracted 1024-element feature vectors for all those files using feature type described in [25] and derived an aggregate malicious/benign label using a 1-/5+ criterion as described above. Invalid PE files were discarded.

Of the valid PE files from which we were able to extract features we further subsampled our training dataset to 9,000,000 training samples, with 7,188,150 (79.87%) malicious and 1,587,035 (17.63%) benign. The remaining 224,815 (2.5%) are *gray* samples, without a benign or malicious label, i.e., samples where the total number of vendor detections is between 2 and 4 and thus do not meet our 1-/5+ labeling criterion. Our validation set was also randomly subsampled from the same period as the training data and used to monitor convergence during training. It consisted of 100,000 samples; of these, 17,620 were benign (17.62%), 79,819 were malicious (79.82%) , and 2,561 were gray (2.56%). Our test set exhibited similar statistics, with 7,656,573 total samples, 1,606,787 benign (21.8%), 5,762,064 malicious (78.2%), and 287,722 gray (3.76%). Further statistics for the distribution of vendor counts and tags in our datasets are presented in Appendix A.1.

The ratios of malicious to benign data in our training, test, and validation sets are comparable, with malicious samples more prevalent than benign samples. Note that this class balance differs substantially from a real-world deployment scenario, where malware is rarely seen. Increasing the prevalence of this low-occurrence class when training on unbalanced data sets is commonly done to avoid overfitting [3] (we have also observed this in practice), and using a data set with a higher proportion of malicious samples assuming a sufficient number of benign samples – may lead to a more precise decision boundary, and better overall performance as measured by the full ROC curve. Further, when using our malicious tags loss, a greater diversity in malware can yield a more diverse tag set to learn from during training.

Note that ROC curves, which we use as performance measures in Sections 4 and 5, are independent of class ratio in the test set (unlike accuracy), since false positive rate (FPR) values depend only on the benign data, and true positive rate (TPR) values depend only on malware. We also focus on improvements in detection at the very low FPR of 0.1% or below, where we see the most dramatic improvements, since several publications by anti-virus vendors [25, 30] and our experience suggest that 0.1% or lower is indeed a practical FPR target for most deployment scenarios. Our model outputs can also be easily (without retraining) rescaled to the desired deployment class ratio, based on the provided ROC curve and/or standard calibration methods, e.g., fitting a weighted isotonic regressor on scores from the validation set with each score contribution weighted according to its ground truth label to correct the class balance discrepancy between the validation set and the expected deployment setting, then using that regressor to calibrate scores during test/deployment.

4 Experimental Evaluation

In this section, we apply the auxiliary losses presented in in Section 3, first individually, each loss in conjunction with a main malicious/benign loss, and then simultaneously in one combined model. We then compare to a baseline model, finding that each loss term yields an improvement, either in Receiver Operating Characteristic (ROC) net area under the curve (AUC) or in terms of detection performance at low false positive rates (FPR). We note that none of the auxiliary losses we present below harmed classification relative to the baseline model; at worst, our loss-augmented models had equivalent performance to the baseline model with respect to AUC and low-FPR ROC performance on the aggregate malicious/benign label. Each model used a loss weight of 1.0 on the aggregate malicious/benign loss and 0.1 on each auxiliary loss, i.e. when we add K targets to the main loss, the final loss that gets backpropagated through the model becomes

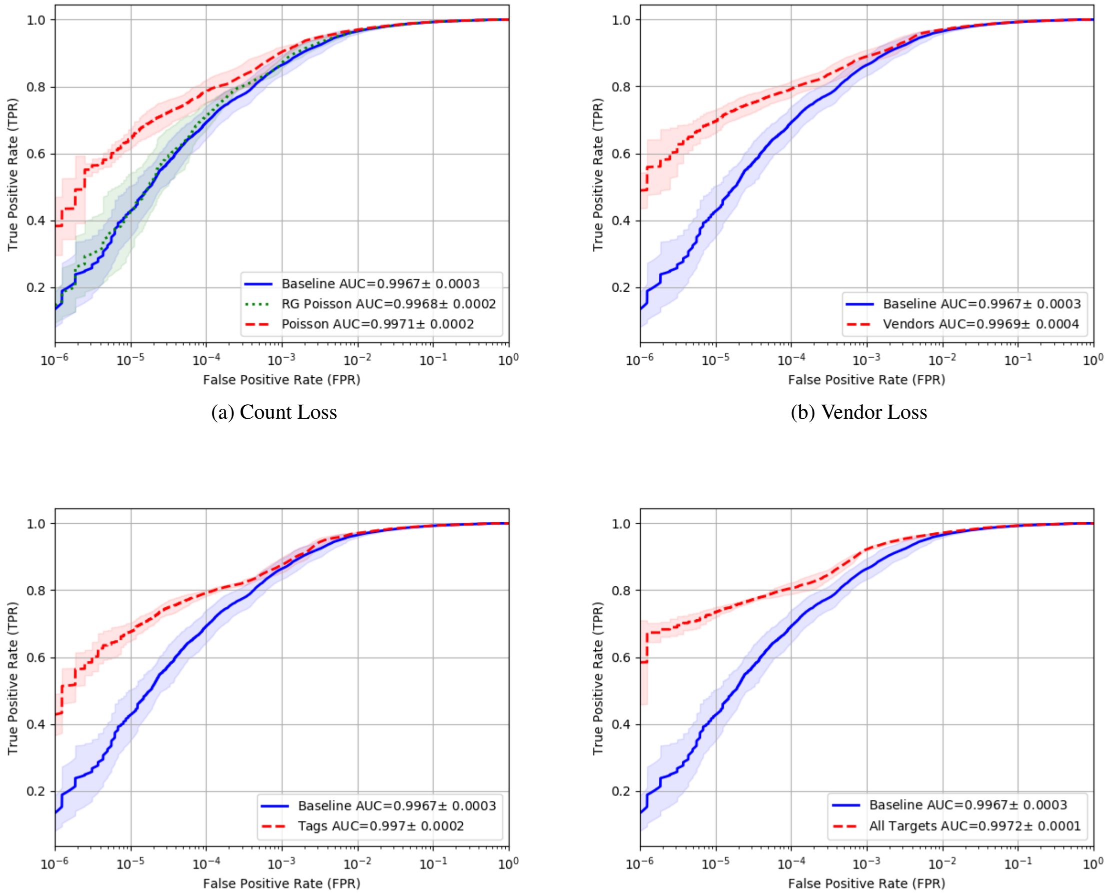
$$L(X,Y) = L_{\mathrm{mal}}(X,Y) + 0.1 \sum_{k=1}^{K} L_k(X,Y). \tag{8}$$

Results are depicted in graphical form in Figure 2 and in tabular form in Table 1.

As the training process for deep neural networks has some degree of intrinsic randomness, which can result in variations in their performance, we report our results in terms of both the mean and standard deviation for the test statistics of interest across five runs. Each model was trained five times, each time with a different random initialization and different randomization over minibatches, and all other parameters (optimizer and learning rate, training data, model structure, number of epochs, etc.) held identical. We compute the test statistic of interest (e.g. the detection rate at a false positive rate of 10^{-3}) for each model, and then compute the average and standard deviation of those results. Notice that the ROC curves in Figure 2 are plotted on a logarithmic scale for visibility, since the baseline performance is already quite high and significant marginal improvements are difficult to discern. For this reason, we also include relative reductions in mean true positive detection error (the rate at which the model fails to detect malware samples – or false negative rate – averaged over the five model results) and in standard deviation from the baseline for our best model in Table 1, and for all models in Table C.1 in the appendix.

4.1 Vendor Count Loss

We employed the same base PE model topology as for our other experiments, with a primary malicious/benign binary cross-entropy loss, and an auxiliary count loss. We experimented with two different loss functions for the count loss – a Poisson loss and a Restricted Generalized Poisson loss



(a) Count Loss

(b) Vendor Loss

(c) Tag Loss

(d) Combined Loss

Figure 2: ROC curves and AUC statistics for count, vendor, and tag experiments compared to our baseline. Lines represent the mean TPR at a given FPR, while shaded regions represent ±1 standard deviation. Statistics were computed over 5 training runs, each with random parameter initialization. (a) *Count loss*. Our baseline model (blue solid line) is shown compared to a model employing a Poisson auxiliary loss (red dashed line), and a dispersed Poisson auxiliary loss (green dotted line). (b) Auxiliary loss on multiple vendors malicious/benign labels (red dashed line) and baseline (blue solid line). (c) Auxiliary loss on semantic attribute tags (red dashed line) and baseline (blue solid line). (d) Our combined model (red dashed line) and baseline (blue solid line). The combined model utilizes an aggregate malicious/benign loss with an auxiliary Poisson count loss, a multi-vendor malicious/benign loss, and a malware attribute tag loss.

(equations 3 and 5 respectively). For the Poisson loss, we used an exponential activation over a dense layer atop the base to estimate $\mu^{(i)}$. For the Restricted Generalized Poisson (RG-Poisson) loss, we followed a similar pattern using two separate dense layers with exponential activations on top; one for the $\mu^{(i)}$ parameter and another for the $\alpha^{(i)}$ parameter. The choice of an exponential activation is consistent with statistics literature on Generalized Linear Models (GLMs) [18].

Results on the malware detection task using Poisson and RG-Poisson losses as an auxiliary loss function are shown in Figure 2a. When compared to a baseline using no auxiliary loss, we see a statistically significant improvement with the Poisson loss function in both AUC and ROC curve, particularly in low false positive rate (FPR) regions. The RG-Poisson loss, by contrast, yields no statistically significant gains over the baseline in terms of AUC, nor does it appear to yield statistically significant gains at any point along the ROC curve.

This suggests that the RG-Poisson loss model is ill-fit, which could stem from a variety of issues. First, if counts are under-dispersed, an over-dispersed Poisson loss could be an inappropriate model. Under-dispersion could occur if certain vendors disproportionately trigger simultaneously or be-

	FPR				
	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
TPR Baseline	0.427 ± 0.076	0.692 ± 0.049	0.864 ± 0.031	0.965 ± 0.007	0.9928 ± 0.0007
TPR Poisson	0.645 ± 0.029	0.785 ± 0.034	0.903 ± 0.016	0.970 ± **0.001**	0.9932 ± **0.0002**
TPR RG Poisson	0.427 ± 0.116	0.711 ± 0.041	0.870 ± 0.016	0.966 ± 0.003	0.9930 ± 0.0003
TPR Vendors	0.697 ± 0.034	0.792 ± 0.024	0.889 ± 0.020	0.970 ± 0.004	0.9928 ± 0.0014
TPR Tags	0.677 ± 0.027	0.792 ± **0.009**	0.875 ± 0.022	0.971 ± 0.004	0.9932 ± 0.0008
TPR All Targets	**0.735** ± **0.014**	**0.806** ± 0.017	**0.922** ± **0.004**	**0.972** ± 0.003	**0.9934** ± 0.0004
% Error Reduction (All Targets)	**53.8%**	**37.0%**	**42.7%**	**20.0%**	**8.3%**
% Variance Reduction (All Targets)	**81.6%**	65.3%	**87.1%**	57.1%	94.3%

Table 1: Top: Mean and standard deviation true positive rates (TPRs) for the different experiments in Section 4 at false positive rates (FPRs) of interest. Results were aggregated over five training runs with different weight initializations and minibatch orderings. Best results consistently occurred when using all auxiliary losses and are shown in bold. Bottom: Percentage reduction in missed true positive detections and percentage reductions in ROC curve standard deviation resulting from the best model (All Targets) compared to the baseline across various FPRs. State-of-the-art results are shown in **bold**.

cause counts are inherently bounded by the net number of vendors. Second, a Poisson model, even with added dispersion parameters, is an ill-posed model of count data, but removing the dispersion parameter removes a dimension in the parameter space to over-fit on. Inspecting the dispersion parameters predicted by the RG-Poisson model, we noted that they were relatively large, which supports the latter hypothesis. We also noticed that the RG-Poisson model converged significantly faster than the Poisson model in terms of malware detection loss.

4.2 Modeling Individual Vendor Responses

Incorporating an auxiliary multi-label binary cross-entropy loss across vendors (cf. Section 3.3) in conjunction with the main malicious/benign loss yields a similar increase in the TPR at low FPR regions of the ROC curve (see Figure 2b) to the Poisson experiment. Though we do not see a significant increase in AUC, since the improvement is integrated across an extremely narrow range of FPRs, this improvement in TPR at lower FPRs may still be operationally significant, and does indicate an improvement in the model.

4.3 Incorporating Tags as Targets

In this experiment we extend the architecture of our base network to predict, not only the malware/benign label, but also the set of 11 tags defined in Section 3.4. For this, we add two fully connected layers per tag to the end of the base architecture (see Section 3.4) which serve to identify each tag from the shared representation. Each of these tag-specialized layers predicts a binary output corresponding to presence/absence of the tag and has an associated binary cross-entropy loss that gets added to the other tag losses and the main malicious/benign loss. The overall loss for this experiment is a sum containing one term per tag, weighted by a loss weight of 0.1 (as mentioned at the beginning of this section), and one term for the loss incurred on the main task.

The result of this experiment is represented via the ROC curves of Figure 2c. Similar to section 4.2 we see no statistical difference in the AUC values with respect to the baseline, but we do observe substantial statistical improvement in the predictions of the model in low FPR regions, particularly for FPR values lower than 10^{-3}. Furthermore, we also witness a substantial decrease in the variance of the ROC curve.

4.4 Combined Model

Finally, we extend our model to predict all auxiliary targets in conjunction with the aggregate label, with a net loss term containing a combination of all auxiliary losses used in previous experiments. The final loss function for the experiment is the sum of all the individual losses where the malware/benign loss has a weight of 1.0 while the rest of the losses have a weight of 0.1.

The resulting ROC curve and AUC are shown in Figure 2d. The AUC of 0.9972 ± 0.0001 is the highest obtained across all the experiments conducted in this study. Moreover, in contrast to utilizing any single auxiliary loss, we see a noticeable improvement in the ROC curve not only in very low FPR regions, but also at 10^{-3} FPR. Additionally, variance is consistently lower across a range of low-FPR values for this combined model than for our baseline or any previous models. An exception is near 10^{-6} FPR where measuring variance is an ill-posed problem because even with a test dataset of over 7M samples, detecting or misdetecting even one or two of them can significantly affect detection rate.

In order to account for the effect of gray samples in the evaluation of our detection model, we re-scanned a subset of those at a later point in time, giving the AV community time to update their detection rules, and evaluated the prediction issued by the model. Even though it is naturally harder to determine maliciousness of these samples (otherwise they would not initially have been categorized as gray), we find that our model predicts the correct labels for more than 77% of them. A more in depth analysis of grey samples is deferred to Appendix B.

5 Discussion

In this section, we examine the effects of different types of auxiliary loss functions on main task ROC curve. We then perform a sanity check to establish whether our performance increases result from additional information added to our neural networks by the auxiliary loss functions or are the artifact of some regularization effect.

5.1 Modes of Improvement

Examining the plots in Figure 2, we see three different types of improvement that result from our auxiliary losses:

1. A bump in TPR at low FPR ($< 10^{-3}$).
2. A net increase in AUC and a small bump in performance at higher FPRs ($\geq 10^{-3}$).
3. A reduction in variance.

Improvement 1 is particularly pronounced in the plots due to the logarithmic scale, but it does not substantially contribute to net AUC due to the narrow FPR range. However, this low-FPR part of the ROC is important from an operational perspective when deploying a malware detection model in practice. Substantially higher TPRs at low FPR could allow for novel use cases in an operational scenario. Notice that this effect is more pronounced for auxiliary losses containing multi-objective binary labels (Figs. 2b, 2c, and 2d) than for the Poisson loss, suggesting that it occurs most prominently when employing our multi-objective binary label losses. Let us consider why a multi-objective binary loss might cause such an effect to occur: At low FPRs, we see high thresholds on the detection score from the main output of the network. To surpass this threshold and register as a detection, the main sigmoid output must be very close to 1.0, i.e., very high response. Under a latent correlation with the main output, a high-response hit for an auxiliary target label could also boost the response for the main detector output, while a baseline model without this information might wrongly classify the sample as benign. We hypothesize that improvement 1 occurs from having many objectives simultaneously and thereby increasing chances for a high-response target hit. The loss type may or may not be incidental, which is consistent with its noticeable but less pronounced presence under a single-objective Poisson auxiliary loss (Figure 2a).

Improvement 2 likely stems from improvements in detection rate that we see around 10^{-3} FPR and higher. Notice that these effects are more pronounced in Figs. 2a and 2d, are somewhat noticeable in Figure 2b, and are not noticeable in Figure 2c, consistent with the resultant AUCs. This suggests that the effect occurs most prominently in the presence of an auxiliary count loss. We postulate that this occurs because our aggregate detection label is derived by *thresholding* the net number of vendor detections for each sample but doing so removes a notional view of confidence that a sample is malicious. Alternatively stated, thresholding removes information on the difficulty of classifying a malicious sample or the extent of "maliciousness" that the number of detection counts provides. Bear in mind that some detectors are better at detecting different types of malware than others, so more detections suggest a *more malicious* file, e.g., with more malware components, or a more widely blacklisted file (higher confidence). Providing information on the number of counts in an auxiliary loss function may therefore provide the classifier more principled information on how to order detection scores, thus allowing for more effective thresholding and a better ROC curve.

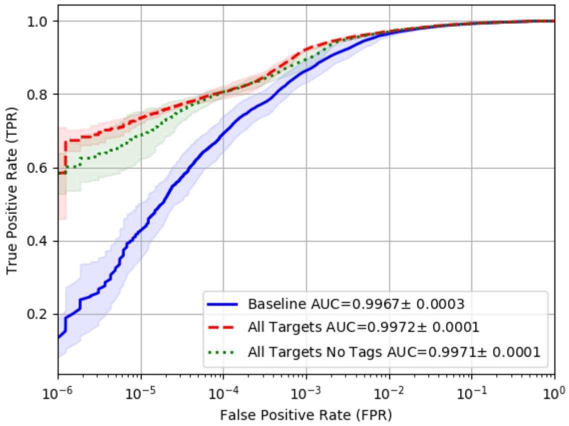


Figure 3: When we remove the attribute tags loss (green dotted line) we get a similar shaped ROC curve with similar ROC compared to using all losses (red dashed line), but with slightly higher variance in the ROC. This supports our hypotheses about effects of different loss functions on the shape of the ROC curve. The baseline is shown as a blue solid line for comparison.

Improvement 3 occurs across all loss types, particularly in low FPR ranges, with the exception of *very low* FPRs (e.g., 10^{-6}), where accurately measuring mean and variance is an ill-posed problem due to the size of the dataset (cf. section 4.4). Comparing the ROC plots in Figure 2, the reduction in variance appears more pronounced as the number of losses increases. Intuitively, this is not a surprising result since adding objectives/tasks imposes constrains the allowable weight space – while many choices of weights might allow a network to perform a single task well only a subset of these choices will work well for all tasks simultaneously.

Thus, assuming equivalent base topology, we expect a network that is able to perform at least as well on multiple tasks as many single-task networks to exhibit lower variance.

Combining all losses seems to accentuate all improvements (1-3) with predictable modes which we attribute to our various loss types (Figure 2d) – higher detection rate at low FPR brought about primarily by multi-objective binary losses, a net AUC increase and a detection bump at 10^{-3} FPR brought about by the count loss, and a reduced variance brought about by many loss functions. To convince ourselves that this is not a coincidence, we also trained a network using only Poisson and vendor auxiliary losses but no attribute tags (cf. Figure 3). As expected, we see that this curve exhibits similar general shape and AUC characteristics that occur when training with all loss terms, but the variance appears slightly increased.

In the variance reduction sense, we can view our auxiliary losses as regularizers. This raises a question: are improvements 1 and 2 actually occurring for the reasons that we hypothesize or are they merely naive result of regularization?

5.2 Representation or Regularization?

While the introduction of some kinds of auxiliary targets appears to improve the model's performance on the main task, it is less clear why this is the case. The reduction in variance produced by the addition of extra targets suggests one potential alternative explanation for the observed improvement: rather than inducing a more discriminative representation in the hidden layers of the network, the additional targets may be acting as constraints on the network, reducing the space of viable weights for the final trained network, and thus acting as a form of additional regularization. Alternatively, the addition of extra targets may simply be accelerating training by amplifying the gradient; while this seems unlikely given our use of a validation set to monitor approximate convergence, we nevertheless also investigate this possibility.

To evaluate these hypotheses, we constructed three additional targets (and associated loss functions) that provided uninformative targets to the model: i) a pseudo-random target that is approximately independent of either the input features or the malware/benign label; ii) an additional copy of the main malware target transformed to act as a regression target; and iii) an extra copy of the main malware target.

The random target approach attempts to directly evaluate whether or not an additional pseudo-random target might improve network performance by 'using up' excess capacity that might otherwise lead to overfitting. We generate pseudo-random labels for each sample based off of the parity of a hash of the file contents. While this value is effectively random and independent of the actual malware/benignware label of the file, the use of a hash value ensures that a given sample will always produce the same pseudo-random target. This target is fit via standard binary cross-entropy loss against a sigmoid output,

$$L_{\text{rnd}}(X,Y) = -\frac{1}{M}\sum_{i=1}^{M} y^{(i)} \log f_{\text{rnd}}(\mathbf{x}^{(i)}) + (1-y^{(i)}) \log\left(1 - f_{\text{rnd}}(\mathbf{x}^{(i)})\right), \quad (9)$$

where $f_{\text{rnd}}\left(\mathbf{x}^{(i)}\right)$ is the output of the network which is being fit to the random target $y^{(i)}$.

In contrast, the duplicated regression target evaluates whether further constraining the weights *without* requiring excess capacity to model additional independent targets has an effect on the performance on the main task. The model is forced to adopt an internal representation that can satisfy two different loss functions for perfectly correlated targets, thus inducing a constraint that does not add additional information. To do this, we convert our binary labels (taking on values of 0 and 1 for benign and malware, respectively) to -10 and 10, and add them as additional *regression* targets fit via mean squared error (MSE). Taking $y^{(i)}$ as the i^{th} binary target and $f_{\text{MSE}}(\mathbf{x}^{(i)})$ as the regression output of the network, we can express the MSE loss as:

$$L_{\text{mse}}(X,Y) = \sum_{i=1}^{M}\left(f_{\text{mse}}\left(\mathbf{x}^{(i)}\right) - 20\left(y^{(i)} - 0.5\right)\right)^2. \quad (10)$$

Finally, in the case of the duplicated target, the model effectively receives a larger gradient due to a duplication of the loss. The loss for this label uses the same cross-entropy loss as for the main target, obtained by substituting $f_{\text{dup}}(\mathbf{x}^{(i)})$ for $f_{\text{mal}}(\mathbf{x}^{(i)})$ in equation 1 as the additional model output that is fit to the duplicated target. Note that we performed two variants of the duplicated target experiment: one in which both the dense layer prior to the main malware target and the dense layer prior to the duplicated target were trainable, and one in which the dense layer for the duplicate target was frozen at its initialization values to avoid the trivial solution where the pre-activation layer for both the main and duplicate target were identical. In both cases, the results were equivalent; only results for the trainable case are shown.

Both f_{rnd} and f_{dup} are obtained by applying a dense layer followed by a sigmoid activation function to the intermediate output of the input sample from the shared base layer (**h** in Figure 1), while $f_{\text{mse}}(\mathbf{x}^{(i)})$ is obtained by passing the intermediate representation of the input sample **h** through a fully connected layer with no output non-linearity.

Results of all three experiments are shown in Figure 4. In no case did the performance of the model on the main task improve statistically significantly over the baseline. This suggests that auxiliary tasks must encode relevant information to improve the model's performance on the main task. For each of the three auxiliary loss types in Figure 4, there is

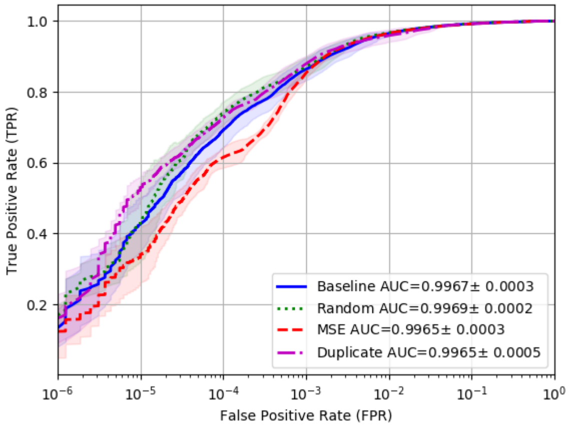


Figure 4: ROC curves comparing classification capabilities of models on the malware target when either random (green dotted line), regression (red dashed line), or duplicated targets (magenta dashed and dotted line) are added as auxiliary losses. Note that with the exception of the regression loss – which appears to harm performance – there is little discernible difference between the remaining ROC curves. The baseline is shown as a blue solid line for comparison.

no additional information provided by the auxiliary targets: the random target is completely uncorrelated from any information in the file (and indeed the final layer is ultimately dominated by the bias weights and produces a constant output of 0.5 regardless of the inputs to the layer), while the duplicated and MSE layers are perfectly correlated with the final target. In either case, there is no incentive for the network to develop a richer representation in layers closer to the input; the final layer alone is sufficient given an adequate representation in the core of the model.

6 Related Work

Applications of ML to computer security date back to the 1990's [21], but large-scale commercial deployments of deep neural networks (DNNs) that have led to transformative performance gains are a more recent phenomenon. Several works from the ML-Sec community have leveraged DNNs for statically detecting malicious content across a variety of different formats and file types [25, 26, 23]. However, these works predominantly focus on applying regularized cross-entropy loss functions between single network outputs and malicious/benign labels per-sample, leaving the potential of multiple-objective optimization largely untapped.

A notable exception, which we build upon in this work, is [11], in which Huang et al. add a multiclass label for Microsoft's malware families to their classification model using a categorical cross-entropy loss function atop a softmax output as an auxiliary objective. They observed that adding targets in this fashion increased performance both on the detection task and on the malware family classification task. Our work builds upon theirs in several respects. First, while their work used 4000-dimensional dynamic features derived from Windows API calls, we extend multi-target approaches to lower-dimensional static features on a larger data set. In this respect, our work pioneers a more scalable approach, but lacks the advantages of dynamic features that their approach provides. Second, we demonstrate that improvements from multi-target learning also occur using far more targets, and we introduce heterogeneous loss functions, i.e., binary cross-entropy and Poisson, whereas their work employs only two categorical cross-entropy losses. Finally, our work introduces loss-weighting to account for potentially missing labels, which may not be problematic for only two targets but become more prevalent with additional targets.

Despite the lack of attention from the ML-Sec community, multi-target learning has been applied to other areas of ML for a long time. The work of Abu-Mostafa [2] predates most explicit references to multi-task learning by introducing the concept of *hints*, in which known invariances of a solution (e.g., translation invariance, invariance under negation) can be incorporated into network structure and used to generate additional training samples by applying the invariant operation to the existing samples, or – most relevant to our work – used as an additional target by enforcing that samples modified by an invariant function should be *both* correctly classified *and* explicitly classified identically. Caruna [5] first introduced multi-task learning in neural networks as a "source of inductive bias" (also reframed as inductive transfer in [4]), in which more difficult tasks could be combined in order to exploit similarities between tasks that could serve as complementary signals during training. While his work predates the general availability of modern GPUs, and thus the models and tasks he examines are fairly simple, Caruna nevertheless demonstrates that jointly learning related tasks produces better generalization on a task-by-task basis than learning them individually. It is interesting to note that in [5] he also demonstrated that learning multiple copies of the same task can also lead to a modest improvement in performance (which we did not observe in this work, possibly due to the larger scale and complexity of our task).

Kumar and Duame [17] consider a refinement on the basic multi-task learning approach that leads to clustering related tasks, in an effort to mitigate the potential of *negative transfer* in which unrelated tasks degrade performance on the target task. Similarly, the work of Rudd et al. [22] explores the use of domain-adaptive weighting of tasks during the training process.

Multi-target learning has been applied to extremely complex image classification tasks, including predicting characters and ngrams within unconstrained images of text [14], joint facial landmark localization and detection [19], image tagging and retrieval [12, 31], and attribute prediction [1, 22] where a common auxiliary task is to challenge the network to classify additional attributes of the image, such as manner of dress for full-body images of people or facial attributes (e.g., smiling, narrow eyes, race, gender). While a range of neural network structures are possible, common exemplars include largely independent networks with a limited number of shared weights (e.g., [1]), a single network with minimal separation between tasks (e.g., [22]), or a number of parallel single-task classifiers in which the weights are constrained to be similar to each other. A more complex approach may be found in [24], in which the sharing between tasks is learned automatically in an online fashion.

Other, more distantly connected domains of ML research reinforce the intuition that learning on disparate tasks can improve model performance. Work in semi-supervised learning, such as [16] and [20], has shown the value of additional reconstruction and denoising tasks to learn representations valuable for a core classification model, both through regularization and through access to a larger dataset than is available with labels. The widespread success of transfer learning is also a testament to the value of training a single model on nominally distinct tasks. BERT [8], a recent example from the Natural Language Processing literature, shows strong performance gains from pre-training a model on masked-word prediction and predictions of whether two sentences appear in sequence, even when the true task of interest is quite distinct (e.g. question answering, translation).

Multi-view learning (see [32] for a survey) is a related approach in which multiple inputs are trained to a single target. This approach also arguably leads to the same general mechanism for improvement: the model is forced to learn relationships between sets of features that improve the performance using any particular set. While this approach often requires all sets of features to be available at test time, there are other approaches, such as [28], that relax this constraint.

7 Conclusion

In this paper, we have demonstrated the effectiveness of auxiliary losses for malware classification. We have also provided experimental evidence which suggests that performance gains result from an improved and more informative representation, not merely a regularization artifact. This is consistent with our observation that improvements occur as additional auxiliary losses and different loss types are added. We also note that different loss types have different effects on the ROC; multi-label vendor and semantic attribute tag losses have greatest effect at low false positive rates ($\leq 10^{-3}$), while Poisson counts have a substantial net impact on AUC, the bulk of which stems from detection boosts at higher FPR.

While we experimented on PE malware in this paper, our auxiliary loss technique could be applied to many other problems in the ML-Sec community, including utilizing a label on format/file type for format-agnostic features (e.g., office document type in [23]) or file type under a given format, for example APKs and JARs both share an underlying ZIP format; a zip-archive malware detector could use tags on the file type for auxiliary targets. Additionally, tags on topics and classifications of embedded URLs could serve as auxiliary targets when classifying emails or websites.

One open question is whether or not multiple auxiliary losses improve each others' performances as well as the main task's. If the multiple outputs of operational interest (such as the tagging output) can be trained simultaneously while also increasing (or at least not decreasing) their joint accuracy, this could lead to models that are both more compact and more accurate than individually deployed ones. In addition to potential accuracy gains, this has significant potential operational benefits, particularly when it comes to model deployment and updates. We defer a more complete evaluation of this question to future work.

While this work has focused on applying auxiliary losses in the context of deep neural networks, there is nothing mathematically that precludes using them in conjunction with a number of other classifier types. Notably, gradient boosted classifier ensembles, which are also popular in the ML-Sec community could take very similar auxiliary loss functions even though the structure of these classifiers is much different. We encourage the ML-Sec research community to implement multi-objective ensemble classifiers and compare with our results. Our choice of deep neural networks for this paper is infrastructural more than anything else; while several deep learning platforms, including PyTorch, Keras, and Tensorflow among others easily support multiple objectives and custom loss functions, popular boosting frameworks such as lightGBM and XGBoost have yet to implement this functionality.

The analyses conducted herein used metadata that can naturally be transformed into a label source and impart additional information to the classifier with no extra data collection burden on behalf of the threat intelligence feed. Moreover, our auxiliary loss technique does not change the underlying feature space representation. Other types of metadata, e.g., the file path of the malicious binary or URLs extracted from within the binary might be more useful in a multi-view context, serving as input to the classifier, but this approach raises challenges associated with missing data that our loss weighting scheme trivially addresses. Perhaps our weighting scheme could even be extended, e.g., by weighting each sample's loss contribution according to certainty/uncertainty in that sample's label, or re-balancing the per-task loss according to the expected frequency of the label in the target distribution. This could open up novel applications, e.g., detectors customized to a particular user endpoints and remove sampling biases inherent to multi-task data.

8 Acknowledgments

This research was sponsored by Sophos PLC. We would additionally like to thank Adarsh Kyadige, Andrew Davis, Hillary Sanders, and Joshua Saxe for their suggestions that greatly improved this manuscript.

References

[1] ABDULNABI, A. H., WANG, G., LU, J., AND JIA, K. Multi-task cnn model for attribute prediction. *IEEE Transactions on Multimedia 17*, 11 (2015), 1949–1959.

[2] ABU-MOSTAFA, Y. S. Learning from hints in neural networks. *J. Complexity 6*, 2 (1990), 192–198.

[3] ANDERSON, H. S., AND ROTH, P. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).

[4] CARUANA, R. A dozen tricks with multitask learning. In *Neural networks: tricks of the trade*. Springer, 1998, pp. 165–191.

[5] CARUNA, R. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference* (1993), pp. 41–48.

[6] CHOLLET, F., ET AL. Keras, 2015.

[7] CLEVERT, D.-A., UNTERTHINER, T., AND HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[8] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[9] DU, P., SUN, Z., CHEN, H., CHO, J.-H., AND XU, S. Statistical Estimation of Malware Detection Metrics in the Absence of Ground Truth. *arXiv e-prints* (Sept. 2018), arXiv:1810.07260.

[10] FAMOYE, F. Restricted generalized poisson regression model. *Communications in Statistics-Theory and Methods 22*, 5 (1993), 1335–1354.

[11] HUANG, W., AND STOKES, J. W. Mtnet: a multitask neural network for dynamic malware classification. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016), Springer, pp. 399–418.

[12] HUANG, Y., WANG, W., AND WANG, L. Unconstrained multimodal multi-label learning. *IEEE Transactions on Multimedia 17*, 11 (2015), 1923–1935.

[13] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[14] JADERBERG, M., SIMONYAN, K., VEDALDI, A., AND ZISSERMAN, A. Deep structured output learning for unconstrained text recognition. *arXiv preprint arXiv:1412.5903* (2014).

[15] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] KINGMA, D. P., MOHAMED, S., REZENDE, D. J., AND WELLING, M. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems* (2014), pp. 3581–3589.

[17] KUMAR, A., AND DAUME III, H. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417* (2012).

[18] MCCULLAGH, P. *Generalized linear models*. Routledge, 2018.

[19] RANJAN, R., PATEL, V. M., AND CHELLAPPA, R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[20] RASMUS, A., BERGLUND, M., HONKALA, M., VALPOLA, H., AND RAIKO, T. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems* (2015), pp. 3546–3554.

[21] RUDD, E., ROZSA, A., GUNTHER, M., AND BOULT, T. A survey of stealth malware: Attacks, mitigation measures, and steps toward autonomous open world solutions. *IEEE Communications Surveys & Tutorials 19*, 2 (2017), 1145–1172.

[22] RUDD, E. M., GÜNTHER, M., AND BOULT, T. E. Moon: A mixed objective optimization network for the recognition of facial attributes. In *European Conference on Computer Vision* (2016), Springer, pp. 19–35.

[23] RUDD, E. M., HARANG, R., AND SAXE, J. Meade: Towards a malicious email attachment detection engine. *arXiv preprint arXiv:1804.08162* (2018).

[24] Ruder12, S., Bingel, J., Augenstein, I., and Søgaard, A. Sluice networks: Learning what to share between loosely related tasks. *stat 1050* (2017), 23.

[25] Saxe, J., and Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on* (2015), IEEE, pp. 11–20.

[26] Saxe, J., Harang, R., Wild, C., and Sanders, H. A deep learning approach to fast, format-agnostic detection of malicious web content. *arXiv preprint arXiv:1804.05020* (2018).

[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[28] Vapnik, V., and Izmailov, R. Learning using privileged information: similarity control and knowledge transfer. *Journal of machine learning research 16*, 2023-2049 (2015), 2.

[29] Vapnik, V., and Vashist, A. A new learning paradigm: Learning using privileged information. *Neural networks 22*, 5-6 (2009), 544–557.

[30] Weiss, G. M. Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter 6*, 1 (2004), 7–19.

[31] Wu, F., Wang, Z., Zhang, Z., Yang, Y., Luo, J., Zhu, W., and Zhuang, Y. Weakly semi-supervised deep learning for multi-label image annotation. *IEEE Trans. Big Data 1*, 3 (2015), 109–122.

[32] Xu, C., Tao, D., and Xu, C. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634* (2013).

A Dataset Statistics

A.1 Vendor Counts Distribution

To better characterize the distribution of the Vendor Counts auxiliary target for our Poisson loss experiment, we plot a histogram representing the distribution of the number of vendor convictions in Figure A.1. The *x*-axis depicts the number of vendors that identify a given sample as malicious, while the *y*-axis represents the number of samples for which that number of detections was observed in our training dataset (note the logarithmic scale). The statistics of the test and validation datasets are similar and not shown here due to space considerations.

We note that there is a peak at zero detections, accounting for the majority of the benign files. Most of the samples considered malicious by our labeling scheme have more than 20 individual detections out of 67 total vendors considered, with a peak around 57 detections.

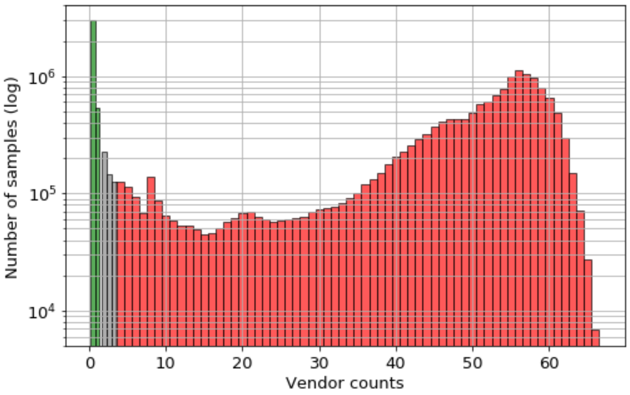


Figure A.1: Histogram of vendor detections per file. Files with zero or one detections (green bars) are considered benign under our labeling scheme, samples with two, three or four vendor detections (gray bars) are considered gray files, and files with more than four detections (red bars) are considered malicious.

A.2 Individual Vendor Responses

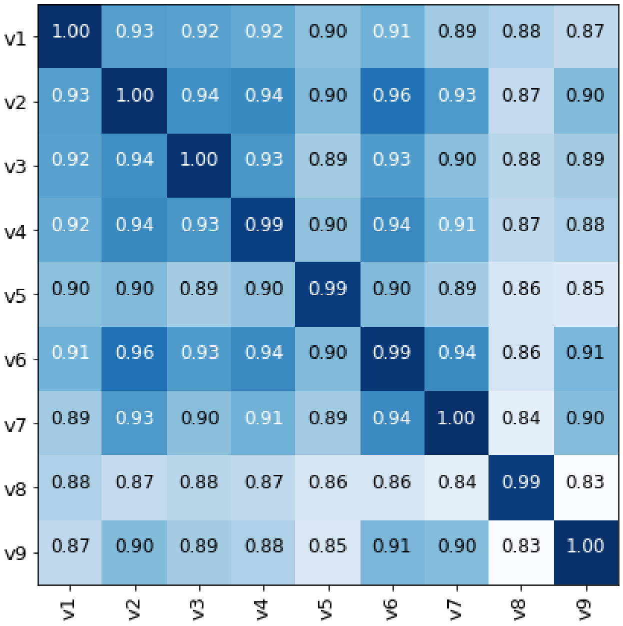
Table A.1 summarizes the number of samples identified as malware, benign and number of missing samples per vendor for the nine vendors used to compute the auxiliary per-vendor malware loss. In Figure A.2 we plot the pairwise similarity of the predictions between vendors. The value in the *j*, *k* position of the matrix is the fraction of samples for which the predictions of vendor *j* are equal to the predictions for vendor *k*. Even though the predictions by each vendor are created in a quasi independent manner, they tend to agree for most of the samples. The diagonal elements of the matrix indicate the fraction of samples for which we have a classification by the vendor (fraction of non-missing values).

	Malware	Benign	None
v1	13,752,004 (69%)	6,110,180 (31%)	20,979 (<1%)
v2	14,751,413 (74%)	5,122,728 (26%)	9,022 (<1%)
v3	14,084,689 (71%)	5,713,116 (29%)	85,358 (<1%)
v4	14,438,043 (73%)	5,239,896 (26%)	205,224 (1 %)
v5	13,778,367 (69%)	5,922,859 (30%)	181,937 (1 %)
v6	15,065,196 (76%)	4,704,695 (24%)	113,272 (1 %)
v7	14,935,624 (75%)	4,927,436 (25%)	20,103 (<1%)
v8	12,704,512 (64%)	7,009,855 (35%)	168,796 (1 %)
v9	14,234,545 (72%)	5,613,604 (28%)	35,014 (<1%)

Table A.1: Individual vendor counts for files in the training set identified as malicious, benign, or missing value for the set of nine vendors used in the per-vendor malware loss 3.3.

A.3 Semantic Tags Distribution

In this section we analyze the distribution of the semantic tags over three sets of samples: i) samples in the training set; ii) samples in the test set; and iii) those which the baseline model classifies incorrectly but our model trained with all targets classifies correctly either as malicious or benign samples. The percentages in Table A.2 represent the number of samples in each set labeled with a given tag. The total number of samples in the test set for which the improved model makes correct conviction classification but the baseline model fails is 665,944. The binarization of the predictions for the baseline and the final model was done such that each would have a FPR of 10^{-3} in the test set. As shown below, those samples with the *adware* tag are the ones that most benefit from the addition of auxiliary losses during training, however we also see notable improvements on *packed* samples, *spyware*, and *droppers*.



	v1	v2	v3	v4	v5	v6	v7	v8	v9
v1	1.00	0.93	0.92	0.92	0.90	0.91	0.89	0.88	0.87
v2	0.93	1.00	0.94	0.94	0.90	0.96	0.93	0.87	0.90
v3	0.92	0.94	1.00	0.93	0.89	0.93	0.90	0.88	0.89
v4	0.92	0.94	0.93	0.99	0.90	0.94	0.91	0.87	0.88
v5	0.90	0.90	0.89	0.90	0.99	0.90	0.89	0.86	0.85
v6	0.91	0.96	0.93	0.94	0.90	0.99	0.94	0.86	0.91
v7	0.89	0.93	0.90	0.91	0.89	0.94	1.00	0.84	0.90
v8	0.88	0.87	0.88	0.87	0.86	0.86	0.84	0.99	0.83
v9	0.87	0.90	0.89	0.88	0.85	0.91	0.90	0.83	1.00

Figure A.2: Vendor predictions similarity matrix. Each entry in the matrix represents the percentage of samples that are the same for any two vendors. The elements in the diagonal of the matrix represent the percentage of the samples for predictions from the vendor are present (i.e., not missing). Note that diagonal values of less than 1.0 are due to missing labels (compare to the final column of table A.1), which we treat as disagreeing with *any* label.

	Train Set	Test Set	Improvement over baseline
adware	21 %	18%	41 %
crypto-miner	7 %	2%	1 %
downloader	25 %	18%	11 %
dropper	29 %	22%	17 %
file-infector	19 %	12%	9 %
flooder	1 %	1%	<1%
installer	7 %	1 %	5 %
packed	34 %	25%	19 %
ransomware	5 %	6%	1 %
spyware	40 %	25%	18 %

Table A.2: Tag statistics for three sets of interest: train set; test set; and the set of samples for which the full model classifies correctly but the baseline model fails.

B Gray Samples Evaluation

In Section 3.6 we observed that 2.5% of the samples in our training set, and 3.7% of the samples in our test set are considered gray samples by our labeling function. While training this is not necessarily an issue since we can assign a weight of zero for those samples in their malware/benign label as noted in Section 3.5. For the evaluation of the detection algorithms though, the performance on those becomes more relevant. To evaluate how our proposed detection model performs on those samples we re-scanned a random selection of 10,000 gray samples in the test set 5 months later than the original collection. From these, 5,000 were predicted by the model as benign and 5,000 as malicious. We expect, after this time-lag, that, with updated detection rules from the AV community, that samples originally labeled as “gray” that are effectively malicious will accrue additional detections and samples originally labeled as “gray” that are effectively benign will accrue fewer detections as vendors have re-written their rules to suppress false positives and recognize false negatives. Thus, the gray sample labels will tend to converge to either malicious or benign under our 1-/5+ criterion. Out of these 10,000 rescans, we were able to label 5,653 gray samples: 3,877 (68.6%) as malicious and 1,776 (31.4%) as benign.

In Figure B.1 we plot the ROC curve for the predictions on the re-scanned samples for our final model trained with all targets, which achieves an AUC of 0.84. Even though the AUC is much lower than the one obtained on our test set, our model trained with knowledge from 5 months earlier (after the first collection) still correctly predicts more than 77% of the samples correctly. We measure this by binarizing the predictions using a threshold that would achieve an FPR of 10^{-3} on the original test set. Furthermore we note that the samples we are evaluating on in this case are more difficult or even ambiguous in nature, to the point that at the original collection time there was not consensus across the AV community.

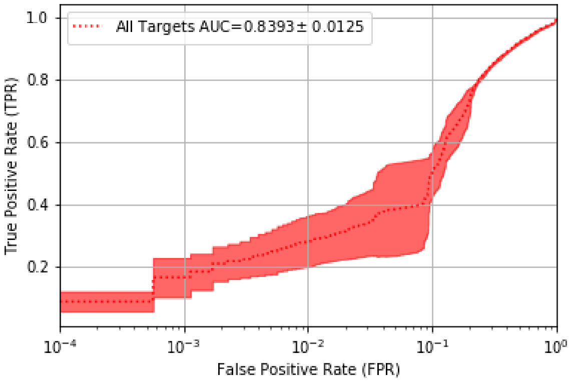


Figure B.1: Mean and standard deviation ROC curve over rescanned samples.

C Relative Improvements

In Table C.1 we present the relative percentage reduction both in true positive detection error and standard deviation with respect to the baseline model trained only using the malware/benign target for various values of false positive rates.

	FPR				
	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}
Poisson	38.05, 61.84	30.19, 30.61	28.68, 48.39	14.29, **85.71**	5.56, **97.14**
RG Poisson	0.00, -52.63	6.17, 16.33	4.41, 48.39	2.86, 57.14	2.78, 95.71
Vendors	47.12, 55.26	32.47, 51.02	18.38, 35.48	14.29, 42.86	0.00, 80.00
Tags	43.63, 64.47	32.47, **81.63**	8.09, 29.03	17.14, 42.86	5.56, 88.57
All Targets	**53.75**, **81.58**	**37.01**, 65.31	**42.65**, **87.10**	**20.00**, 57.14	**8.33**, 94.29

Table C.1: Relative percentage reductions in true positive detection error and standard deviation compared to the baseline model (displayed as *detection error reduction*, *standard deviation reduction*) at different false positive rates (FPRs) for the different experiments in Section 4. Results were evaluated over five different weight initializations and minibatch orderings. Best detection error reduction consistently occurred when using all auxiliary losses. Best results are shown in **bold**.