Mitigating Adversarial Attacks against Machine Learning for Computer Security

David J. Elkind

CrowdStrike, Inc.

david.elkind@crowdstrike.com

October 25-26, 2019

Overview

1-minute summary

1. Static analysis of portable executable (PE) malware is vulnerable to attempts at evasion.
2. Even unsophisticated evasion attempts, such as appending ASCII bytes to the overlay, can make a PE file evasive.
3. I developed a regularization strategy that encourages neural networks to ignore modifications that add "chaff" data to PF files.
4. For a pair of files, one ordinary software sample x and one with added ASCII text $\tilde{x}$, penalize the model proportional to the difference in the neural network's hidden representations $h(\cdot)$:

$$\min_{\theta} \text{Loss}(\theta) + \lambda \| h(x; \theta) - h(\tilde{x}; \theta) \|_2$$

5. My experiments show that this regularization strategy improves the robustness of the neural network to this variety of attack.

Machine learning is vulnerable because adversaries control PE file construction

- The process of creating PE malware is controlled by the adversary.
- This means that the adversary has tremendous latitude to attempt to evade machine learning models.
- If we remove every feature an adversary can modify, (almost?) no features will be left to use in classification.
- Instead, we have to develop models which are **robust** to evasion attempts, in the sense that attempted evasion does not dramatically change the model's classification of $\tilde{x}$.

Goal: Robust machine learning *ignores* evasion attempts

- Appending ASCII text to a file's overlay doesn't change how the file operates (probably), so its benign or malicious qualities are left intact.
- Therefore, we want a machine learning model to treat the modified file x and the non-modified file $\tilde{x}$ *as if they are the same*, because the modification is irrelevant to the operation of the file.
- Regularization which penalizes the difference in the hidden representations achieves our goal: the model is encouraged to have hidden representations for the modified file $h(\tilde{x})$ that match the non-modified file's representation $h(x)$ because the penalty is minimized at 0 when $h(x) = h(\tilde{x})$.

$$\min_{\theta} \text{Loss}(\theta) + \lambda \| h(x;\theta) - h(\tilde{x};\theta) \|_2$$

I call this **pairwise hidden regularization**.

Nothing about this regularization strategy is particular to appending ASCII bytes

Pairwise hidden regularization isn't particular to appending ASCII bytes. ASCII modifications are

- cheap to do to a file (don't have to parse the PE),
- easy to understand,
- doesn't break fragile tooling.

Future research should look beyond ASCII modifications

We focused on appending bytes to the overlay because it's cheap and easy.

- Modifying a PE file using `lief` can break the file.
- Sometimes, `EMBER` will refuse to parse a *modified* file.
- We submitted a ticket to the `lief` Github repo but haven't heard back. :-(

This isn't inteded as a criticism of `EMBER` or `lief`!

But we do need more robust tooling to study a wider range of modifications to PE files.

Focusing regularization on the hidden state enforces consistency in how neural networks "think"

I chose pairwise hidden regularization because it only operates on hidden representations.

$$\lambda \| h(x) - h(\tilde{x}) \|$$

An alternative regularizer operates on the predicted probabilities, which are derived from the hidden representation:

$$\lambda \| \sigma(Wh(x) + b) - \sigma(Wh(\tilde{x}) + b) \|$$

This alternative enforces a consistency in *decision*, but

- the alternative penalty is only large when predicted probabilities are completely mismatched;
- as long as both samples are "in the same tail" of the saturating nonlinearity σ, large $\| h(x) - h(\tilde{x}) \|$ will be suppressed.

The loss function already penalizes incorrect predictions; my pairwise regularization penalizes incorrectly *interpreting* the modified example.

These experiments use a simple feedforward network

- I use the `Ember 2017` data set and feature extraction engine (2351 features).
- I use a feed-forward network with two 256-unit hidden layers, batch norm and 1 residual connection.
- The network is trained until the probability that the loss is decreasing over the previous 5 epochs is less than 0.001.
- The only difference between the baseline model and the robust model is the novel regularization. Both models are trained on pairs of modified and non-modified samples.
- For each model, I use a ROC curve to choose a classification threshold with a FPR of 10^{-3}.
- The baseline model is also trained on *pairs* of samples (i.e., it has the benefit of *data augmentation* via modified samples); the only difference is that pairwise hidden regularization is not applied to the baseline model.

Disclaimer! These experiments have no bearing on CrowdStrike's products

- This analysis was conducted using the open-source `EMBER` data set and feature vectors.
- `EMBER` feature vectors are completely different from the data sources and proprietary feature extraction engines that CrowdStrike uses in its products.
- These results do not have any bearing on the efficacy of any of CrowdStrike's machine learning models, because training a model on different data gives a different result.

Regularizing differences in hidden representations enforces similarity between modified and non-modified files

The baseline model consistently has a larger value of $\|h(x) - h(\tilde{x})\|_2$ compared to the novel model ($\lambda = 10^{-2}$) throughout training.

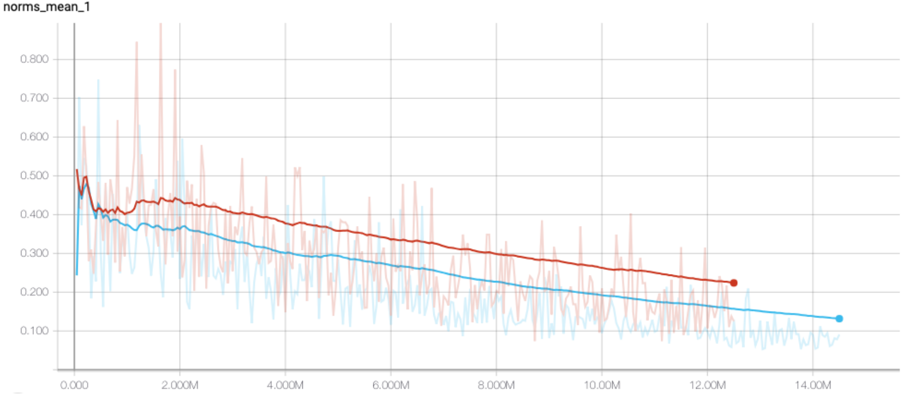


Figure shows how $\|h(x) - h(\tilde{x})\|_2$ evolves during training on a per-mini-batch basis.

Experiment 1 uses "large" modifications in the same way as the training data

For each sample in the *test partition* of the EMBER dataset, we generate samples with a "large" discrepancy in the feature space.

1. Do feature extraction to obtain the feature vector x.
2. Append between 8 and 1023 random ASCII bytes.
3. Do feature extraction to obtain the modified feature vector $\tilde{x}$.
4. If $\|x - \tilde{x}\|_\infty > 0.1$, stop; add $\tilde{x}$ do the dataset.
5. Otherwise, repeat until budget of modifications per sample is exceeded (4 attempts).

Experiment 1 tests each model against the *same corpus* of modified files.

Does pairwise regularization make models more robust?

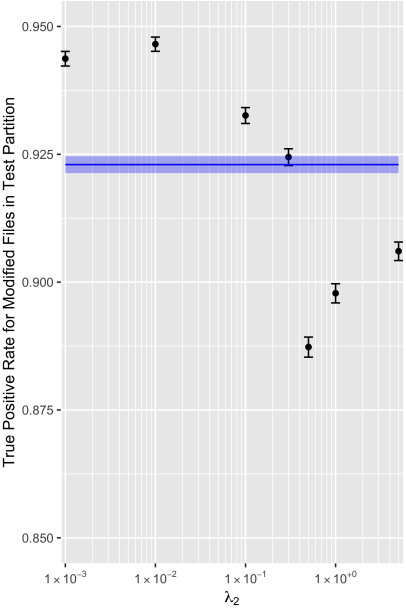
Yes!

Smaller values of λ are more robust than the baseline model.

Choosing the right size of λ improves the model.

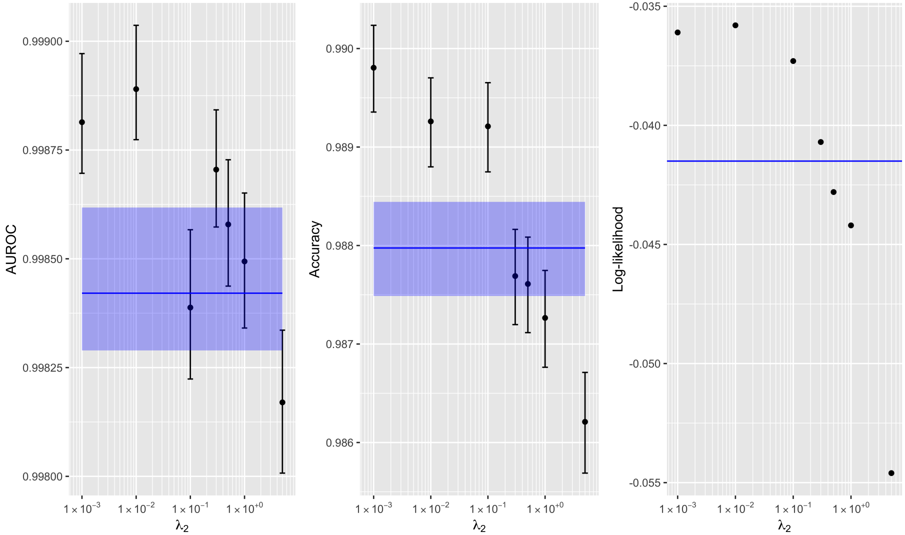
But λ too large makes the model worse.

Bands display 95% confidence intervals.



AUROC, accuracy and log-loss show the same pattern

We see the same pattern with typical model performance statistics: small λ improves the model, but choosing λ too large makes it worse.



Experiment 2: How hard is it to evade the regularized model?

The purpose of this experiment is simulate one method that an attacker would attempt to evade a machine learning model using the ASCII bytes evasion.

For each model, for each sample:

1. Test if a non-modified sample from the test partition is detected.
2. If the sample is detected, modify the sample by appending between 8 and 1023 ASCII bytes.
3. If the modified sample is not detected, proceed to the next sample.
4. If the modified sample is detected, try modifying the sample again until the budget is exceeded (5 attempts).

Why only 5 attempts at evasion?

The point of experiment 2 is to test whether or not it's **easy** to find an evasive sample.

If more attempts are required to make an evasive sample, then it costs more to attack the model.

How is Experiment 2 different from experiment 1?

Two major differences compared to experiment 1:

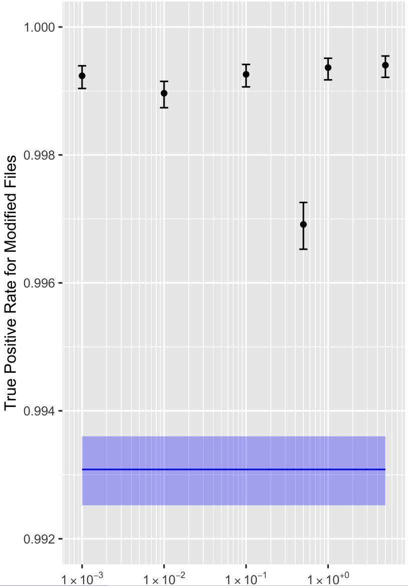
1. In experiment 2, modifications to feature in the test set can have *any* amount of distortion. In experiment 1, we attempted to come up with "large" distortions to the feature vectors.
2. For each model, a *different set* of modified files is produced to attempt evasion. In experiment 1, the same corpus of modified files was used for all models.

Does pairwise hidden regularization make the model more secure?

Yes!

For each of the 100,000 malware samples in the `EMBER` test partition:

1. Do feature extraction and test whether the model detects the sample at the chosen threshold.
2. If the sample is detected, make at most 5 attempts to modify the sample to evade detection.



Experiment 3: How does pairwise hidden regularization change detection rates of **non-modified** files?

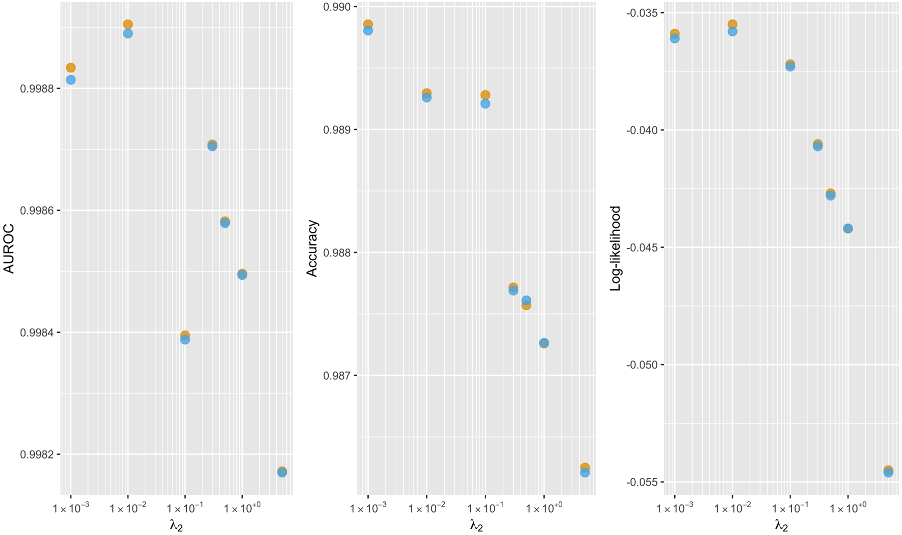
Experiment 3 tests what effect pairwise hidden regularization has on classifying **non-modified** samples in the `EMBER` test partition.

By contrast, experiment 1 and experiment 2 test the effectiveness against samples that were **modified** to be evasive.

Intuitively, we expect that the performance on modified samples should be comparable to the non-modified samples because of the similarities in their latent representations.

Is this intuition correct?

Experiment 3: Performance is very similar for modified and non-modified samples.



(Statistical intervals omitted for clarity.)

Future work

- Determine if specific sequences of bytes are “more evasive” than other sequences (expanding what Fleshman did at DEFCON 2019);
- Extend to other modifications beyond appending ASCII bytes;
- Extend to sequences of several *different* modifications;
- Generalize beyond pairs to arbitrary tuples of heterogenous modifications;
- Produce new modifications of files during network training, instead of a static corpus.

Machine learning and security researchers should study a wide range of modifications

Security researchers should examine all potential modifications to a binary to create evasive malware. Some suggestions appear in [Anderson et al. 2018]

- Appending bytes to the overlay
- Adding an import which isn't called
- Adding a section that's never accessed & is filled with random data
- Appending data to a section
- Creating a new entry point that just jumps to the old entry point
- Removing the signature
- Removing the debug
- Break optional header checksum
- Packing the binary