CS6264 Project 7 Tutorial

Exploring Modeling & Attacking PE Models via MLSploit

Disclosure

We are always looking to improve our homework assignments. If you see any errors, whether they are grammatical or technical, please email the TAs to report them. If anything is not clearly stated, please contact the TAs.

Goals

The purpose of this assignment is to gain experience with training machine learning (ML) and deep learning (DL) models classifying Windows portable executable (PE) malware into families. Specifically, the models will be given two different datasets: benign PE files and malicious PE files from multiple families. After training, you will attack those models using an evasion attack called the Mimicry Attack [1]. Finally, you will be tasked with improving the models which were attacked and write a report about your experiences and observations.

Warning

The malware binary we provide you (and the malware produced by MLSploit) is real malware. Do not under any circumstances execute these malware **EVER**. It is a compiled form of the rbot malware family and antivirus companies are well-aware of their existence (https://github.com/ytisf/theZoo). We have not applied any static obfuscation to them so they should be easily detectable by AV companies. You are to use these binaries responsibly by only *reading* their byte contents (e.g., using tools like https://github.com/erocarrera/pefile).

Requirements

To perform this lab, you will need a basic understanding of what computer malware is, along with some knowledge of how ML models are trained to detect and classify them. You will need to know how to program and how to train ML models using libraries (e.g., http://scikit-learn.org/stable/). Finally, you should be familiar with how the PE file format is structured (https://docs.microsoft.com/en-us/windows/win32/debug/pe-format).

Overview: Grading is out of 100 points.

[10 points] **Task 1**: Training DL Models

- Train LSTM, CNN, and RNN models on API call sequences.

[10 points] **Task 2**: Attacking DL Models

- Attack models via mimicry attack.

[20 points] **Task 3**: Detecting Attack

- Train model based on static features to detect the attack sample.

[10 points] **Task 4**: Training classic ML Models

- Train models based on additional features (API existence, frequency, and arguments).

[10 points] **Task 5**: Attack Transfer to ML Models

- Evaluate how the attack sample evades detection from these models.

[10 points] **Bonus Task**

- Use machine learning to evade malware detection models. You will use two open-source toolsets: one which detects malware using static features and one which trains a reinforcement learning model to craft malware samples which evade the detection model.

[40 points] **Written Report**

- You will be graded based on how well this report is written. Times New Roman 12-pt font. Include diagrams of your work and citations to back up your claims and assumptions. Can include screenshots of your experience with MLSploit, but add these to an Appendix at the end of the report (you will not be graded based upon them).

Setup

You will be using MLSploit for a majority of this assignment. You will use it to train and attack machine learning and deep learning models of PE malware.

- Either (1) connect to Georgia Tech's VPN, (2) visit Georgia Tech's browser-based VPN, or (3) be on Georgia Tech's campus network. You will not be able to access the website without doing this.
    a. https://oit.gatech.edu/services/end-point-computing/virtual-private-network-vpn
    b. https://faq.oit.gatech.edu/content/how-do-i-get-started-campus-vpn
    c. https://vpn.gatech.edu/https/mlsploit.org/
- Create an account on https://mlsploit.org
    a. Use a strong password.
- Read the "PE Module Tutorial" and "MalwareLab Module Tutorial" attached with this assignment.
- For your own information, the functions of the PE module (with exception to the PE transformer - which performs binary rewriting to statically modify PE binaries in order to implement attacks) are open sourced here: https://github.com/evandowning/mlsploit-pe
- Finally, only a certain number of jobs will be allowed to run at once on our internal servers. So if you see that your job is "PENDING" don't worry. It should change to "RUNNING" within an hour or so depending on the jobs other students are running.

Submitting Answers

Compress the deliverables for each task into a .tar.gz file called [GTUsername]_cs6264_lab07.tar.gz with the following directory layout:

- task1/
  - pe.model.zip
  - prediction.zip
  - *.log.txt files
- task2/
  - attack-exe.zip: the attack samples generated from MLSploit to evade your models
  - attack-feature.zip
  - attack-prediction.zip
  - attack.cfg.zip: Configuration file from MLSploit
  - *.log.txt files

Submitting Answers

- task3.a/
  - detection1.py: Source code (preferably in Python) that will train a new model that will detect the attack from the previous task
  - detection2.py
  - detection3.py
  - (others here if you wish)
- task3.b/
  - model1.zip
  - model2.zip
  - model3.zip
  - (others here if you wish)
- task4/
  - pe.model.zip
  - prediction.zip
  - *.log.txt files

Submitting Answers

- task5/
  - prediction.zip
  - *.log.txt files
- bonus/
  - model.zip
  - *.log.txt files
- report.pdf:
  - This report should contain screenshots of your findings and explanations for why the certain screenshot happened. For example, if your screenshot is comparing the results of how well different models detected the attack in part 2, then an explanation for why the results differed should be included.

Introduction

One of the goals of a security company is to detect malware over time: old known malware (e.g., Zeus), new malware variants (e.g., Gameover Zeus, Spyeye, Carberp), and new malware (e.g., Petya). Once a new malware family (or variant) has been discovered, the company needs to update their end-host solution to be able to identify it. The purpose of this lab is to expose you to this scenario: to classify malware into families. The goal of classifying malware is to understand what type of malware you have identified. This is subtly different from malware detection (the goal of which is to determine if something is malware or not). The benefit of classifying malware is to use what you have seen in the past to help with your task. For example, if your goal is to automate cleaning a victim's machine after an infection, it could be useful to know what family of malware infected it (i.e, family *A* is known to affect files and registers *x, y,* and *z*, so those need to be removed or reset). If your goal is to reverse engineer a malware sample (e.g., to study it further), it could be useful to know what family it belongs to in order to identify specific regions you know exist (e.g., family *A* is known to have a ransomware encryption routine and an IRC C&C routine, thus we should look for API calls which are indicative of these activities).

Introduction

There are too many new samples and variants generated every day to simply hand-label each malware binary. Luckily, machine learning (ML) can be used to classify malware samples without having to do so. In general, machine learning works as is illustrated in Figure 1. First, the analyst collects many malware samples with their associated malware family labels. Second, they identify and extract features for discriminating between each malware family (e.g., API call sequences, static features, network features, etc.). Finally, they train a ML model to distinguish these features for each family. To evaluate how well the model has learned, they save a portion of their dataset (not used for training the model) and test how accurate the model is at predicting the labels of each unseen sample.

Introduction

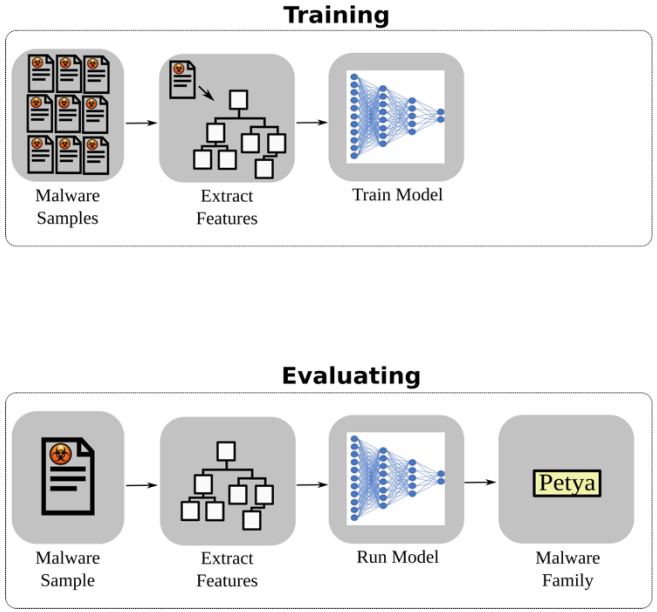

Figure 1: General workflow of training and using a malware classifier

Introduction

However, this process can be tedious and requires some knowledge in order to

(1) identify and extract these features and

(2) implement, train, and evaluate the ML models.

Thus, we developed MLSploit. MLSploit is a framework developed by Georgia Tech PhD students with oversight from Georgia Tech Professors and Intel research scientists (http://istc-arsa.iisp.gatech.edu/). Its goal is to assist its users to train and test various machine learning solutions (e.g., image classification, object recognition, malware detection & classification) against various attacks without requiring a significant amount of experience in machine learning and information security. The tool decreases the barrier of entry normally required to run these types of experiments and evaluations.

You will notice that this tool is relatively easy to use: it essentially does the work for you in terms of implementation. The only challenges you face in this project are

(1) deciding what parameters to use for each ML model (in order to produce an accurate model) and

(2) having an imagination to explain the results you see and deciding what features you'll choose to accomplish Task 3.

Task 1: Training DL Models

Your first task will be to train 3 deep learning models using API call sequences. As described in the tutorial, select “New Pipeline” and name it something like “DL Training”. Then click-and-drag training ensemble models from the PE module in the left-hand column. Train an LSTM, RNN, and CNN model on sequences on pe.train.txt (provided to you). Begin with a sequence window size of 32, but try increasing and decreasing it to achieve better accuracies. Note that shorter sequences will take much longer to train. Also attach an ensemble evaluation and use pe.eval.txt. Download the resulting model files (contained in a single zip file), predictions (also in a zip file), and log files.

Task 1: Training DL Models

- **Behind the scenes**: We have implemented using these three deep learning models using Keras (https://keras.io/). The sequences model takes as input, the window size. This window size is the length of each subsequence that the model gets trained on. The model can perform malware detection (i.e., binary_classification) or malware classification (i.e., multi_classification). After running this module, you will get a trained model along with the predictions that it made for each sample.
- **Time duration**: This should take about an hour or two to finish running (depending on the servers' usage), so start this ASAP.
- **MLSploit Input Files**: pe.train.txt, pe.eval.txt
  MLSploit Output Files: pe.model.zip, prediction.zip, and log.txt files.
- **MLSploit Hint**: Set pipeline to train and evaluate all three models simultaneously (i.e., select "true" for all three sequence_xxx) while you find the ideal window size.
- **Hint**: you can find the accuracies in the log.txt file, the prediction.zip file gives you the classification results of the samples of each model.
- **Deliverable**: Output files produced by MLSploit website.

Task 2: Attacking DL Models

Perform the mimicry attack [1] using MLSploit similar to how it is described in the tutorial. Create a new pipeline and name it something like “Mimicry Attack”. Upload the file “pe.benign.txt” given to you as well as a zipped file of the models you trained in Task 1. Also, upload the file “pe.target.txt” given to you. It contains the filename of the malware you will be morphing to attack the models. The mimicry attack will discover a sequence of system calls which will precisely mimic the system calls produced by a benign application. Thus, any classifier will conceptually misclassify it as benign software (given the same datasets). This task will produce a series of API calls which will evade the models along with a configuration file.

Give these files to the PE transformer, which will generate whole new samples. This module will produce a runnable executable which will exhibit the behaviors the attack module defined. The parameter for the PE Transformation function asks for a “hash value”, but it’s really the file name of the binary. Finally, give the resulting malware to the models and evaluate them to demonstrate that the attack worked.

For the PE Transformer’s hash parameter use “rbot-original.exe”. For generations, use 10, it will produce 10 malicious executables for mimicry attack.

Note that these attack predictions are made off of what the malware is *expected* to execute. Later, in Task 5, we’ll actually execute these malware samples and observe what is *actually* executed (verifying that our malware will dynamically evade these models).

Task 2: Attacking DL Models

- **Behind the scenes**: This code creates new sections in the Windows PE file that call API calls with various arguments. It then searches for targeted calls within the binary (e.g., whenever it calls WriteFile()) and inserts jump instructions to this new section (thus inserting a new API call after, say, WriteFile()). This process of modifying a binary is called binary rewriting. After running this module, you will receive multiple binaries (10) which have been rewritten to insert API calls during the malware's execution. Thus its goal is to evade dynamic API-based malware detection/classification models.
- **Time duration**: This should take around 10 minutes to finish running (depending on the servers' usage).
- **MLSploit Input Files**: pe.benign.txt, pe.target.txt, pe.model.zip [From task 1 training pipeline's outputted files. NOTE that this file is large and may take a minute to upload. So be patient after clicking "Upload files". Do not leave the page or press the "x" at the top of the "Manage Files" window. It will appear after a minute or two.]
- **MLSploit Output Files**: attack-feature.zip, attack-prediction.zip, attack.cfg.zip, attack.exe.zip, log.txt files.
- **MLSploit Hint**: The pipeline described in the tutorial performs all of the necessary steps as described. The attack-exe.zip has the password "infected". **Do not execute these binaries under any circumstances**.
- **Deliverable**: Output files produced by MLSploit website.

Task 3: Detecting Attack

For this task, you will not be using MLSploit. You will be writing a program to train a ML model yourself using features you choose. Since MLSploit is open-sourced (and there are numerous tutorials on scikit-learn), it should be painless for you to train a ML model. The only real challenge is picking the features you'll train it on. We have provided you with a code template. **This task is left intentionally bare with few guidelines - use your imagination and think critically about the features you will choose**. Now is the time to be creative!

Task 3: Detecting Attack

Train a new model on static features (i.e., features you can get directly from the samples without executing it) to detect the attack instance. Since we trained models on dynamic features (i.e., API call sequences), we need to find some other feature not based on API call (since our Mimicry attack ran on those features). Use the provided static features file for all samples (benign and malicious) to come up with new features (combinations of features from the static features file) to more reliably detect the attack (as you'll see some of the benign samples get detected as anomalous as well).

You need to unzip the attack.exe.zip (which contains 10 mimicry attacks) file and statically examine the differences between it and malware-original.exe to discover some interesting artifact (or side-effect) of the attack (e.g., using pefile, readelf, objdump etc.). Compare your assumptions to the static features of the other malicious and benign files to determine if your hypothesis is valid. The features can be found in the JSON file attached with this assignment. These static features were extracted with EMBER (referenced below). Notice that all features for all binaries pe.train.txt are provided. **You now need to extract them from attack-x.exe using EMBER.** To do this, review ember's code on how they extract their features (https://github.com/endgameinc/ember/blob/master/scripts/train_ember.py#L25, https://github.com/endgameinc/ember/blob/master/ember/__init__.py).

Task 3: Detecting Attack

Important distinction: The goal of this task is to detect **evasive** malware (i.e. morphed malware via unnatural artifacts introduced into the binary). Its goal is not to detect the malware itself. Your results should show that the benign software (along with the original malware sample) are **nominal** and the attack malware samples as **anomalous**.

Hint 1: For example, because of the way the PE transformer adds API calls to be executed dynamically, more "jmp" instructions are added. So two basic features which could be used are the number of "jmp"-related instructions and the distance to the addresses jumped to. You may not use these features for your solution, but this is one example. Other features you can use are: binary size, binary section entropy, etc. For inspiration, check out the static features extracted and modeled by a cybersecurity company (reading their paper may also help [2]): https://github.com/endgameinc/ember. You can use this script to help you extract features: https://github.com/evandowning/ember/blob/mlsploit/scripts/extract.py.

Hint 2: We recommend you use Python to create your new models because it has support for easy-to-use ML libraries (http://scikit-learn.org/stable/). You can also look at the open-sourced modeling code that the PE module uses to get an idea of how to extract features from malware and train ML models on them: https://github.com/evandowning/mlsploit-pe.

Task 3: Detecting Attack

Deliverable: Source code to train models to detect attack (given entire dataset). (We have provided you a template for you to get started, it is up to you to choose to use it or not.) Use at least three different features from the static features file (separately or in combination as your choice) on three different models. Explain in your report why you chose the features and models you did. For example, provide a graph comparing the feature you chose for detecting the attack sample with the unmodified dataset (origin benign/malicious dataset). Is the model you chose good for the features you extracted (e.g., low versus high dimensional features)? Also discuss if these new features scale for future unknown samples and why? Discuss how an attacker can also evade your new features? Provide citations to academic or industry papers where appropriate.

Task 4: Training classical ML Models

Train at least 3 more ML models on other features (API call existence, frequency, and arguments) using MLSploit. For each feature you can train any one of a number of classical ML models: Random Forest, Neural Network, Naive Bayes, k-Nearest Neighbors, or Stochastic Gradient Descent. Read scikit-learn’s documentation to learn more about each model. Two parameters can be manipulated: the number of trees in the random forest, and the value of k for k-Nearest Neighbors.

Task 4: Training classical ML Models

- **Behind the scenes**: Similar to Task 1, this module implements classical ML algorithms from scikit-learn (http://scikit-learn.org/stable/). In Task 1, you trained DL models on API call sequences. This module trains other API-related features: existence (whether or not an API call was executed), frequency (the number of times an API call was executed), and arguments (the arguments passed to each API call).
- **Time duration**: In the worst case, if all model architectures are chosen for each feature (for a total of 15 trained models), expect it will take at least 30 minutes to finish (depending on MLSploit's server's traffic).
- **MLSploit Input Files**: pe.train.txt, pe.eval.txt
  MLSploit Output Files: pe.model.zip, prediction.zip, and log.txt files.
- **Deliverable**: Trained models produced by MLSploit website.

Task 5: Attack Transfer to ML Models

- Evaluate whether or not the attack sample evades the ML models as well using MLSploit.
- To do this, we must first execute the malware in a controlled, responsible environment. We use an in-house analysis framework called MalwareLab. Copy the 10 attack executables (from attack.exe.zip of Task 2) into a new single folder “attack-exe” and compress it via zip **with no password**. Upload that zipped folder to mlsploit.org. Then, run the “Submit” function via the MalwareLab pipeline (as demonstrated in its tutorial).

Task 5: Attack Transfer to ML Models

- **Behind the scenes**: This module executes the evasive malware samples you generated back in Task 2 in a safe and controlled environment.
- **Time duration**: Each sample is executed for 2 minutes in parallel. So this may take a few minutes to complete (depending on the traffic on the MalwareLab server).
- **MLSploit Input Files**: attack-exe.zip **MLSploit Output Files**: log.txt files

Task 5: Attack Transfer to ML Models

Once the samples are finished executing, we can evaluate our trained classical models (from Task 4) on the dynamic traces. To do this, we can create a new pipeline with just “Ensemble-Evaluate”. The MalwareLab module will output logs denoting the SHA-256 value of each executable submitted. Create a file “attack.eval.txt” of these hash values (similar to how pe.train.txt and pe.eval.txt are formatted). I.e., each line is formatted as “hash_value TAB rbot”

Task 5: Attack Transfer to ML Models

- **Behind the scenes**: This module is something you've already run before. It extracts features from the malware you just executed and runs them through the models you just trained. This will show whether or not the samples which evaded the DL models in Task 1 *also* evade the ML models you trained in Task 4.
- **Time duration**: This may take up to 30 minutes (depending on the traffic on the MalwareLab server).
- **MLSploit Input Files**: attack.eval.txt pe.model.zip [the one that was outputted from tasks 1 & 4]
- **MLSploit Output Files**: prediction.zip and log.txt files.
- **Deliverable**: Results on how DL & ML models classify the attack samples from Task 2 above. Explain in your report why the models were or weren't successful at classifying the malware. How do these new features compare to the ones you studied in Task 1?

Bonus Task

The goal of this task is to gain experience using reinforcement learning (a type of machine learning technique) to generate malware samples which evade a pretrained model. To train this new model, you will be using Ember, a non-peer reviewed (i.e., self-published) paper which extracted static features from 50k PE malware samples and 50k benign PE samples and trained a classifier to determine if the sample is benign or malicious [2]. They achieved decent accuracy with their features (which are based on prior works which extract static features from malware for malware detection).

First, please read this paper [2] to fully understand their methodology.

Next, run this on MLSploit by creating a new pipeline (using PE Module) with (1) Ensemble-Train (enabling Ember) and (2) Ensemble-Evaluate (enabling Ember).

Bonus Task

- **Behind the scenes**: The code MLSploit uses can be found here: Ember (https://github.com/evandowning/ember/tree/mlsploit). MLSploit will use this code (specifically train_ember.py and test_ember.py) to train the Ember model using the authors' dataset. The dataset of 100k samples used in the Ember paper already exists on MalwareLab's servers.
- **Time duration**: This should take about 30 minutes to finish. **MLSploit Input File**: pe.train.txt and pe.eval.txt
- **MLSploit Output Files**: pe.model.zip and log.txt files.

Bonus Task

After training the Ember model, you will be using another implementation by the same authors to evade it [3]. Their methodology uses reinforcement learning to modify already-existing malware binaries by directly changing the contents of the binary such that it does not break the binary (similar to the PETransformer module). Before, you were attacking models using a fundamental attack on *features* (i.e., the mimicry attack). Now you will (essentially) use ML to attack ML.

First, please read this paper [3] to fully understand their methodology.

Next, run this on MLSploit by creating a new pipeline (using PE Module) with Ember-Attack. The output logs will display numbers (along with warnings you can ignore) that represent how “malicious” the sample is on a scale from [0,1]. In their paper, the authors use a threshold of 0.9, meaning any binary that scores 0.9 and above is considered as malicious, and benign if it scores below a 0.9. Gym-malware will continue modifying malware samples until it can find the ideal combination of modifications so as to cause the Ember model to score that sample below 0.9 (you will see this in the log files which are outputted). You will also be given each modified malware sample so you can examine them for yourselves (ember-attack.zip). As stated before, please do not distribute these samples outside of this class and do not run these samples - these are real malware samples!

Bonus Task

- **Behind the scenes**: The code MLSploit uses can be found here: Gym-malware (https://github.com/evandowning/gym-malware/tree/mlsploit). Logic for what features they modify and how can be found here: https://github.com/evandowning/gym-malware/blob/mlsploit/gym_malware/envs/controls/manipulate2.py#L367.
- **Time duration**: This should take about 30 minutes to finish. **MLSploit Input File**: pe.model.zip
- **MLSploit Output Files**: ember-attack.zip and log.txt files.
  Deliverable: Please add your answers to the questions to the written report document above.

Bonus Task

After training and generating malware samples which evade the static model, answer the following questions and put them into your report (above):

Questions:

(1) What things were changed about the binaries to make them successfully evasive? (see standard output)

(2) How effective is this method compared to mimicry attack? I.e., number of successfully evasive malware samples produced by both methods, time it takes to produce evasive samples, etc.

(3) Similar to how the mimicry attack was implemented to produce new binaries, how does this method compare? How can a defender identify likely suspicious binaries?

(4) Submit the evasive samples to VirusTotal. What does it classify them as? What does this tell you about AV company static-based malware detection solutions? Attach the VirusTotal reports to your solutions folder.

Written Report

Please compile a document describing your experience and your decision processes with training DL models and choosing features to detect the attack sample, as well as the results of the attack samples on the classical ML models. Include diagrams of your work and citations to back up your claims and assumptions. Can include screenshots of your experience with MLSploit, but add these to an Appendix at the end of the report (you will not be graded based upon them).

Quality is always valued higher than quantity. There is no page limit nor minimum page requirement, but do not write more pages than is needed to communicate your ideas and answers to questions effectively.

References

[1] D. Wagner and P. Soto, “Mimicry attacks on host-based intrusion detection systems,” in Proceedings of the 9th ACM Conference on Computer and Communications Security, 2002, pp. 255–264, Accessed: Oct. 07, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=586145.

[2] H. S. Anderson and P. Roth, “Ember: an open dataset for training static PE malware machine learning models,” arXiv preprint arXiv:1804.04637, 2018. [Online]. Available: https://arxiv.org/abs/1804.04637.

[3] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, “Learning to evade static pe machine learning malware models via reinforcement learning,” *arXiv preprint arXiv:1801.08917*, 2018, [Online]. Available: https://arxiv.org/abs/1801.08917.