Semester Plan & Progress Reports

Luke Kurlandski

08/2022 - 12/2022

Check-in 0 (08/29/2022)

Goal for the semester is to develop a research objective to work on for the RPA in Spring 2023.

Plan (09/08/2022)

- By the end of September (09/30/22)
  - Find a research topic to focus on. Ideas include
    - Practical black-box evasion attacks within the problem space (directly modifying the malware itself and ensuring its functionality remains unchanged)
    - Poisoning attacks on malware classifiers
    - Code modeling/NLP techniques applied on malware source code, opcode, etc.
    - In general, focus on PE malware (90% of malware) and static malware research (more suited to my current skillset)
  - Obtain a computer from Charles.
  - Read other student's RPAs.
- By the end of October (10/31/22)
  - Determine research objective within topic.
    - What results I want to show
    - What conclusions I want to make
  - Literature review month.
- By the end of November (11/30/22)
  - Develop a methodology to achieve objective.
    - Experiments to run
    - Determine other knowledge needed
  - Literature review month.
- By the end of December (12/31/22)
  - Possibly have some preliminary results
  - Adjust methodology as needed
  - Rinse and repeat
- Semester-long goals/activities
  - Read as much as possible.
  - Take notes.
  - Pass classes.
  - Attend dissertation defenses, research talks, etc.
  - Produce one-page progress reports each month.

Check-in 1 (10/12/2022)

Progress Report

For the most part, I am slightly ahead of schedule in terms of developing an idea to work on. However, it is a hard idea and remains to be seen whether or not it will work. The poster for AI@RIT serves as the more conceptually-rigorous component of this month's updates. I did not read other student's RPAs because it seemed like that might be more beneficial to save for either December or January.

Plan (10/12/2022)

- By Monday 10/31/22
  - Proof-of-concept example demonstrating that rewriting snippets can make them less suspicious-looking.
  - Initial benign-looking and malicious-looking nonparallel corpora.
  - Emphasize literature review.
- By Monday 11/21/22
  - Initial unsupervised translation system (denoising and backtranslation).
  - Refined nonparallel corpora (function segmentation, different explanation algorithms).
  - Assess whether switching to Android malware would be productive.
  - Emphasize literature review.
  - Check-in II.
- By Monday 12/19/22
  - Refined translation system (pretrained LLM encoder/decoder).
  - Setup adversarial attack simulation.
  - Attain preliminary results.

Check-in 2 (11/21/2022)

Progress Report

For the past several weeks, we have been investigating whether or not an adversarial attack that rewrites malware's source code could be successful. To determine whether such an attack could actually fool a classifier, we modify the bytes of PE malware inside of the PE file's .text section and run the perturbed malware through a raw-byte classifier to see if it evades detection. Preliminary results indicate that using random bytes or baseline/padding bytes does not produce perturbed malware that can evade a classifier. However, using bytes from the .text section of a benign executable has been successful to some degree. At the very least, this confirms that the source code within malware is meaningful to the classifier. If we can figure out a way to make malicious source code appear benign, we should be able to make at least some malware evade classification.

We want to have an understanding of how many bytes in the malware we would need to change to launch a successful attack. Therefore, it makes sense to repeatedly select a small chunk of bytes from a malware sample, perturb it, and evaluate the perturbed sample against the classifier. There are two main aspects to consider when performing these incremental substitution experiments. The first is how to select bytes from the malicious example to be replaced at every iteration. We experiment with selecting chunks of the malicious executable in order, randomly, and according to how suspicious an explanation algorithm considers the chunk. The second aspect to consider is how to select bytes from the benign sample to use as a replacement. We experiment with selecting the chunks of the benign executable that correspond to the offset of the chunk selected from the malicious executable and selecting chunks which are the least suspicious according to an explanation algorithm.

Preliminary results on a poorly-learned model seemed to suggest that the SHAP explanation algorithm was successful at identifying suspicious chunks of malware and replacing these suspicious chunks was an efficient way to produce perturbed malware that evades a classifier.

To build upon this, we use a pretrained low-memory "Global Channel Gating" MalConv model trained extensively on a large corpus of malicious and benign executables. Running experiments on this model produces some interesting results, which partially contradict our previous findings (Figure 1). First, the overall evasion rate of the perturbed malware was slightly lower than in our initial experiments. Second, the relationship between the percentage of perturbed malware examples classified as benign and the amount of bytes modified was more linear, rather than logarithmic. Third, selecting the most suspicious chunk of malware for replacement, instead of a random chunk or each chunk in order, only slightly improved the evasion rate. Fourth, selecting the least benign chunk from the malicious executable slightly improves the evasion rate compared to selecting the corresponding chunk.

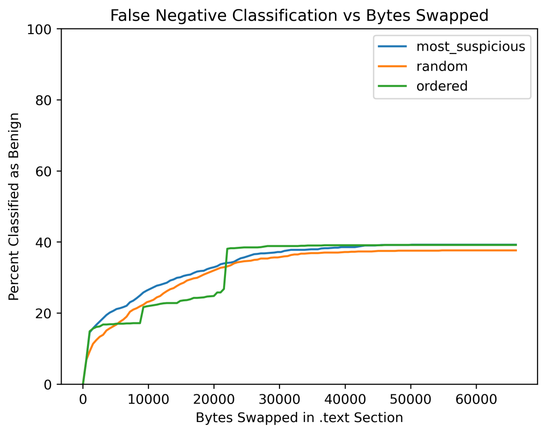
The second, third, and fourth observations are somewhat concerning. Ideally, we hoped there would be a more logarithmic relationship between the evasion rate and the proportion of bytes being replaced because this would mean that less bytes would need to be modified as part of the attack. We also anticipated more significant differences between the different methods of selecting bytes from the malicious sample to be replaced and more significant differences between the methods for selecting bytes from the benign sample to replace with. These observations may indicate that the explanation algorithm is not very successful at identifying which regions are suspicious-looking and which are not.

My idea of using machine translation to rewrite malware rests on the premise that we can identify which snippets of code are benign-looking and which are malicious-looking. With that in mind, I believe I need to spend more time on this aspect of the research and verify that the explanation algorithms provide useful information.

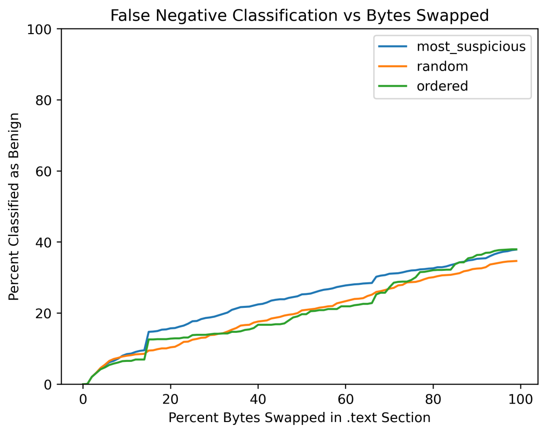
Plan (11/21/2022)

The plan I created last check-in was definitely overly ambitious. Here is a loose plan moving forward.

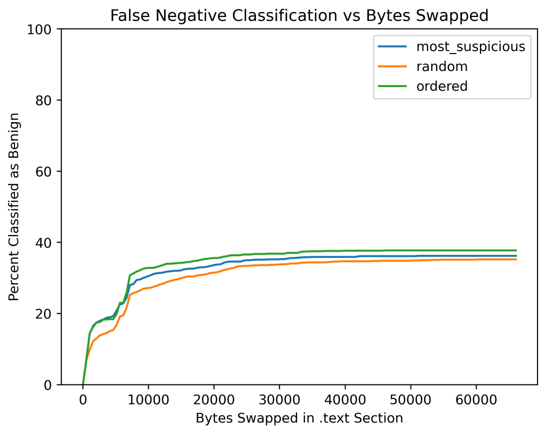
- By Monday 12/05/22
  - Find some evidence that the results from an explanation algorithm are useful
  - Build nonparallel corpora using explanation methods
- By Monday 12/19/22
  - Build a pseudo NMT model that produces syntactically correct x86
  - Check-in 3
- By Monday 01/16/22
  - Other research goals TBD
  - RPA Related Work first draft
  - RPA committee members list



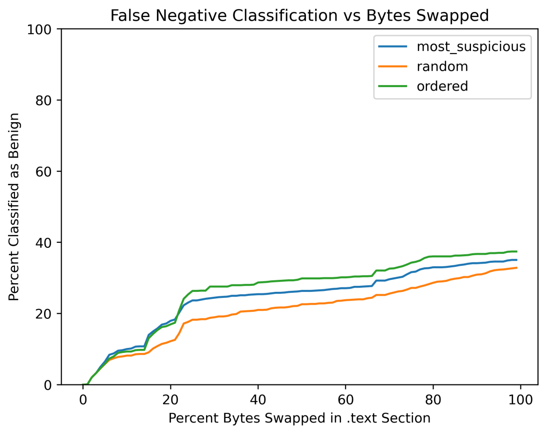
(a) Corresponding-benign swap method (absolute).



(b) Corresponding-benign swap method (proportion).



(c) Least-suspicious-benign swap method (absolute).



(d) Least-suspicious-benign swap method (proportion).

Figure 1: False negative rate vs the amount of bytes swapped using different methods of incrementally perturbing a malicious executable. There are four subfigures. Subfigures (a) and (b) use the chunk of bytes from the benign executable that corresponds to the offset of the chunk of bytes from the malicious executable that was selected for replacement. Subfigures (c) and (d) use the next least suspicious chunk of bytes selected from the benign executable at every swap iteration. Each subfigure displays three lines. Each line indicates the false negative rate for a different method of selecting chunks to substitute from the malicious file. The three different methods are: ‘most_suspicious’, which selects the most suspicious chunk from the malware for modification, ‘random’, which selects a random chunk, and ‘ordered’, which selects chunks in the order they appear in the original malicious executable.[1]

[1]Theoretically, the three lines in each individual plot should achieve the same evasion rate. The fact that they do not is because of a bizarre feature/bug in the original implementation of the Global-Channel-Gating MalConv. It is not a major issue, but I have been in contact with Edward Raff about the feature/bug on GitHub (https://github.com/NeuromorphicComputationResearchProgram/MalConv2/issues/6)