Weekly Log

Luke Kurlandski

November 14, 2022

W1: 08/21/2022

Notes

- There are limited examples of applying LMs specifically to malware. Found no examples of using a highly advanced transformer-based architecture, such as BERT.
- Could possibly train a code LM specifically on malware, e.g., MalBERT or MalELECTRA.
- Malware generation using seq2seq? Malware obfuscation using code-repair techniques?
- Several works use word2vec, RNNs, LSTMs for language modeling malware, but there is no one paper that uses all three on the same dataset for a holistic comparison. Furthermore, most of these datasets are not publicly available.
- No paper has used the AST approach taken by Uri Alon et al. specifially for malware
- Perhaps use a large language model of code after decomposition, such as PolyCoder, for the classification? Or would this not work because the decompiled malware would not have useful identifiers? Would it work if ASTs were used to train the LM?
- Is there a BERT for Assembly language? If so, that could be an excellent LM for malware.

Report

- Reading about malware detection and classification
- At this point I'm probably more comfortable/qualified with static malware detection than dynamic detection
- The research in using language modeling techniques on malware is fairly new and undeveloped
- Many papers doing this do not compare their methods to any baseline method
- Many do not use publicly available datasets
- Ideal dataset for this research is probably Sophos-ReversingLabs 20 Million dataset

Minutes

- This week I will work on replicating some of the experimental results produced in other papers

W2: 08/27/22

Onboarding Meeting

- Dr. Pan, Dr. Wright, and I will meet during the Malware Group meeting time, during the reading group when I am presenting, and as needed.
- The RPA is essentially a short research paper with a robust literature review.
- I should begin preparation for it immediately.
  - See rubrics on RIT site.
  - Could focus on a particular type of malware detection, such as Windows macro malware, PDF malware or more general malware detection.
  - Could look into generating adversarial malware examples, along the lines of code2vec.
- "binary lifting" from binary to intermediate (similar to assembly, which could be recompiled again)
- Goal is for second half of semester to have a solid objective for research
- Innovation: if its been done ten times, its not innovative enough
- Read other students' RPAs

Notes

- Adversarial malware: generate embeddings for malware opcode and substitute to fool the detector
- Little research done on poisoning attacks

Report

- Read about adversarial malware generation and defense.
- Most literature is concerned with adapting CV methods that use continuous representations to malware with discrete representations.
- Might be novel to explore adversarial malware with continuous representations, i.e., malware represented using a learned embedding.
- Adversarially modifying the embedding would be simple, but figuring out how that can be achieved by modifying the malware itself would be challenging.
- Look into NLP/CV/audio processing adversarial techniques that use embeddings.
- Dr. Pan suggested and idea of swapping malware API calls with similar calls as an adversarial approach.
- If using embeddings as a feature representation this would fit into that idea nicely.

Minutes

- Develop a plan for the semester

W3: 09/02/2022

Notes

- Would using LM techniques in malware detection make them robust to adversarial attacks?
- Idea: build a [novel?] LM-based classifier, demonstrate its robustness to adversarial attacks, propose a new adversarial attack based upon NLP techniques!!!! Preferably use windows, android, and PDF datasets
- AST for assembly code probably not logical, but would it even be needed?
- Diverse ensembles more robust to adversarial attacks?
- Could be interesting to train an ensemble using dramatically different feature reps, then test an RL, GAN, Graph, and traditional Adversarial malware attacks
- Research on attacks should be focused on the black box problem space scenario.
- Identify problematic sequences of opcode and replace them with computationally equivalent ones to fool detector?
- Adversarial Malware needs to be verified for correctness in Cuckoo Sandbox
- An adversarial attack designed specifically for ensembles?
- Adversarial training works...but is there a point where retraining fails?
- Have RL, genetic, or GAN attacks been tested against ensembles? One paper tested various attacks against a small ensemble, but what about a large one? IE test black box attacks against ensembles, possibly that use different feature reps.
- Test ensembles using different feature rep vs ensembles using the same feature rep
- Read every source that uses black box attacks in the feature space. Determine whether or not their methods use reasonable defense mechanisms. Test their methods using reasonable defense mechanisms.
- Does the surrogate model negate the black box model entirely? Are all black box attacks really white box?
- Testing adversarial techniques against AV software would be more practical than just testing against a single DNN
- Idea: adversarial testing against an ensemble of AV softwares, like VirusTotal

Report

- Read more about adversarial malware evasion attacks
- Observed that many attacks assume unrealistic knowledge about the classification system or are ineffective at or incapable of producing functioning adversarial examples
- Existing black box, practical adversarial evasion attacks generally use “easy” classifiers
- Interesting to see how their techniques hold up against a more realistic defender, like an ensemble or a commercial AV product

Minutes

-

W4: 09/12/22 - 09/19/22

General

- Concerning swapping blocks of suspicious code with less suspicious but equivalent alternatives, the AST approach might be useful because it represents the actual functionality of the code and code produce code that functions the same, but uses weird sequences of tokens.
- Could use code2seq to take assembly to language, then use codex to take language to assembly to get non deterministic outputs
- Interesting paper that uses trains GPT2 on malware bytecode
- Perhaps a code generation tool like codex could repeatedly nondeterministically mask and substitute chunks of assembly for adversarial evasion
- Use codex to modify the source code of malware
- Large scale "instruction substitution" in poisoning attacks?
- Using code generation techniques to change the important code itself is an extremely challenging problem. Several methods use various strategies to append some form of bytes into sections of the code. We could use code generation techniques to insert logically functional code blocks into the malware. At that point though, we might as well just copy and paste assembly from a specific source.
- Could use code generation to write C code, then compile it, and insert the compiled code into the malware. Use genetic or RL algorithms to find where to insert it, or simply insert it behind a if-false statement.
- Code summarization of a portion of assembly code via code2vec or something like it. Then code generation from the summary back into assembly.
- To create a labeled assembly language, compile C code and use function names as labels.
- Shellcode_IA32 has some code generation with codeBERT
- PalmTree is a really effective embedding setup
- Potentially use the function boundary identification to identify functions to swap
- Perform seq2seq learning trying to make the frequency of opcodes match that of goodware. Evaluate upon unit tests for C code.

Malware Group

Report

- Interested in Adversarial Malware Evasion Attacks

Minutes

- Code generation techniques to produce adversarial examples, eg, code2vec/code2seq on assembly
- Would have to generate code that functions, but doesn't appear malicious.
- Oakland paper Saidur shared in the Slack group
- Android research might be more promising than PE research
- Benign Android data is easier to attain
- AndroidZoo is a widely used dataset
- Dr. Pan will send out some papers

Reading Group

-

Weekly Scrum

Report

- Looking into using LLMs for adversarial malware generation
- Hard problem and not exactly sure what directions to focus on
- The start would be with a language model at the assembly or binary level
- The state-of-the-art seems to be PalmTree (Pre-trained Assembly Language Model for InsTRuction EmbEdding), which uses a BERT architecture with different training tasks adopted for code
- This gives us really good embeddings for assembly instructions
- BERT has been adopted for sequence tasks in natural language and in code tooling, so there could be some extensions made to PalmTree to get interesting things going at the assembly level
- Took a little bit of wandering to find PalmTree and other binary analysis tools
- This weekend, will develop more concrete ideas and write up that abstract

Minutes

- Injecting code underneath if false blocks probably not a good direction to take
- Probably need to perform some kind of function segmentation

W5: 09/19/22 - 09/26/22

General

- Dr. Wright has mentioned modifying API calls...wouldn't this have to be done at the source code level? I let my knowledge of Operating Systems and Computer Architecture fade a little...
- Using code2code to modify the byte histograms and byte-entropy of EMBER might be successful
- Explainability of EMBER
- Microsoft Malware Classification Challenge (MMCC) is a multiclass classification challenge. This could be an alternative to benign/malicious classification because it alleviates the challenges associated with gathering benign data.
- Function identification at the assembly level to build monolingual corpora

Malware Group

Report

- Worked out a research objective for the remainder of the semester
- Wright/Pan talked about replacing suspicious sections of code as part of an adversarial attack
- Should be able to use sequence to sequence models, similar to machine translation or transcompilation
- Related work:
  - LLM for assembly code prior to fine-tuning for downstream tasks
  - Generating assembly-level code from English descriptions
- I think we can use a pre-trained LLM encoder-decoder architecture to perform the translation
- Things that remain to be worked out are:
  - What makes assembly code look suspicious vs benign?
  - How to actually train the model, eg supervised/unsupervised and what is the objective?
- What I need to do:
  - Learn more about sequence to sequence models, neural machine translation, and transcompilation
  - Learn more about the properties of malware vs benignware
- Other:
  - Produce abstract from these ideas
  - Presentation tomorrow remotely

Minutes

- Features of malicious code:
- Look at sequence of API calls. Here's what we see in ransom ware. Could just start opening up other files in the API calls. Default approach: run a model on the malware and figure out what is important. Or use gradient based methods to figure out what is important.
- Statically we know what API calls are imported and control flow graphs. Dynamically we can get the sequence of API calls.
- Histogram of byte sequence frequencies

- Substitute API calls
- Find API calls that can be substituted
- Identify certain functions/areas that are malicious looking
- Could use histogram/heat map to identify those regions
- Gadgets:
- **using explainability, figure out what are chunks of code flagged for maliciousness, then construct a number of replacement sequences. Dumb version is to just generate some kind of replacement and test to see if it works. Smart version is to replace bit by bit and monitor the gradient of the loss function. Then follow the loss function greedily.**
- Maybe don't worry about making the malware runable
- Identify code and replace with code that has the same functionality
- **One identify the functions to be changed by explainable. This by itself could be a very interesting problem. See if this has already been done.**
- Saidur will share some papers about histograms

Reading Group

- Yin Pan complemented my presentation skills
- Could have been better prepared for some of the results sections
- Could have worked out a method of having a few notes

Weekly Scrum

Report

- Looked into using explainability to decide which regions of malware code could be modified
- Started with EMBER classifier
  - Can precisely determine the importance of all features used by the GBT model
  - Saidur's discussed using byte histograms as a classification feature
  - Between 7%-12% of EMBER's decision making process can be attributed to these features
  - Simply manipulating these properties could work on EMBER
- Byte-stream classifiers (MalConv)
  - A few papers investigate methods to observe, which portions of the malware trigger high gradient activation, ie, they are important
  - Regions of high activation seem to be distributed throughout the malware file, which means that an attack made only to the code could still be effective
  - Next paper on my plate uses explainability of MalConv as part of an attack
- Dr. Wright brought up some ideas about API calls...wouldn't those modifications need to be performed at the source code level?
- I find this term API call rather nebulous...could you give a specific example

Minutes

-

Malware Group

Report

- Code Generation Attack
  - Related work to determine which regions within malware MalConv learns the most from
  - Sources differ on whether gradient activation is highest within headers
  - Unknown why classifier identifies the regions, which could help generate benign-looking code
  - Alternatively, could turn to machine translation techniques
    * Frame problem as have source and target languages
    * Source language is malicious-looking code; target is benign-looking code
    * Use unsupervised NMT methods that require no parallel data
    * Denoising Auto Encoder: reconstruct source from noisy source input
    * Backtranslation: use source-to-target and target-to-source models trained in unison (similar to GANs)
    * Inspired by Backtranslation: On the Fly Back Translation, Cross Domain Training
    * Summarize and Generate: source code to NL, NL to target code
  - MT is generally considered a simpler task than generation, so this might be helpful approach
  - Wouldn't Dr. Wright's discussion of API calls need to take place at the level of source code?
- After reading many papers analyzing MalConv, I started thinking about how it could be improved
- Enhancing MalConv
  - Went on a tangent for the past several days
  - MalConv learns an 8 dimensional embedding layer during training
  - Interested in using pretrained embedding layer
  - Train MalConv-like classifier on assembly instead of raw bytes
  - Maintains the spirit of MalConv
  - Major performance enhancements made to MalConv in 2021 improve memory usage and training time
  - Pretrained embeddings could yield a higher performing MalConv with less training data
  - Idea is less novel, but more concrete

Minutes

- Look into how to do this stuff with Android
- Yin will send a paper on obfuscation Android stuff
- Wright is not interested in using pretrained embeddings in malware detection
- Old student built something that moved from machine code to IR to machine code
- Potentially incorporate some of the automatic unit testing stuff
- Will have a discussion with Dr. Wright on Tuesday/Thursday

Week 6

11:00 Monday 09/26/22 - 10:59 Monday 10/03/22

General

- Which citation should I grab when multiple citations exist?

Reading Group

- Canceled

One on One with Dr. Wright

- Unit Test Creation
  - Could manually make unit tests for a number of malicious chunks of code
  - Look into possible automatic unit testing frameworks for assembly
  - Then when swapping them with benign-looking variants, could test to verify that they work
  - The hope would be that the malicious-looking chunks are a feature of many malware samples, so the unit test creation would not have to be repeated
- Unit Test Mapping
  - Could also use open source code with available unit tests
  - build embeddings for chunks of code in malware
  - Find unit-tested code that has the same embedding!

Weekly Scrum

Report

-

Minutes

-

Malware Group

Report

- Working on building non-parallel monolingual corpora for translation approach
- My skills in Pytorch are somewhere in between novice and intermediate
- Burned a lot of time making little progress due to stupid hangups, eg, beta versions of Captum, torchtext, torchvision, ambiguous documentation of third part dependencies, bugs in research codebases
- Where I stand right now:
  - Have functioning pretrained CNN classifier
  - Have SOREL-20M PE Malware dataset
- What I'm working on
  - Figuring out how to use Captum explainability

Minutes

- We need some sort of proof-of-concept that the translation is possible
- Malware authors modify the code in extremely bizarre ways to fool malware analysts
- Try to find some example of obfuscated malware that triggers a classifier; substitute it with code that makes the classifier less confident
- For RPA
  - Clear understanding of the literature
  - Understanding of how to advance the state of the art
  - Preliminary results

Week 7 & 8

11:00 Monday 10/03/2022 - 10:59 Monday 10/10/2022

General

-

Reading Group (Tuesday)

-

Weekly Scrum (Thursday)

Report

- Need to demonstrate that we can rewrite pieces of malware to make them look less like malicious
- To do this, I plan to measure the importance of a chunk of code, rewrite it, and then remeasure it
- Goes back to using explainability to find malicious looking chunks of code, which has not gone smoothly
- Selected LowMemMalConv/MalConvGCT because I have pretrained models
- Spent several trying to apply integrated gradients, which I now believe is not possible
- Can work around the embedding layer, but MalConv2 has other complex non-differentiable layers
- Working on perturbation-based explainability methods, eg Feature Ablation, Shapley Value Sampling
- Complexity has a linear relationship to the number of features, which for the raw-byte classifiers is large
- Plan on running that code over the weekend, hopefully over an ssh connection from New Jersey
- May want to write and train a simple raw-byte classifier soon so I can use the gradient methods
- This brings up challenges associated with acquiring benign training

Minutes

-

Malware Group (Monday)

Report

- Have results of several explainability experiments on eight malicious/benign executables
  - Feature Perturbation
  - Feature Occlusion
  - Occlusion
  - Shapley Value Sampling
- Have file offset positions of highly relevant regions of malware
- Working with IDA to modify them
- Dr. Pan has mentioned ASTs a few times. At the moment, I'm keeping structured representations of code in mind for later stages of my project, unless you think they would be extremely useful at this moment?
- Have access to AndroZoo dataset.
- Working on analyzing executables with Cuckoo Sandbox prior to modifying. Is this still the state of the art software for this task? Cuckoo 2.x only supports Python 2.7 at the moment, but a full rewrite is in the works to a Cuckoo 3.x series.

Minutes

-

Week 9

11:00 Monday 10/17/2022 - 10:59 Monday 10/24/2022

General

- partway through cuckoo sandbox setup
- left off at Preparing the Host ¿ Configuration

Reading Group (Tuesday)

- Machine Unlearning, Saidur presentation

Weekly Scrum (Thursday)

Report

- Canceled

Minutes

-

Malware Group (Monday)

Report

- Previously was working on producing a proof of concept example demonstrating that malware portions could be rewritten to reduce how suspicious they look
- Ended up getting sidetracked from that objective and I went ahead and produced nonparallel corpora of malicious-looking and benign-looking code snippets
- Several weeks ago I think I misspoke in one of our Thursday meetings about SHAP, differentiability, and MalConv; I used SHAP values to identify the malicious and benign regions of malware
- Started working on a sequence to sequence model to ingest x86 instructions and regurgitate x86 instructions
- Goal would be to build this up into an unsupervised translation model
- Had a slightly confusing email chain with Dr. Pan, could you clarify what you mean by workable malware?

Minutes

-

Week 10

11:00 Monday 10/24/2022 - 10:59 Monday 10/31/2022

General

-

Reading Group (Tuesday)

-

Weekly Scrum (Thursday)

Report

- Trying to modify malware's .text section to flip classifier's prediction
- Tried substituting with
  - random bytes
  - padding value
  - benign-looking bytes from the same executable
  - a benign executeable's entire .text section
- These changes have caused only minor reduction in the classifier's confidence, e.g., .9999 to .9990
- Possible issues include
  - Model hugely over confident
  - Model trained on small collection of data and non-diverse benign data
  - Using only first 1MB of the malware to make everything run faster
  - Modifications made to processed input tensors instead of the raw binary
  - Explainability methods not working well due to complexity of model
- TODO:
  - Carefully analyze my code for sources of error
  - Perform the modifications in the file space instead of the feature space
  - Reevaluate the importance of the different sections of the malware file
  - Consider training a simpler CNN model, using a more complete dataset, or using entire malware file
- Tool to open up a PE executable in an IDE-like environment and start typing x86 instructions?

Minutes

-

Malware Group (Monday)

Report

- Sequence to sequence model spits out x86 instructions (progress on boilerplate NLP, e.g., tokenization)

Minutes

-

Week 11

11:00 Monday 10/31/2022 - 10:59 Monday 11/06/2022

General

- Idea: clustering malware based upon PalmTree embeddings for classification

Report

- I am identifying the .text section location correctly.
- Issues still exist with certain files that may have been packed.
- For now I'm simply avoiding those files.
- Reason was not able to reduce classifier uncertainty in previous reports was that the benign executable I was performing swaps with was actually a false positive, so its .text section was malicious-looking.
- Experiment:
  - 1424 true positive malware test files
  - Record initial classifier confidences
  - Substitute .text section with benign executable
  - Mean reduction in confidence was 49%
  - Able to flip the prediction on 48% of malware files

Minutes

- Proof on concept demonstrated sufficiently
- Could test on multiple malware files or use the other MalConv classifier

Supplementary Material

Experiment:

- Trained MalConv2 classifier on Windows System32 benign PEs and SOREL-20M PE malware .
- Conducted explainability & swapping experiments with 1500 PE malware files.

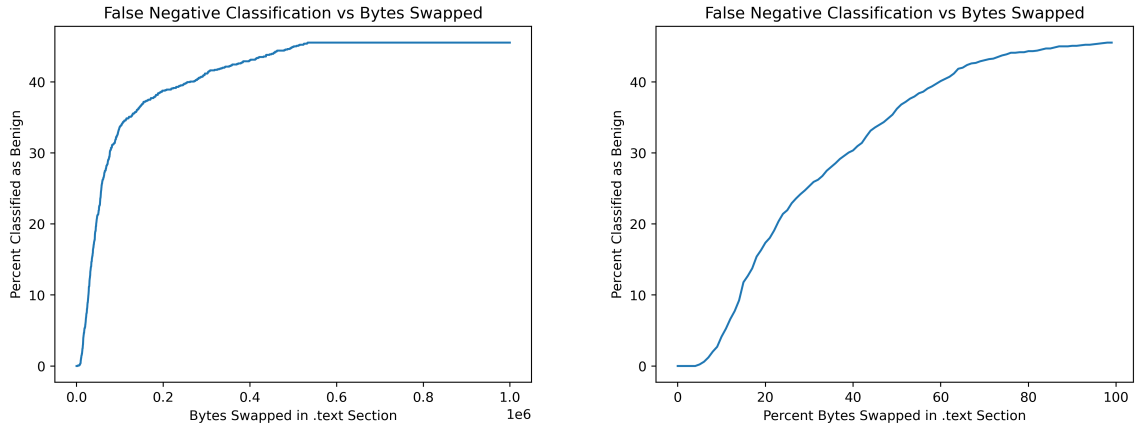
Potential Shortcomings:

- Using a malware classifier I trained myself on a smallish dataset.
- Only performing swapping on files that don't cause me any problems. There are weird artifacts in the malware files that cause bizarre errors in my pipeline. Probably packing or other obfuscation techniques, but I'm still working on figuring it out.
- These experiments taking a long time to run, so I'm taking these shortcuts to get results in a reasonable amount of time.

Incremental Swap

Algorithm:

- Use SHAP explainability method to identify which regions of malware are the most malicious-looking
- Identify the most suspicious 256 byte chunk in the malware's .text section
- Replace this chunk with a corresponding chunk (same offset within the .text section) from a benign file
- Measure classifier's confidence and prediction on modified malware
- Repeat until all bytes have been replaced



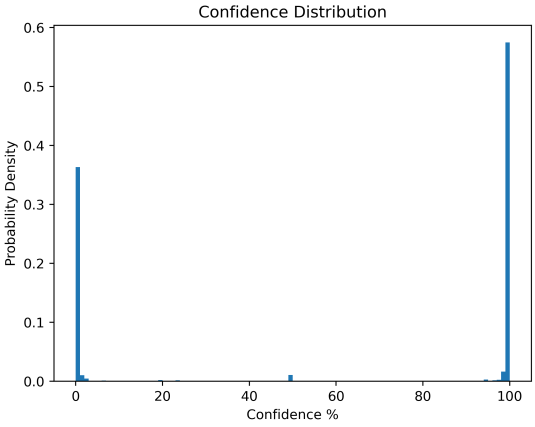
(a) Number of bytes swapped.

(b) Proportion of bytes swapped.

Figure 1: The false negative rate as more bytes from the .text section are replaced with bytes from a benign file. This is an average over 1500 malware files.

Analysis:

- The left-hand figure is much sharper than the right-hand figure
- This is because the size of the .text sections of the malware files is severely skewed right (many malware files with small .text sections; few malware with large .text sections)
- These results indicate that a significant proportion of the bytes in the .text section would have to be altered to successfully evade a classifier
- However, results in the next section seem to indicate that higher evasion rates can be achieved if the replacement bytes are better selected



h

Figure 2: Distribution of confidence scores.

Full Swap

Algorithm:

- Identify the .text section of malware
- Replace entirely with bytes
  - Random bytes - random values
  - Padding byte - the padding byte value used by the classifier
  - Benign bytes - the .text section from one of several different benign files. Benign files 0, 1, & 2 are from the training set; benign files 3, 4, & 5 are unseen by the classifier.
- This is significantly faster than doing it incrementally, so I have a bit more flexibility in how I do things
- There are actually more malware files in this experiment and some of the files are an order of magnitude larger than the previous experiment, which is why some of the results are slightly different
- This classifier is unsurprisingly extremely over confident, see Figure 2

benign file	Confidence (%)
0	9.90 e-04 %
1	3.37 e-04 %
2	98.9 e-00 %
3	1.67 e-04 %
4	4.36 e-00 %
5	4.98 e-03 %

method	Average difference in confidence (%)	TP examples flipped to FP (%)
random	-0.35%	0%
padding	-0.14%	0%
benign file 0	-49%	48%
benign file 1	-68%	69%
benign file 2	-1.1%	0.76%
benign file 3	-45%	43%
benign file 4	-17%	13%
benign file 5	-64%	63%

Analysis:

- Replacing with random bytes or the padding byte was not effective

- Using different benign files as the swapping bytes can have a huge impact on the number of successful adversarial examples

Week 13

11:00 Monday 11/13/2022 - 10:59 Monday 11/20/2022

General

- CUDA_VISIBLE_DEVICES=2 python script.py

Report

-

Minutes

-