

Research Notes

Luke Kurlandski

November 22, 2022

Efficient Estimation of Word Representations in Vector Space

Metadata

Authors: Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean

Published: 2013

Read: 04/2022

Reaction

- word2vec

Further Reading

-

Notes

-

Distributed Representation of Words and Phrases and their Compositionality

Metadata

Authors: Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, Jeff Dean

Published: 2013

Read: 04/2022

Reaction

- word2vec

Further Reading

-

Notes

-

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Metadata

Authors: Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

Published: 2018

Read: 04/2022

Reaction

- BERT
- contextual word embeddings can differentiate between different meanings of words

Further Reading

-

Notes

-

Multi-task Learning based Pre-trained Language Model for Code Completion

Metadata

Authors: Fang Liu, Ge Li, Yunfei Zhao, Zhi Jin

Published: 2020

Read: 07/2022

Reaction

- Similar to BERT but for code

Further Reading

-

Notes

-

code2vec: Learning Distributed Representations of Code

Metadata

Authors: Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav

Published: 2019

Read: 07/2022

Reaction

- Similar to word2vec but for code

Further Reading

-

Notes

-

code2seq: Generating Sequences From Structured Representations of Code

Metadata

Authors: Uri Alon, Shaked Brody, Omer Levy, Eran Yahav

Published: 2019

Read: 07/2022

Reaction

- Builds upon code2vec

Further Reading

-

Notes

-

Attention is All You Need

Metadata

Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin
Published: 2017
Read: 07/2022

Reaction

- Original Transformer paper

Further Reading

-

Notes

-

Structural Language Models of Code

Metadata

Authors: Uri Alon, Roy Sadaka, Omer Levy, Eran Yahav

Published: 2018

Read: 08/2022

Reaction

-

Further Reading

-

Notes

Introduction

- Focuses on any code completion, i.e., predicting a missing code segment from surround code
- Differs from other code completions in that it is unrestricted and more general purpose
- Introduces Structural Language Modeling (SLM) - estimates the probability of a program's AST
- Uses joint modeling of source and target code, rather than multimodal encoders and decoders

Code Generation as Structural Language Modeling

- Model probability of a program, analogous to probability of a sentence in LMs
- AST should allow the model to generalize better between languages
- Computes the probability of an AST via a DFS traverse and the chain rule
- Essentially ensures the target node for completion is last in the DFS traversal, so the entire known sequence can be modeled
- Models a partial tree as a set of paths: the paths from every leaf to a_t and the path from the root to a_t
- Uses EOS_{tok} and EOS_{node} to end the sequence of token/node generation which controls the depth and breadth of the ASTs generated
- Decomposes terminal nodes into a sequence of terminal nodes by splitting the node into subtokens, e.g., toLower-Case becomes to...lower...case...EOS_{tok}

Model Architecture

- Each AST node is represented using three embedding matrices: subtoken, type, and index
- The entire AST path is encoded using an LSTM whose final states are extracted
- Multiple AST paths are aggregated into a single vector representation
- Uses a copy mechanism to assist with predicting because programs often have repetition

Related Work

- Frames code generation as predicting the next node in all partial AST paths, so generalizes many other previous works

Automated malware detection using artifacts in forensic memory images

Metadata

Authors: Rayan Mosli, Rui Li, Bo Yuan, Yin Pan

Published: 2016

Read: 08/2022

Summary

- Investigates a heuristic approach to malware detection already active on the system.
- Trains classifiers on three features extracted from memory images: registry activity, imported libraries, and API function calls.
- Uses malware samples gathered from VirusShare.

Thoughts

-

Cited

- Heuristic machine learning malware detection methods:
 - “Analysis of malware behavior: Type classification using machine learning” (2015)
 - “Analysis of features selection and machine learning classifier in android malware detection” (2014)
 - “Malicious behavior detection using windows audit logs” (2015)
 - “Amal: High-fidelity, behavior-based automated malware analysis and classification” (2015)
 - “Three- phase behavior-based detection and classification of known and unknown malware”

Cited By

-

Notes

Introduction

- Investigates three feature types from memory images: registry activity, imported libraries, and API function calls
- Traditional approaches include signature scanning and monitoring running malware
- Memory images and memory forensics involve examining a memory dump
- Uses tools like Memoryze, WinPMEM, Volatility, and Rekall

Related Work

- Current malware detection based on static and dynamic malware features
- Memory-based detection has several advantages, such as 1) compressed malware must decompress in memory, 2) artifacts can be extracted without a VM, 3) can be used to detect memory-only malware

Methodology

- Used Tfidf vectorization of the registry activity, API calls, and DLL imports
- Used grid searching to trim unimportant features

A Behavior Based Approach for Malware Detection

Metadata

Authors: Rayan Mosli, Rui Li, Bo Yuan and Yin Pan

Published: 2017

Read: 08/2022

Summary

- Handles are pointers used to access system objects that must be opened and closed.
- This work uses handles as a heuristic measure to identify malware with machine learning.
- Uses malware samples gathered from VirusShare.

Thoughts

-

Cited

-

Cited By

-

Notes

- Signature based detection and machine learning from static malware features are both easily defeated by anti detection techniques
- Behavioral based detection is more sensitive, but requires running the malware in sandbox environment
- Handles are pointers used to access system objects without knowing their location in memory
- Handles must be opened and closed
- This approach uses the number of handles accessed to assess if a program is malicious or not
- Section handles and process handles were major indicators of malware

A General Path-Based Representation for Predicting Program Properties

Metadata

Authors: Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav

Published: 2018

Read: 07/2022

Summary

- One of the great challenges in code analysis tasks is determining an effective representation of code for learning.
- Instead of treating source code as a stream of tokens, this work represents source code using paths in its abstract syntax tree.
- This approach is more general than sequence approaches and less language specific.

Thoughts

-

Cited

-

Cited By

-

Notes

-

A systematic Evaluation of Large Language Models of Code

Metadata

Authors: Frank F. Xu, Uri Alon, Graham Neubig, Vincent Josua Hellendoorn

Published: 2022

Read: 08/2022

Summary

- Performs a systematic evaluation of several open/closed source large language models of code.
- Produces a new model, PolyCoder, intended to be multi-lingual, but does not achieve high performance.

Thoughts

- Does not mention code2vec or code2seq, which is unusual because Uri Alon is a co-author.

Cited

- “Evaluating large language models trained on code” (2021) Codex code model
- “Language models are few-shot learners” (2020) GPT-3 code model

Cited By

-

Notes

Introduction

- Most of the top performing code language models are not publicly accessible
- Multilingual vs monolingual LMs
- Evaluates the LMs: Codex, GPT-J, GPT-Neo, GPT-NeoX, and CodeParrot
- Trains the first code LM on multiple programming languages (PolyCoder)

Related Work

- Pre-training methods: left-to-right, masked, and encoder-decoder LMs
- Left-to-right autoregressive LMs predict the probability of a token given previous tokens and are particularly useful for code generation tasks
- Masked LMs predict the probability of a token given its context and can provide more powerful modeling of an entire sequence’s representation
- Encoder-decoder LMs use an encoder to map an input sequence using some form of pre-training objective, then a left-to-right LM decoder and are effective at sequence to sequence.
- The main code datasets are can be pure source code or a mixture of code and natural text (the Dump)

Evaluation Settings and Models

- Extrinsic evaluation performed with code generation
- Intrinsic evaluation performed with perplexity
- Deduplicated training source code files using hashes

Results and Conclusions

- PolyCoder tends to be outperformed in the extrinsic evaluations
- For the C language, PolyCoder has the lowest perplexity (good)
- Perplexity trends closely with the HumanEval benchmark and is probably a good, efficient metric for evaluation
- Generally, larger models perform better, but Codex performs disproportionately well

A Comprehensive Review on Malware Detection Approaches

Metadata

Authors: Ömer Aslan, Refik Samet

Published: 2020

Read: 08/2022

Summary

- This paper provides an overview on malware detection approaches, including: signature, behavior, heuristic, model checking, deep learning, cloud, mobile devices, and IoT-based malware detection.

Thoughts

- Research combining deep learning and malware detection is relatively novel.
- Contains information about some of the widely-used malware datasets.

Cited

- “Deep neural network based malware detection using two dimensional binary program features” (2015)
- “MtNet: A multi-task neural network for dynamic malware classification” (2016)
- “Droid-Sec: Deep learning in Android malware detection” (2014)
- “Adversarial malware binaries: Evading deep learning for malware detection in executables” (2018)
- “Effective Android malware detection with a hybrid model based on deep auto encoder and convolutional neural network” (2018)
- “MALDC: A depth detection method for malware based on behavior chains” (2019)

Cited By

-

Notes

Introduction

- Signature-based detection cannot detect unknown/novel malware species
- Behavioral, heuristic, and model-based detection are being used

Problem Definition

- Malware detection is NP-complete, so no method can be perfect
- Obfuscation techniques include: encryption (in various levels of complexity, e.g., oligomorphic/polymorphic), metamorphic (dynamic code hiding), stealth (difficult to analyze correctly), and packaging (compressed and encrypted malware)

Malware Detection Techniques and Algorithms

- Three stages of malware detection: malware analysis, feature extraction, and classification
- Malware analysis answers how the malware works, what is affected
- static vs dynamic analysis, both begin with reverse engineering
- Feature extraction is most commonly done with ngrams or graph-based models
- NGrams builds and ngram model out of system calls or API calls
- graph-based model builds graph with system calls as vertices and relationship between them as edges

- Datasets:
 - NSL-KDD
 - Drebin
 - Microsoft malware classification challenge - requires feature extraction
 - ClaMP
 - AAGM
 - EMBER
- a variety of ML algorithms have been shown to be successful (no mention of DL in this paper)

Malware Detection Approaches

- Signature based detection is being supplemented with behavior, heuristic, and model-checking techniques
- Signature Detection
 - Generate a signature from malware executable via a variety of methods: string scanning, top and tail scanning, entry point scanning, and integrity checking
 - All of the above methods are somewhat vulnerable to basic obfuscation techniques
 - More advanced methods exist
- Behavior Detection
 - Runs the malware and examines its behaviors, although some malware will not run in protected sandbox environment, thus will be marked as benign
 - Three steps: determine behaviors, extract features of behaviors, apply classification
- Heuristic Detection
 - Uses a variety of techniques and established rules to detect malware, but is prone to false positives
- Model-Checking Detection
 - Uses a variety of techniques and established rules to detect malware, but is prone to false positives
- Deep Learning Detection
 - Powerful, but can be fooled by evasion techniques
 - Little-researched topic
- Cloud Detection
 -
- Mobile Device Detection
 -
- IOT Detection
 -

Evaluation on Malware Detection Approaches

- Signature, behavior, heuristic, and model-based techniques are well studied
- DL, cloud, mobile, and IoT techniques are new

Malware Classification with Recurrent Networks

Metadata

Authors: Razvan Pascanu, Jack W. Stokes, Hermineh Sanossian, Mady Marinescu, Anil Thomas

Published: 2015

Read: 08/2022

Further Reading

- “A survey of malware detection techniques” (2007)
- “Large-scale malware classification using random projections and neural networks” (2013)
- “A biologically inspired immune system for computers” (1994) the first malware detection system

Summary

- Uses RNNs to model the API calls made by malware programs and passes the malware embeddings to classifiers.
- Presumably, takes a dynamic approach and runs the malware in a VM to attain the API calls.
- Uses a proprietary Microsoft malware dataset.

Thoughts

- The base model could be extended to a transformer-based model.

Cited

-

Cited By

-

Notes

Introduction

- Handcrafted feature selection approaches are often not robust to obfuscation
- RNN learns the language of malware, i.e., LM, by predicting the next API call
- The hidden state of the RNN is extracted and used as a feature vector, then fed to a different classifier
- Uses Max Pooling over values of hidden units in time (future works will experiment with Attention)
- Uses a Bi-directional model, i.e., two separate models one working left to right the other right to left

Algorithm Description

- Max Pooling and Half Frame address limited memory window of RNN and ESN
- Max pooling selects the maximum hidden state output for each sequence. The sequence representation is the concatenation of the last state and the max pooling layer.
- Half Frame uses the hidden state from the middle of the sequence and the hidden state from the end of the sequence

Experimental Results

- Uses 114 high level events as initial features
- Compared to bag of words-based models (unigram and trigram)

Malware classification with LSTM and GRU language models and a character-level CNN

Metadata

Authors: Ben Athiwaratkun, Jack W. Stokes

Published: 2019

Read: 08/2022

Summary

- Malware is dynamically analyzed via emulation and a sequence of system calls is extracted.
- The system calls are modeled using LSTMs and GRUs before feeding embeddings to classifiers.
- Proposes a novel end-to-end character-level CNN, which does not perform as well as the two-stage process.
- Uses a proprietary Microsoft malware dataset.

Thoughts

- Creates embeddings for the system calls extracted using dynamic analysis.
- Doesn't model malware language so much.

Cited

- "Malware classification with recurrent networks" (2015) the foundation and baseline for this work
- Deep learning for malware classification:
 - "Large-scale malware classification using random projections and neural networks" (2013) uses system calls
 - "Deep neural network based malware detection using two dimensional binary program features" (2015)
 - "Mtnet: A multi-task neural network for dynamic malware classification" (2016)
 - "Visualized malware classification based-on convolutional neural network" (2016)

Cited By

-

Notes

Introduction

- Previous work trained an LM using RNN/ESN then used feature outputs to train a classifier
- This work uses LSTM, GRU, Attention, and CharCNN as LMs seeking to improve the performance
- Maps API calls to 113 high-level abstraction operations for features

Model

- Previous work found ESN + Max Pooling and RNN + Max Pooling were the most successful for the LM and an LR classifier for the classifier
- This work explores a LSTM and GRU for the LM with an Attention Mechanism instead of Max Pooling and MLP/LR classifiers
- They also deploy a single-stage CharCNN malware classifier forgoing the LM entirely

Conclusions

- LSTM + Max Pooling + LR outperforms other models
- Demonstrates the benefits of the semi supervised pre-training
- Demonstrates the efficacy of LMs for file stream events

Learning and Evaluating Contextual Embedding of Source Code

Metadata

Authors: Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, Kensen Shi

Published: 2020

Read: 08/2022

Summary

- Developed CuBERT (Code Understanding BERT), a pre-trained contextual Python code understanding model.
- Fine-tuned CuBERT outperformed word2vec, BiLSTM, and Transformer classification models.

Thoughts

- Concept could be applied to other transformer models, e.g., CuELECTRA.

Cited

- “A literature study of embeddings on source code” (2019)
- “CodeBERT: A pre-trained model for programming and natural languages” (2020)
- “Global relational models of source code” (2020)

Cited By

-

Notes

Introduction

- Compared against BiLSTMs and Transformers
- Classification and code repair
- Uses raw source code tokens instead of a structured format of code

Related Work

- CodeBERT targets paired natural language and programming language tasks, where NL is paired with PL

Experimental Setup

- Uses Python datasets for pre-training and fine tuning with measures taken to avoid duplication
- Fine-Tuning Tasks. Five classification and one more complex.
 - variable misuse classification: given function, classifier must predict whether there is variable misuse anywhere within the function
 - wrong binary operator: determine if any binary operator in the function is incorrect
 - ped operand:
 - function-docstring mismatch:
 - exception type:
 - variable misuse localization and repair
- Baselines: four variants of word2vec for embeddings, BERT, BiLSTM, and pre-trained BiLSTM

Results and Conclusions

- Contextual code token embeddings prove more beneficial than word embeddings
- Pre-trained + fine tuned transformer beats the plain old fine tuned transformer

Large-scale malware classification using random projections and neural networks

Metadata

Authors: George E. Dahl, Jack W. Stokes, Li Deng, Dong Yu

Published: 2013

Read: 08/2022

Summary

- Random projections reduce the typically sparse feature space of malware representation to dimensions suitable for DNNs.
- Uses dynamically extracted features from a Microsoft dataset.

Thoughts

-

Cited

-

Cited By

-

Notes

Introduction

- Malware classification also involves massively sparse feature representations
- Even after feature selection there can be too many features for NN classifiers
- This work uses random projections to further reduce the dimensionality
- Values minimizing FP ($< .001$) more than FN ($< .05$)

System

- Three features: null-terminated patterns in process memory, tri-grams of system API calls, and distinct combinations of a single system API call and one input parameter
- Feature selection to reduce the number of features to a reasonable amount, 179k
- Uses random projections to reduce feature size to a couple of thousand, but also explore PCA

Results and Conclusion

- One layer NN and One Layer NN + RBM pre-training achieved the best results
- Using around 4000 features from random projections and 768 hidden units worked best

A Literature Study of Embeddings on Source Code

Metadata

Authors: Zimin Chen, Martin Monperrus

Published: 2019

Read: 08/2020

Summary

- Provides a literature overview of embeddings for source code tokens, functions, sequences of functions, and binary code.

Thoughts

- Poor paper.

Cited

-

Cited By

-

Notes

Introduction

- Classifies articles as embeddings of a) tokens, b) functions, c) sets of function calls, d) binary code, d) other

Embeddings for Source Code

- Several works have used word2vec directly for token embeddings
- There is rooms for research in contextual embeddings for source code

Binary Black-box Evasion Attacks Against DL Static Malware Detectors with Adversarial Byte-Level Language Model

Metadata

Authors: Mohammadreza Ebrahimi, Ning Zhang, James Hu, Muhammad Taqi Raza, Hsinchun Chen

Published: 2020

Read: 08/2022

Summary

- MalRNN, appends bytes to the end of malware files to fool static detection systems.
- Discusses Adversarial Example Generation (AEG) and Anti-Malware Evasion (AME).
- Source code and dataset are supposedly available at <https://github.com/johnnyzn/MalRNN>, but were originally property of Microsoft.

Thoughts

-

Cited

- “Modeling malware as a language” (2018)
- “Non-negative networks against adversarial attacks” (2019) a type of NN which is robust against adversarial examples
- “Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only” (2018) a simple but effective static malware detection system
- “Malware detection by eating a whole exe” (2018) presumably similar to the above, but more widely cited

Cited By

-

Notes

Introduction

- DL static malware detectors have proven successful, but can be easily fooled by adversarial examples
- To successfully launch an adversarial malware attack, an attacker needs knowledge about the anti malware system
- MalRNN can generate adversarial examples without knowledge of the anti malware system
- DL malware detectors can consume raw malware binary as input and automatically extract features
- These detectors are vulnerable to Adversarial Example Generation attacks (AEG)
- Anti Malware Evasion (AME) seeks to use these adversarial examples to improve the model
- One common technique is to append bytes to the end of malware, bytes which are never executed and only confuse the anti malware system
- This technique learns a language model on binary executables and uses them to generate benign-looking content for AME without knowledge of the anti-malware system

Related Work

- AEG comes in four forms
 - white box - adversary has full access to attack target
 - grey box - the attacker has access to features that are important to the classifier but parameters of NN detector not available

- black box - adversary has no access to attack target specs, features, or parameters, but it can obtain feedback from the attack (confidence score)
- binary black box - receives a binary response instead of confidence score
- Binary Black Box is the most common in real world applications
- A lot of AME research uses a black box threat model, which aren't terribly realistic to real-world applications
- Most binary black box studies target signature-based systems, thus are not hugely relevant
- GRU alleviates the vanishing gradient problem with RNNs; sequence to sequence models allow the RNN to map different sized inputs and outputs

Proposed Method (MalRNN)

- goal is to evade the malware system in a binary black box threat scenario
- MalRNN essentially injects binary sequences into binaries in an attempt to fool the classifier.
- Injects them at the end of the binary to preserve functionality
- The LM is learned upon benign binaries, so it can learn to mimic them
- binary executables were converted in Hex
- Character level sequence to sequence with gated recurrent units
- encoder decoder RNNs

Experiment and Results

- Compare with three well-known anti malware detectors
- Compare with three other methods of appending bytes to the end of malware

Modeling malware as a language

Metadata

Authors: Yara Awad, Mohamed Nassar, Haidar Safa

Published: 2018

Read: 08/2022

Summary

- Creates a “malware language” from disassembled malware binary opcode.
- Models the language using word2vec and performs classification with kNN.
- Uses a private Microsoft dataset.

Thoughts

- With a larger dataset, could these techniques be used to learn a BERT understanding model?

Cited

- “Malware Analysis and Classification: A Survey” (2014) machine learning techniques for malware classification
- “Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A state-of-the-art survey” (2009) machine learning techniques for malware classification

Cited By

- “Recurrent neural network for detecting malware” (2020) various features representation including word2vec rep with RNN classifier
- “Automatic Malware Clustering using Word Embeddings and Unsupervised Learning” (2019) categorizes different malware species using word2vec embeddings from system calls
- “Transfer Learning for Malware Multi-Classification” (2019) evaluates the potential for transfer learning by extending a binary classifier to a multiclass task

Notes

Introduction

- Signature and heuristic based approaches only improve after unknown malware infects a system
- Traditional ML based approaches involve feature extraction and classification
- Dynamic analysis can still be hindered
- This work explores static analysis via language modeling

Related Work

- NB classification on strings within malware, treating malware as an image vector and computing euclidian distance, analyzing function call graphs
- Lots of work done using ngrams methodologies

Methodology

- Defined a malware language with concepts of what a vocabulary, word, and document are
- Two malware languages, one where words are single assembly instructions and one where words are the instruction and an abstraction of the operands
- Transform assembly code to conform to malware language
- Model transformed instances using word2vec model into matrices

- Uses kNN and word mover's distance (WMD) algorithm to perform classification

Results and Conclusions

- Future work should investigate the efficacy of RNNs and LSTMs

MalBERT: Using Transformers for Cybersecurity and Malicious Software Detection

Metadata

Authors: Abir Rahali, Moulay A. Akhloufi

Published: 2022

Read: 08/2022

Summary

- Extracts an AndroidManifest.xml metadata file from binary files with Jdax (statically/heuristically).
- Uses BERT to model contents of this metadata file.
- Uses the publicly available AndroZoo dataset.

Thoughts

- Does not actually use BERT to model the source code.
- Terribly written.

Cited

- “A malware detection method based on sliding local attention mechanism” (2020)
- “Robust malware detection using residual attention network” (2020)
- “Ransomware classification using patch-based cnn and self-attention network on embedded n-grams of opcodes” (2020)
- “Malware analysis of imaged binary samples by convolutional neural network with attention mechanism” (2018)
- “Androzoo: Collecting millions of android apps for the research community” (2016) Android malware and benignware dataset

Cited By

-

Notes

Introduction, Background, and Related Work

- Malware detection using BERT, first transformer based method
- Most sequence to sequence tasks use encoder decoder with two RNNs
- Transformers can be more efficiently parallelized and can capture long-range dependencies within sequences than RNNs
- Mentions BERT, distilBERT, RoBERTA, and XINet
- Several works use DNNs and attention mechanisms to classify malware

Methodology

- Uses Androzoo public malware dataset

Recurrent neural network for detecting malware

Metadata

Authors: Sudan Jha, Deepak Prashar, Hoang Viet Long, David Taniar

Published: 2020

Read: 08/2022

Summary

- Uses an RNN classifier with three feature representations: one hot encoding, random vector representation, and word2vec embeddings from disassembled malware opcode.

Thoughts

- A transformer or a LSTM model would probably work better..
- Badly written

Cited

-

Cited By

- “MalBERT: Malware Detection using Bidirectional Encoder Representations from Transformers” (2021) a second ‘MalBERT’ paper with a different MalBERT
- “XLCNN: pre-trained transformer model for malware detection” (2022)
- “Static Malware Detection Using Stacked BiLSTM and GPT-2” (2022)
- “Malware Detection Using Transformers-based Model GPT-2” (2021)

Notes

Introduction

- Uses an RNN classifier with three feature representations: one hot encoding, random vector representation, and word2vec embeddings
-

Related Work

-

Methods

-

Results and Conclusions

-

Malware detection with LSTM using opcode language

Metadata

Authors: Renjie Lu

Published: 2019

Read: 08/2022

Summary

-

Thoughts

- Used a strange dataset, 1000 malware and 100 benign...work would be better if a larger set used

Cited

-

Cited By

-

Notes

Introduction

- Use IDA Pro to disassemble malware into assembly, then
- Extract opcode sequences from each assembly format file, then
- Use word embeddings to learn features from opcode

Malware Detection Methodology

- Data processing
 - Extracts program instructions from the .text segment of the assembly file
 - Extracts opcode sequence using algorithm that ignores meaningless opcodes
- Opcode Representations in Vector Space
 - Filter out infrequent opcodes to build a opcode vocab of 391
 - Convert opcode sequence into one-hot vector...why?
 - Use Gensim to get feature vector for opcode
- Feature Representation by LSTM
 - Two stage LSTM can handle opcode sequences of different length
 - Mean pooling layer after second LSTM

Results and Conclusions

-

Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo

Metadata

Authors: Aparna Sunil Kale, Vinay Pandya, Fabio Di Troia, Mark Stamp

Published: 2022

Read: 08/2022

Summary

- Experiments four different word embeddings (HMM2vec, word2vec, ELMo, and BERT) and four different classifiers (SVM, kNN, RF, and CNN) on static malware opcodes for malware classification.
- Finds that english BERT fine-tuned on malware opcode yields the best feature representation.

Thoughts

- Concatenates word embeddings, when they should be added or averaged together.
- Perhaps word2vec's poor performance is explained by this choice.
- A structured approach such as code2vec should be considered.

Cited

- "Malware classification with word embedding features" (2021) the inspiration for this work
- "PE header analysis for malware detection" (2018) dataset of malware families
- "Malware detection using machine learning based on word2vec embeddings of machine code instructions" (2017) malware classification using word2vec embeddings and CNNs
- "Malware detection using dynamic birthmarks" (2016) malware classification using dynamic features rather than static ones

Cited By

-

Notes

Introduction

- ML models use features such as API calls, opcode sequences, and byte n-grams
- This work applies word embedding techniques to opcode sequences before classification with a variety of classifiers
- Embedding techniques include: BERT, ELMo, HMM2Vec, and word2vec
- Generates embeddings from mnemonic opcodes
- Opcode calls can be obtained through static analysis while API calls require dynamic analysis

Implementation

- Used seven malware families from a larger malware dataset
- Extracts mnemonic opcode sequence from .exe malware sample, then uses objdump to generate .asm files from which opcode sequences can be extracted
- Retain the $M \in \{20, 31, 40\}$ most frequent opcodes from each sequence
- For word2vec (and the other LMs) concatenates the opcode embeddings to vectorize an example
- Uses English BERT, but fine tunes on the malware opcodes with wordpiece tokenizer

Results and Conclusions

- In binary malware detection, using $M = 31$ proved to be a compromise between accuracy and runtime
- For malware family classification, BERT with SVM and kNN achieves the highest results
- Surprisingly, word2vec classification seems to perform the worst

EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models

Metadata

Authors: Hyrum S. Anderson, Phil Roth

Published: 2018

Read:

Summary

- EMBER is a large (1M) dataset composed of features extracted from binaries.
- EMBER also comes with a tool to extract features from new binaries.

Thoughts

-

Cited

-

Cited By

-

Notes

Introduction

-

Related Work

-

Methods

-

Results and Conclusions

-

SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection

Metadata

Authors:
Published:
Read:

Summary

- A dataset containing both raw binaries and fully feature-extracted examples.

Thoughts

-

Cited

-

Cited By

-

Notes

Introduction

-

Related Work

-

Methods

-

Results and Conclusions

-

Malware Detection by Eating a Whole EXE

Metadata

Authors: Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, Charles K. Nicholas

Published: 2018

Read: 08/2022

Summary

- Proposes a novel CNN-based architecture for classifying raw malware binaries.
- Requires sequence processing on unprecedented scales.
- Avoids sole emphasis on the PE-header and manually extracted features.

Thoughts

-

Cited

-

Cited By

-

Notes

Introduction

- API calls, instructions executed, IP addresses accessed are all examples of dynamic analysis
- Several unique challenges exist in classifying malware from raw binaries: bytes are context sensitive, relationships between byte sequences are not contiguous in the context of the entire binary, sequence classification with millions of time steps

Related Work

- Neural Networks for Long Sequences
- No NN architecture has been used for sequences this long
- Sequence processing usually involves predicting the next token in the sequence, which provides frequent feedback, unlike the task of binary classification
-

Model Architecture

- Contents of a portable executable (PE) binary can be rearranged in almost any order
- The MS-DOS header contains pointer to the PE-Header, which in turn contains pointers to all other portions of the executable
- CNN can capture the invariance between the various important features of the binary
- Uses an embedding layer to map bytes values to fixed length before applying convolutions

Results and Conclusions

- Required 8 GPUs (due to memory constraints) and a month of training for two million samples
- This model may have avoided learning an association between packing and malware, which will be helpful to reduce false positives
- Batch normalization led to poor performing models, because the malware binaries did not follow a normal distribution

Adversarial Examples for Malware Detection

Metadata

Authors: Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, Patrick McDaniel

Published: 2017

Read: 08/2020

Summary

- Modifies an algorithm from CV to minimally modify a malware sample's feature representation to fool malware classifier.

Thoughts

-

Cited

- "The limitations of deep learning in adversarial settings" (2016) discusses adversarial approaches to DL
- "Generating adversarial malware examples for black-box attacks based on GAN" (2017) malware approach for generating adversarial samples based upon GANs

Cited By

- "Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables" (2018)

Notes

Introduction

- ML algorithms can be fooled by minor perturbations in examples
- Malware has less options for where to make the perturbations than in other domains such as image classification
- Approach generalizes to any malware detection system using a differentiable classification function

Methods

- Uses statically determined sparse binary features with a basic MLP
- Essentially seeks to change the minimal number of bits in the feature rep until the system is fooled
- Is careful when modifying the malware to ensure that it is still functional afterwards
- Uses the Drebin Dataset

Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN

Metadata

Authors: Weiwei Hu and Ying Tan
Published: 2017
Read: 08/2022

Summary

- Proposes MalGAN, a GAN-based adversarial malware creator.
- Uses GANs to modify the bits of an API-call binary malware feature vector.
- Highly effective against several black box malware detectors.

Thoughts

- Authors used a discrete malware representation, so could not use some adversarial methods from CV.
- Adversarial malware modification for continuous feature representation using CV (continuous) methods?

Cited

- “Adversarial perturbations against deep neural networks for malware classification” (2016) the gradient based approach to generate adversarial Android malware examples
- “Generative adversarial nets” (2014) hugely cited work
- “Explaining and harnessing adversarial examples” (2014) a gradient based algorithm to generate adversarial examples
- Defense from Adversarial Attacks
 - “Towards deep neural network architectures robust to adversarial examples” (2014) auto-encoders approach
 - “Distillation as a defense to adversarial perturbations against deep neural networks” (2016) defensive distillation
 - “A general retraining framework for scalable adversarial classification” (2016) retraining
 - “Evaluation of defensive methods for DNNs against multiple adversarial evasion models” (2016) overview of defensive methods

Cited By

- “Evading Anti-Malware Engines With Deep Reinforcement Learning” (2019)

Notes

Introduction

- Malware authors usually can only perform black box or binary black box attacks
- Previous work has used gradient-based approaches, a sub model to mitigate disadvantages of black box scenario,
- Malware authors can determine some information about the features the system uses by feeding it carefully selected examples
- This work assumes malware authors can figure out the features the detection system uses

MalGAN

- Generates adversarial examples for binary API features because they are widely used
- Only adds API calls to the feature vector since removing them may crack the malware

- Uses a basic feed forward NN as a substitute detector to classify the generator
- Trains the sub detector on same data as black box detector, but to “fit” the black box detector, uses the black box detector’s predictions as labels for ground truth

Results and Conclusions

- Used data from <https://malwr.com/>
- Most adversarial techniques are for CV
- CV features are continuous, while this malware feature choice is binary
- MalGAN outperforms gradient based approaches that used a white box scenario
- Retraining the black box detector on adversarial examples diminishes the effect of MalGAN, but then malware authors could simply retrain MalGAN again

Adversarial Perturbations Against Deep Neural Networks for Malware Classification

Metadata

Authors: Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, Patrick McDaniel

Published: 2016

Read: 08/2022

Summary

- Modifies an existing gradient-based approach to generate adversarial malware in a white box setting.
- Investigates methods for hardening DNNs against adversarial malware and finds retraining to be most successful.
-

Thoughts

-

Cited

- “Practical black-box attacks against deep learning systems using adversarial examples” (2016) black-box scenario for adversarial attacks
- “The security of machine learning” (2010) broad overview of attacks against machine learning systems

Cited By

-

Notes

Introduction

- Compared to CV, malware domain has two major differences: continuous, differentiable input is replaced with discrete input, and the malware must remain functional following modification
- Uses the Drebin dataset and the gradient-based method from [17]
- Employs the following constraints on adversarial generation:
 - only add/remove features
 - preserve function of malware
 - only add a restricted amount of features, particularly to the AndroidManifest.xml file
- Explores methods of hardening DNNs against adversarial malware:
 - feature reductions
 - distillation (training with class probabilities instead of binary labels)
 - retraining

Related Work

-

Methods

- Uses sparse binary features to represent malware
- Essentially changes one bit of the feature vector at a time until the classifier is fooled
- Distillation uses probability outputs from a classifier to train a second classifier instead of class labels

Results and Conclusions

-

Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach

Metadata

Authors: Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Hao Zhu, Bo Li

Published: 2018

Read: 08/2022

Summary

- Explores defense of poisoning attacks.
- Creates an adversarial malware system and fools three state of the art detectors.
- Uses similarity-based filtering to identify potentially adversarial examples and build a more robust system.

Thoughts

-

Cited

-

Cited By

-

Notes

Introduction

- Malware authors can actually attempt to alter the training data of malware detection systems themselves
- Poisoning attack: attacker assumes control of a subset of training samples
- Introduces camouflage detection, which uses similarity analysis to attempt to identify adversarial examples
- Dataset from Contagio Mobile Website

Related Work

-

Methods

-

Results and Conclusions

-

Arms Race in Adversarial Malware Detection: A Survey

Metadata

Authors: Deqiang Li, Qianmu Li, Yanfang (Fanny) Ye, Shouhuai Xu

Published: 2021

Read: 08/2022

Summary

- Performs a systematic/mathematical overview of Adversarial Malware Defense.

Thoughts

- The evasion attack is much more extensively studied than the poisoning attack.
- Vastly more complicated than it needs to be and completely bogged down with mathematics.
- Has useful suggestions for potential future work.

Cited

- “Prudent practices for designing malware experiments: Status quo and outlook.” (2012) advice on how to conduct malware research effectively and ethically
- “Deceiving end- to-end deep learning malware detectors using adversarial examples” (2018) evasion attacks against MalConv using gradient optimization
- Poisoning attacks:
 - “The security of machine learning” (2010)
 - “Poisoning attacks against support vector machines” (2012)
 - “Security evaluation of support vector machines in adversarial environments” (2014)
 - “Towards poisoning of deep learning algorithms with back-gradient optimization” (2017)
 - “Label sanitization against label flipping poisoning attacks” (2018)
 - “Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach” (2018)
 - “When does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks” (2018)
 - “Why do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks” (2019)

Cited By

- “Adversarial Attacks against Windows PE Malware Detection: A Survey of the State-of-the-Art” (2021)

Notes

Introduction

- Surveys adversarial malware detection through the lens of a unified conceptual framework of assumptions based upon assumptions, attacks, defenses, and security properties
- Evasion vs poisoning adversarial machine learning (AML) attacks
- Adversarial Malware Detection (AMD) [16, 27, 36, 39, 40, 53, 114, 118, 127, 133, 134]
- Three differences to AMD vs AML: no standard feature definitions, feature definitions are often discrete, and malware functionality must be preserved

2. Survey and Systematization Methodology

1. Brief Review on ML-based Malware Detection

- (a) Features can be handcrafted or learned or both

- (b) Two kinds of features: static features (API calls, strings, etc.) and dynamic features (instructions, registry activities)
- (c) Case studies: The Drebin Malware Aetector, The MalConv Malware Detector

2. Framework

- (a) Systemizing Assumptions:
 - i. IID: assumes train/test files are Independently Identically Distributed (IID)
 - ii. Oracle Assumption: there is an oracle that can tell if two files are have the same functionality
 - iii. Measurability Assumption: there is a way to measure the manipulations to transform one file to another
 - iv. Smoothness Assumption:
 - v. Invertibility Assumption: feature extraction is invertible, or partially invertible with side-effects
- (b) Systemizing Attacks:
 - i. Attacker's Objective: indiscriminate (cause false negatives), targeted (cause specific false negatives), availability (cause unsustainable false positives)
 - ii. Attacker's Input (attacker may have some knowledge of A1-A5 and total knowledge of A6-A10):
 - A. A_1 : classifier's training set
 - B. A_2 : classifier's techniques, e.g., ensemble learning, weight regularization (regularization/dropout), adversarial training, verifiable learning (overestimate error of perturbations, then minimize them), robust features (selection), input transformation (non learning methods such as de-obfuscation), classifier randomization (a random subset of an ensemble to prevent attacker from learning about the classifier), sanitizing examples (detect adversarial examples)
 - C. A_3 : knowledge of feature set
 - D. A_4 : knowledge of training algorithm
 - E. A_5 : knowledge of classifier's responses/outputs to samples
 - F. A_6 : the perturbations the attacker can make
 - G. A_7 : attack tactics, e.g., basic evasion, optimal evasion 1 (attempts to minimize the perturbations in evasion), and optimal evasion 2 (attempts to maximize the classifier's loss and create a high confidence attack), basic poisoning, optimal poisoning (maximizes loss of surrogate classifier for high-confidence attack)
 - H. A_8 : attack techniques, e.g., Gradient-based Optimization (), Sensitive Features (inject/remove features to decrease classification error), Mimicry (mimics benign sample), Transferability (train surrogate model), Heuristic Search (does not need invertibility assumption), Generative Model (create adversarial malware vectors with generative model then invert into malware), and Mixture Strategy (combination of techniques)
 - I. A_9 : corresponds to the attacker's available possible adversarial files
- (c) Systemizing Defenses:
 - i. Defender's Objectives: goal is to detect malicious files (adversarial/normal) with minimal FPs
 - ii. Defender's Inputs: the defenders knowledge of A_1, \dots, A_9

3. Systemizing Security Properties

- (a) Representation Robustness: similar files have similar reps
- (b) Classification Robustness: similar reps have same classifications
- (c) Detection Robustness: feature extraction returns similar reps for files with the same functionality
- (d) Training Robustness: small changes in training set do not change classifier much

Systemizing AMD Arms Race

1. Systematizing Attack Literature

- (a) Most studies focus on indiscriminate attacks, evasion attacks
- (b) Oracle assumption widely made
- (c) Knowing defender's feature set is critical for transfer attacks
- (d) Evasion attacks are most effective when given freedom to perform large variety of manipulations

2. Systematizing Defense Literature

- (a) Most studies focus on black-box defenses, evasion attacks
- (b) There is no perfect defense against evasion/poisoning attacks
- (c) Sanitizing adversarial files is effective against black/grey box attacks, but not white box
- (d) Effective defenses often require the defender know the attacker's manipulation set, which is unrealistic in real world scenarios
- (e) The effectiveness of adversarial training depends on the defenders ability to identify the most powerful attack

3. Systematizing AMD Arms Race

- (a) Arms race in PDF malware detection, PDFrate
- (b) Arms race in android malware detection, Drebin
- (c) Arms race in DNN-based malware detection, MalConv

Future Research Directions

- Pinning down the causes of adversarial malware examples
- Characterizing the relationship between transferability and vulnerability, ie, the relationship between how knowledge gained from a surrogate model influences a systems vulnerability
- Investigating adversarial malware examples in the wild
- Quantifying the robustness and resilience of malware detectors
- Designing malware detectors with provable robustness and resilience guarantees (RF robust features [141], AL (Adversarial Training [53] or Ensemble Learning [142]), VL verifiable learning ie intentionally overestimate the error from the attacker [130])

Adversarial Deep Learning for Robust Detection of Binary Encoded Malware

Metadata

Authors: Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, Una-May O'Reilly

Published: 2018

Read: 08/22

Summary

- Modifies gradient-based adversarial approach to modify malware binary files.
- Uses binary features of API calls present in malware.
- Develops SLEIPNIR, a robust training approach to adversarial malware.
- Uses an open source malware dataset of 35k malware and 20k benign.

Thoughts

- Most adversarial malware work seems to assume that the input domain (features) are discrete rather than continuous.

Cited

- “Deep neural network based malware detection using two dimensional binary program features” (2015) DNN malware detection with two dimensional binary PE program features
- “Explaining and harnessing adversarial examples” (2014) FGSM adversarial generation white box attack, as well as general background about adversarial ML
- “Towards deep learning models resistant to adversarial attacks” (2017) original authors of saddle-based AE defense training

Cited By

- “Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection” (2020)
- “Adversarial Examples for CNN-Based Malware Detectors” (2019)

Notes

Introduction

- Presents four methods to create adversarial examples (AEs)
- Presents the SLEIPNIR framework for training adversarial malware detectors
- Source code and dataset available here <https://github.com/ALFA-group/robust-adv-malware-detection>

Related Work

- Adversarial approach Fast Gradient Sign Method FGSM finds the direction that moves the outputs of NN the most and most the inputs along this direction

Methods

-

Results and Conclusions

-

Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features

Metadata

Authors: J Saxe, K Berlin

Published: 2015

Read: 08/22

Summary

- One of the earlier works on DNN malware detection.
- Uses a slightly unconventional set of features.

Thoughts

-

Cited

-

Cited By

-

Notes

Introduction

-

Related Work

-

Methods

- Feature Engineering
 - 2D byte histogram features are essentially slices of the base-2 entropy every 1024 bits
 - Uses counts of PE imported DLLs but hashes them into 256 bins (does not avoid collisions)
 - PE metadata and string features as well

Results and Conclusions

-

Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection

Metadata

Authors: Deqiang Li and Qianmu Li

Published: 2020

Read: 08/22

Summary

- Uses an ensemble of DNNs in malware detection and an mixture of attack strategies in adversarial malware generation.
- The ensemble proves more effective at malware identification and the mixed attack strategies downgrade the performance of VirusTotal.
- Uses discrete input domain features.

Thoughts

-

Cited

- Cited in reference to perturbing the element in the feature space rather than the file space:
 - “Practical evasion of a learning-based classifier: A case study” (2014)
 - “Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks” (2019)
 - “Yes, machine learning can be more secure! A case study on Android malware detection” (2019)
 - “Intriguing properties of adversarial ML attacks in the problem space” (2019)
- “Adversarial training and robustness for multiple perturbations” (2019) the so-called max attack
- “Towards the first adversarially robust neural network model on MNIST” (2018) salt and peppering, pointwise attack

Cited By

-

Notes

Introduction

- Ensembles have been used in adversarial attacks and adversarial defense
- Proposes *mixture of attacks*, which lets attackers use multiple generative methods to make adversarial malware

Related Work

- Ensemble Attacks for improving effectiveness of AEs
 - Attacking multiple classifiers can improve the transferability of AEs
 - Deploying multiple attacks can entirely circumvent a DNNs defenses
- Ensemble Defenses
 -

Preliminaries

- Two types of evasion attacks: perturbations in the file space and perturbations in the feature space

- Perturbations in the feature space must be invertible, that is, one must be able to figure out how to modify the file in order to achieve the change in the feature space
- This file modification could perturb the feature rep in extra, unintentional ways, but experimentally, this does not appear to be a big issue
- Attack strategies in literature include gradient-based attacks (white box), gradient free (grey box) eg salt/pepper [18] and pointwise [18]

Methods

-

Results and Conclusions

-

A Framework for Enhancing Deep Neural Networks Against Adversarial Malware

Metadata

Authors: Deqiang Li; Qianmu Li; Yanfang Ye; Shouhuai Xu

Published: 2021

Read: 08/22

Summary

- Proposes six guiding principles for malware detection systems, ultimately, an ensemble of classifiers, each hardened via an input transformation, adversarial training, and semantic-preserving techniques.
- Thoroughly tests the attack with twenty different attack types.

Thoughts

- Actually recommends binarizing any continuous features so they are more robust to perturbations.
- Future work: how should graph-based or sequential-like feature reps be accommodated?

Cited

- “Intriguing properties of adversarial ml attacks in the problem space” (2020) the problem of adversarial malware examples are much less investigated
- “When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors” (2016) RF confidence scores to detect adversarial malware
- “HashTran-DNN: A. framework for enhancing robustness of deep neural networks against adversarial malware samples” (2018) detecting adversarial malware examples using DAE
- “Yes, machine learning can be more secure! A case study on android malware detection” (2019)
- “The random subspace method for constructing decision forests” (1998) ensembling method for sparse features
- “Gotcha - sly malware!: Scorpion a metagraph2vec based malware detection system” (2018) malware detection with graph-based feature representation

Cited By

- “Deep Learning for Android Malware Defenses: a Systematic Literature Review” (2022)
- “Malware Analysis by Combining Multiple Detectors and Observation Windows” (2021)
- “Ensemble dynamic behavior detection method for adversarial malware” (2022)

Notes

Introduction

- Proposes six guiding principles to enhance robustness of DNNs
- Validates the effectiveness of the framework against grey and whitebox attacks
- Achieves highest score in MIT Lincoln Lab challenge
- Code publicly available at https://github.com/deqangss/aics2019_challenge_adv_mal_defense

Related Work

- Ensemble Learning can improve a system’s resilience to adversarial examples
- Input Preprocessing transforms input to difference representation, which can reduce the impact of perturbations applied to the original input, e.g., feature squeezing [25]

- Adversarial Training add adversarial examples to the training data, but may only be effective for the specific kind of adversarial evasion attack methods the training data is augmented with
- Denoising Auto-Encoder (DAE) Representation Learning learns robust representations that may be less sensitive to perturbations

Adversarial Malware Attacks

- Random Attack (baseline): attacker randomly modifies feature at each iteration
- Mimicry Attack (AMD): perturb malware to mimic a benign examples
- FGSM Attack (CV): perturb feature vector in the direction of the norm of the gradients of the loss function
- Grosse Attack (AMD): manipulate absence of sensitive feature into presence of the feature
- BGA/BCA Attacks (AMD): increases binary features values from 0 to 1 based upon behavior of gradients
- PGD Attack (CV): supports feature addition and feature removal via gradient methods

Adversarial Malware Defense

- Guiding Principles:
 1. Knowing the Enemy: attempt to know as much about incoming attacks as possible
 2. Bridging Grey-Box vs White Box: in grey box scenario, attacker can train a surrogate classifier f' to mimic the classifier f , so defense in grey box scenario is essentially the same as defense in white box
 3. Ensembling
 4. Transformations Against Perturbations: binarize discrete features in binary ones to reduce impact of perturbations
 5. Using Vaccine: use minimax adversarial training to harden models
 6. Preserving Semantics: malware must remain functional after perturbations, so focussing on the import features of the malware should preserve classifier functionality. Using an encoder-decoder, such as DAE, can help with this.
- Create an ensemble of classifiers, each hardened via binarization, adversarial training, and de-noising auto encoder.

Gotcha - Sly Malware!: Scorpion A Metagraph2vec Based Malware Detection System

Metadata

Authors: Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, Melih Abdulhayoglu

Published: 2018

Read: 08/22

Summary

- Models malware content-based and relationship-based features in network structures, generates embeddings for them, then learns an SVM classifier

Thoughts

- Work could be improved upon by using higher dimensional embeddings and a more potent classifier

Cited

- - “Combining file content and file relations for cloud based malware detection” (2011) authors previous work, which uses HIN, but not the meta-graph
- Content Based Feature Approaches
 - “Adversarial Machine Learning in Malware Detection: Arms Race between Evasion Attack and Defense” (2017)
 - “Malicious sequential pattern mining for automatic malware detection” (2016)
 - “Malware detection with quantitative data flow graphs” (2014)
 - “DeepAM: a heterogeneous deep learning framework for intelligent malware detection” (2018)
- Relationship Based Feature Approaches
 - “Polonium: Tera-scale graph mining for malware detection” (2010)
 - “Analyzing file-to-file relation network in malware detection” (2015)
 - “Intelligent malware detection based on file relation graphs” (2015)
 - “FindMal: A file-to-file social network based malware detection framework” (2016)
 - “Guilt by association: large scale malware detection by mining file-relation graphs” (2014)

Cited By

- “alpha-Cyber: Enhancing Robustness of Android Malware Detection System against Adversarial Attacks on Heterogeneous Graph based Model”
- “Sok: Arms race in adversarial malware detection”

Notes

1. Introduction

- (a) Uses a heterogeneous information network (HIN) and a meta-graph to represent content and relation-based features in malware
- (b) Learns low dimensional representations for nodes in HIN

System Architecture

- (a) Data Collector
- (b) Feature Extractor: extracts content-based and relation-based features
- (c) Meta Graph Builder HIN: structural HIN models relationships between entities and meta graphs are built from the HIN

- (d) Metagraph2vec: learns low dimensional feature representations from metagraph
- (e) Malware Detector: uses SVM to classify from latent features

2. Proposed Method

(a) Feature Extraction

- Content-based features: extract windows API calls from Import Tables
- Relation-based features: binary matrices where 1 represents the relationship is present between the i th and j th element
 - R1: file-replace-archive matrix R
 - R2: file-exist-machine matrix E
 - R3: file-create-file matrix C
 - R4: file-include-API matrix I
 - R5: API-belongto-DLL matrix B

(b) Meta-graph Based Relatedness

- A HIN is a graph with an entity type mapping $V \rightarrow A$ and a relationship type mapping $E \rightarrow R$, where A are the valid entities, R are the valid relationships, and the network schema is a graph $T_G = (A, R)$
- $A = \{ \text{PE file, archive, machine, API, DLL} \}$
- $R = \{ R_1, R_2, R_3, R_4, R_5 \}$
- A meta-graph is a DAG defined on a HIN
- Traditional relationship learning typically factors the adjacency matrix of the meta-graph, which is computationally intensive

(c) Metagraph2vec

- HIN Rep Learning attempts to map each vertex in a HIN to a vector
- first, meta-graph guided random walk is proposed to map the word-context concept in a text corpus into a HIN
- then skip-gram is utilized to learn effective node representation for a HIN
- finally, a multi-view fusion algorithm is proposed to incorporate different node representations learned based on different meta-graph schemes.

3. Results and Conclusions

- (a) Uses data from Comodo Cloud Security Center

4. Related Work

- (a) Most malware detection uses either content based features or relation based features

A survey on practical adversarial examples for malware classifiers

Metadata

Authors: Daniel Park, Bülent Yener

Published: 2020

Read: 09/22

Summary

- Reviews literature on adversarial malware techniques that preserve the functionality of the malware.

Thoughts

- A big problem with adversarially modifying malware is maintaining functionality. Using code-understanding tools could allow an attacker to add code blocks that don't actually execute.

Cited

- "Intriguing Properties of Adversarial ML Attacks in the Problem Space" (2020) discusses the difficulty of the inverse transform from feature rep to malware
- "Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers" (2018) GADGET adds API calls to malware
- "Deceiving Portable Executable Malware Classifiers into Targeted Misclassification with Practical Adversarial Examples" (2020) add bogus code blocks and other interesting features
- "HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs" (2019) uses ASTs in AM generation
- "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning" (2018)
- Surveys:
 - "Adversarial Examples: Attacks and Defenses for Deep Learning"
 - "Towards Adversarial Malware Detection: Lessons Learned from PDF-Based Attacks" (2019)
 - "A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web" (2017)
 - "A Survey on Malware Detection Using Data Mining Techniques" (2017)
 - "Survey of machine learning techniques for malware analysis" (2019)

Cited By

-

Notes

1. Introduction

- Focuses on literature that generates functional malware executables, rather than those that just generate challenging features

2. Background

(a) ML for Malware

- Static features include: bytes, opcodes, API calls, system calls, and environment info
- Dynamic features include: opcodes, API calls, systems calls, and environment info extracted during execution

(b) Adversarial examples

- Threat Model: threat vector and surface (means the adversary interacts with the model), knowledge, and capabilities

3. Practical Attacks

(a) Gradient-based Approaches

- Editing Bytes and metadata (all require white box except GADGET)
- Code transformations

(b) Problem-driven Approaches

- Editing Bytes and metadata
- Code transformations

(c) Problem-driven Approaches

-

4. Research Directions

- Defending against practical malware examples: smoothing [2] and randomization [60] for AMD
- Relationships between obfuscated and adversarial examples
- Integration of static and dynamic analysis techniques

Deceiving Portable Executable Malware Classifiers into Targeted Misclassification with Practical Adversarial Samples

Metadata

Authors: Yunus Kucuk, Guanhua Yan

Published: 2020

Read: 09/22

Summary

- Trains three RF classifiers using opcode frequencies, API function imports, and system calls extracted from dynamic analysis
- Uses genetic algorithms to create adversarial examples for each classifier: adding bogus code block, obfuscating API calls, and adding API calls to modify system call usage

Thoughts

- RF would probably be inherently more robust to adversarial attacks than DNNs.

Cited

- “Learning to evade static PE machine learning malware models via reinforcement learning” (2018)
- “Obfuscator-LLVM - software protection for the masses” (2015)

Cited By

- “A Unified Framework for Adversarial Attack and Defense in Constrained Feature Space” (2022)

Notes

1. Introduction

- Trains three RF classifiers using opcode frequencies, API function imports, and system calls extracted from dynamic analysis
- Develops techniques that fool each type of classifier

2. Related Work

-

3. Malware Dataset and Classifiers

- Opcode n-gram features are attained by disassembling binaries (IDA Pro)
- API function features are binary vectors extracted from the .idata section of the PE
- System call feature vectors counts of how often each call is made during dynamic execution

4. Deceiving the Opcode Classifier

- Essentially add bogus code hidden behind if statements that never execute
- Carefully monitors the changes in the feature vector

5. Deceiving the API Classifier

- Performs API obfuscation

6. Deceiving the System Call Classifier

- Adds API calls to increase the counts of specific system calls

7. Conclusions

- Only uses three RF classifiers, when in reality, a diverse ensemble would be most practical
- Only uses genetic algorithms to create adversarial examples

Adversarial Attacks against Windows PE Malware Detection: A Survey of the State-of-the-Art

Metadata

Authors: Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Chunming Wu, Shouling Ji, Tianyue Luo, Jingzheng Wu, Yanjun Wu
Published: 2022
Read: 09/22

Summary

- Three main challenges: format-preserving, executability-preserving and maliciousness-preserving
- Black box attacks in the problem space are theoretically the most model agnostic
- These attacks use creative methods to generate the examples, such as genetic algorithms, RL, GANs, etc.

Thoughts

- 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

Cited

- Surveys for NLP, Graphs, and CV:
 - “Improving the reliability of deep neural networks in NLP: A review” (2020)
 - “Adversarial attack and defense on graph data: A survey” (2018)
 - “Threat of adversarial attacks on deep learning in computer vision: A survey” (2018)
- “Misleading authorship attribution of source code using adversarial learning” (2019) discusses feature-space to problem-space dilemma
- Then, the authors employ an interpretation model of SHAP [85] to assign each n-gram feature with an importance value and observe that the 4-gram “move + and + or + move” feature is a typical malicious feature as it almost does not appear in the benign PE samples. COULD this be used for poisoning attacks????? Also deploys a substitution method
- “An Adversarial Machine Learning Method Based on OpCode N-grams Feature in Malware Detection” (2020) authors substitute blocks of opcode for computationally equivalent blocks of opcode
- Changes API call sequence:
 - “Black-box attacks against RNN based malware detection algorithms” (2017) generative RNNs
 - “Generic black-box end-to-end attack against state of the art API call based malware classifiers” (2018) GADGET
 - “Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers” (2020) BADGER builds upon GADGET
- Black Box - Problem Space Attacks:
 - Reinforcement Learning
 - * ~~“Extending anti-malware engines with deep reinforcement learning”~~ (2019)
 - * “DeepDetectNet vs RLAttackNet: An adversarial method to improve deep learning-based static malware detection model” (2020)
 - * “An IRL-based malware adversarial generation method to evade anti-malware engines” (2021)
 - * “Generating adversarial examples for static PE malware detector based on deep reinforcement learning” (2020)

- * “Binary Black-Box Attacks Against Static Malware Detectors with Reinforcement Learning in Discrete Action Spaces.” (2021)
- * “AIMED-RL: Exploring Adversarial Malware Examples with Reinforcement Learning”
- * “Enhancing machine learning based malware detection model by reinforcement learning” (2018)
- Evolutionary Algorithms
 - * “AIMED: Evolving Malware with Genetic Programming to Evade Detection” (2019)
 - * “ARMED: How Automatic Malware Modifications Can Evade Static Detection” (2020)
 - * “Functionality-preserving Black-box Optimization of Adversarial Windows Malware” (2020) GAMMA
 - * “MDEA: Malware Detection with Evolutionary Adversarial Learning” (2020)
- GANs
 - * “Black-Box Adversarial Attacks Against Deep Learning Based Malware Binaries Detection with GAN” (2020) GAPGAN
 - * “MalFox: Camouflaged Adversarial Malware Example Generation Based on C-GANs Against Black-Box Detectors” (2020)

Cited By

•

Notes

1. Introduction

- Malware is highly structured, unlike images, which makes it difficult to make adversarial approaches preserve the functionality of malware
- Three big challenges: format-preserving, executability-preserving and maliciousness-preserving

2. Related Work

•

3. ML and DL for PE Malware Detection

(a) PE file Layout and Malware

- Contains header information (tells OS how to map the PE file into memory), section information (executable code and associated data), and un-mapped data (chunks of un-used bytes)

(b) Learning Framework for PE Malware Detection

- PE malware usually not publicly released
- Static features include: byte sequences (raw bytes, n-grams of bytes, etc.), readable strings (file names, IP-addresses, etc.), header information (file statistics, imported/exported functions, etc.), greyscale image (bytes to image)
- Dynamic features include: system resource status, file status, registry status, network status
- Hybrid features include (can be extracted statically or dynamically): opcode, system calls, API calls, control flow graph (CFG), function call graph

4. Challenges of Adversarial Attacks for PE Malware

(a) General concept and Taxonomy

- Feature space attack: minimize the difference between an adversarial example and its original counterpart *in the feature space*, \mathbb{X} such that it is misclassified: $\min_{x'} \text{distance}(x', x) \text{ s.t. } x' \in \mathbb{X}, f(x') = y' \neq y$
- Problem space attack: how to modify the real-world examples with minimal cost *in the problem space*, \mathbb{Z} such that the it is misclassified: $\min_{z'} \text{cost}(z', z) \text{ s.t. } z' \in \mathbb{Z}, f(\phi(z')) = y' \neq y$. Does not consider the feature representation of the input samples.
- Because of ϕ , the problem space attacks cannot use gradient-based methods

(b) Three Unique Challenges of Adversarial Attacks for PE Malware

- Problem-feature space dilemma: ϕ can vary and is generally unknown by the attacker, ϕ is unlikely to be invertible
- Challenges of maintaining semantics of adversarial PE malware in problem space: ,

- i. format preserving: follow format of PE files
 - ii. executability preserving: keep the file executable
 - iii. maliciousness preserving: keep the maliciousness intact
 - To ensure these properties are maintained, the original and modified malware should be run in a sandbox and their runtime status should be the same
- 5. Adversarial Attacks Against PE Malware Detection: The State of the Art
 - (a) White Box Attacks
 - i. Feature Space Attacks
 - Raw Bytes Detectors: adding/appending bytes to unused section of the PE file, including carefully selected blocks from benign files
 - API Call List Detectors: various methods to search binary API call space and determine which bits to flip (API calls to add) to create the example
 - Visualization Detectors:
 - Other Detectors: include substituting malicious-looking opcode sequences with equivalent benign-looking ones
 - ii. Problem Space Attacks
 - Almost all are targeted at raw byte based detectors, eg MalConv
 - Found that MalConv gets most of its information from the PE header file, which leads to injecting raw bytes at the end of the PE header
 - (b) Black Box Attacks
 - i. Feature Space Attacks
 - API Call List Detectors: MalGAN, EvnAttack manipulates API imports to mimic goodwill
 - API Call Sequence Detectors: generative RNN, GADGET [110] & BADGER [109] insert API calls into the code
 - Other: RL agent for graph-based detectors [148]
 - ii. Problem Space Attacks
 - Reinforcement Learning Attacks: [8,9] completely black box adversarial malware approach and follow up works: [20, 39, 41, 42, 72, 76, 139], [41, 42, 76] ensure functionality
 - Randomization Attacks: randomly change malware in format-preserving way, then ensure functionality in Cuckoo Sandbox
 - Evolutionary Algorithm Attacks: AIMED [15], GAMMA [36]
 - GAN Attacks: GAPGAN [145] dominates MalConv
 - (c) Summary of Attacks
 - Most white-box attacks use gradient based methods, but to preserve the malware functionality are malware-detector specific
 - Black box attacks in the feature space develop corresponding manipulations in the problem space, so are malware-detector specific
 - Black box attacks in the problem space are theoretically the most model agnostic
- 6. Adversarial Defense for PE Malware Detection
 - (a) Adversarial Training: mitigates effect of adversarial attacks, but introduces significant training costs
 - (b) Other Defense Methods:
- 7. Discussions
 - (a) Beyond Adversarial Attacks
 - Universal Adversarial Perturbation: finding a single perturbation (series of transformations to any arbitrary input object) can be applied over large set of inputs for misclassification
 - Poisoning Attacks: goal is to make classifier misclassify malware with a specific kind of trigger [113]
 - Usually the label of the poisoned malware cannot be directly manipulated, only the malware itself
 - (b) Future Work
 - More research on attack methods than attack defense
 - Attack defenses are usually attack-specific
- 8. Conclusions
 -

Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning

Metadata

Authors: Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, Phil Roth
Published: 2018
Read: 08/22

Summary

- Uses RL to create adversarial malware
- RL agent randomly modifies malware while preserving its validity
- Attacks are black box and modifications are made directly to the malware itself
- Source code: <https://github.com/endgameinc/gym-malware>
- Allows the RL agent to know/use the feature rep of the classifier, but provides arguments as to why it is not necessary

Thoughts

- Could be interesting to train an ensemble using dramatically different feature reps, then test an RL, GAN, Graph, and traditional Adversarial malware attacks

Cited

-

Cited By

-

Notes

1. Introduction
 - DNNs are susceptible to gradient-based attacks; non-differentiable algorithms susceptible to genetic attacks
 - Proposes a RL based strategy that does not even require a score (black box)
2. Related Work
 - Evading ML Models
 - (a) Direct gradient-based attacks: model must be differentiable and the weights must be accessible by attacker
 - (b) White box against models that report a score (grey box)
 - (c) Binary black box
 - How this work differs:
 - Output from classifier is boolean
 - Feature space and classifier details are unknown
 - No oracle to guarantee the sample is valid
3. Methods
 - (a) Reinforcement Learning
 - Goal is to derive a policy given a function that estimates the expected utility of a state and an action, Q
 - Deep Q-learning uses a DNN to determine the Q function
 - Actor Critic model with Experience Replay (ACER) uses DNNs to learn both the policy model π and the Q -function

- Agent gets an estimate of the environment state $s \in S$, represented by a feature vector of the malware (does not need to correspond to the internal rep of malware used by anti-malware classifier)
- Q-function select an action to take from A , where allowable actions are any modifications to the PE file that: a) do not break the PE format and b) do not alter the functionality of the malware
- Reward is 0 if the adversarial malware is detected, else R

(b) Environment

- Malware exists as raw bytes in game environment, but is represented as a state with a \mathbb{R}^{2350} vector (PE header metadata, etc., etc.)
-

(c) Action Space

- A variety of modifications are allowed, such as: manipulating section names, creating new sections, appending bytes, packing/unpacking file etc.

4. Experimental Setup

- Uses gradient boosted decision trees
- Postulate that agents learning is simpler when 1) feature rep by model under attack has similar learning rep as agent, 2) agent actions are observable by the state representation, 3) agent's actions can impact any part of the feature vector

5. Conclusions

-

A Unified Framework for Adversarial Attack and Defense in Constrained Feature Space

Metadata

Authors: Thibault Simonetto, Salijona Dymishi, Salah Ghamizi, Maxime Cordy, Yves Le Traon

Published: 2021

Read: 09/22

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

- Proposes a domain-agnostic unified framework for constrained feature-space attacks
- Uses the framework to develop a generic constraint language and two adversarial attack strategies

2. Related Work

-

3. Methods

- (a) Develops a mathematical model for describing the constraints of adversarial modifications
- (b) Uses said model to develop Constrained Projected Gradient Descent (CPGD) and Multi-Objective Generation of Constrained Adversarial Examples

4. Conclusions

-

Adversarial Attacks on Deep-learning Models in Natural Language Processing: A Survey

Metadata

Authors: WEI EMMA ZHANG, QUAN Z. SHENG, AHOUD ALHAZMI, CHENLIANG LI
Published: 2020
Read: 09/22

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

-

2. Overview

- (a) Deep Learning in NLP involve: FFNNs, CNNs, RNNs/LSTMs/GRUs, Seq2Seqs, Transformers, RL models, GANs/VAEs

3. From Image to Text

- (a) CV methods include: Limited memory Broyden-Fletcher Goldfarb-Shanno algorithm (L-BFGS), Fast Gradient Sign Method (FGSM), JAcobian Saliency Map Adversary (JSMA), C&W Attack, DeepFool, Substitute Attack, and GAN attacks

- (b) Differences between images and text include:

- Discrete vs continuous inputs
- Perceivable vs unperceivable perturbations
- Semantic vs semanticsless

- (c) Vectorizing Textual Inputs

- Word-count encoding: use bag of words to acquire vector rep of text based on word counts
- One-hot encoding: each token is assigned a boolean vector where 1 indicates its position in the text
- Dense encoding: eg word2vec

- (d) Perturbation Measurement

- Norm-based: requires continuous (embedding) representation, but usually results in incomprehensible adversarial texts
- Grammar/Syntax Measurements: use grammar or perplexity to ensure adversarial examples are valid
- Semantic preserving measurement: use distance functions to ensure semantic meaning of adversarial text is similar
- Edit distance measurements:
- Jaccard similarity coefficient:

4. Attack

(a)

5. Defense

(a)

6. Future Work

(a)

Evading Anti-Malware Engines With Deep Reinforcement Learning

Metadata

Authors: ZHIYANG FANG, JUNFENG WANG , BOYA LI, SIQI WU, YINGJIE ZHOU , AND HAIYING HUANG

Published: 2019

Read: 09/22

Summary

- Proposes DQEAF, a deep-Q network to evade anti malware engines
- Shown to be more effective than the original RL adversarial approaches

Thoughts

- How do these RL attacks fare against an ensemble of classifiers?
-

Cited

- “Automatically evading classifiers” (2016) genetic programming black box adversarial malware
- “Evading machine learning malware detection” (2017) the other OG RL adversarial malware attack

Cited By

- “Feature Selection for Malware Detection Based on Reinforcement Learning” (2019) uses RL to select the optimal features for malware detection
- “Binary Black-box Evasion Attacks Against Deep Learning-based Static Malware Detectors with Adversarial Byte-Level Language Model” (2020) MalRNN black box attack

Notes

1. Introduction

- Agent inspects malware then selects a sequence of functionality-preserving actions to modify the samples
- Extracts features with a raw binary byte stream
- DQEAF has the following:
 - only four actions taken (small state space)
 - low dimensional raw binary stream
 - high evasion success rate

2. Related Work

- Improvements of this work over other RL approaches are:
 - (a) every action is guaranteed to be effective on all samples
 - (b) low dimensions of observations reduce instability
 - (c) priority is taken into consideration during the replay of past transitions

3. Methods

(a) Primary Concept

- In RL, an agent seeks to learn optimal actions that maximize rewards in an environment

(b) Model Structure

- Uses a Markov Decision Process: $(S, A, \gamma, R(A, S))$, where S is a set of states, A is a set of actions, γ is a discount factor, $R(A, S)$ is a reward value after state transition from $s \in S$ to $s' \in S$

(c) Environment

- RL agent’s environment is the feature rep the agent observes

- Used byte histogram counts

(d) Action Space

- larger action spaces make it harder for the RL agent to learn effective policy
- ARBE: append random bytes to end of PE file
- ARI: append random function to import address table
- ARS: append randomly named section to section table of PE file
- RS: remove signature from certificate table of DataDiscovery

(e) Reward

- 0 if labeled as malicious
- calculated from formula if labeled benign (goal), but reward decreases with respect to the number of turns taken

(f) Agent

- Deep convolutional Q-network
- Two CCN networks: Q-value and Q-target

(g) Training

-

4. Performance and Evaluation

- Runs malware in Cuckoo sandbox to ensure the file has not been corrupted
- Uses IDA Pro to generate function call graphs, flow charts, and binary files that ensure that the structure and function of the malware is the same
- Uses gradient boosted classifier trained on 100k samples using several different kinds of features (different from RL environment)
- Supposedly outperforms the original RL-based adversarial attack

5. Conclusions

-

Evading Machine Learning Malware Detection

Metadata

Authors: HS Anderson, A Kharkar, B Filar, P Roth

Published:

Read:

Summary

- General black box RL adversarial approach with source code: <https://github.com/EndgameInc/gym-malware>
- Essentially identical to “Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning”

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

-
-

2. Related Work

- Malware attacks largely fall into one of the following categories: direct gradient-based attacks, attacks against models that report a score, binary black box attacks
- MalGAN trained surrogate model from binary outputs then used gradients to perturb the malware samples
- However, MalGAN requires knowledge of the feature space used to train the surrogate model and the GAN
- Furthermore, MalGAN performs perturbations in the feature space and does not create malicious malware

3. Methods

- (a) Methodology is essentially identical to “Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning”

4. Conclusions

-

MERLIN – Malware Evasion with Reinforcement Learning

Metadata

Authors: Tony Quertier, Benjamin Marais, Stephane Morucci, and Bertrand Fournel

Published: 2022

Read: 09/22

Summary

- Uses RL with DQN and REINFORCE algorithms to create adversarial examples
- Performs grey-white-box attacks against MalConv, LGMBM EMBER, and a Grayscale classifier
- Successfully performs pure black box attacks against commercial antivirus software, AV

Thoughts

- First paper I've seen that tests against a commercial AV software

Cited

- “Bodmas: An open dataset for learning based temporal analysis of PE malware” (2021)
- “Mal- Fox: Camouflaged Adversarial Malware Example Generation Based on C-GANs Against Black-Box Detectors” (2020)
- “Functionality-preserving black-box optimization of adversarial windows malware” (2021) - particularly interesting method based on a constrained minimization problem while preserving executable files functionalities
- “MAB-malware: A reinforcement learning framework for attacking static malware classifiers” (2020)

Cited By

-

Notes

1. Introduction

-

2. Related Work

-

3. Algorithms and RL framework

(a) RL Agents

- Deep Q-Network (DQN): uses a DNN to predict the optimal action-value function, Q
- Monte Carlo Policy Gradient (REINFORCE): maximizes the expected return of the policy with respect to the parameters

(b) Environments

- Lets the RL agent know the feature vectors used by the classifiers and the classifier's prediction score
- The AV classifier is black box

(c) Actions

- Uses the actions of Anderson et al.
- LIEF can make modifications to a PE file, but can sometime break the functionality of the file
- During RL, the malware is verified to be functional using Cuckoo sandbox
- After a successful evasion, the malware is verified to have the same behavior using other sandbox tools

4. Experiments

- BODMAS dataset was hardest to evade, so this dataset was used
- MalConv, Greyscale were easy to evade
- EMBER and AV were harder to evade, REINFORCE outperformed DQN
- In general, REINFORCE took more diverse actions than DQN, which may be responsible for its improved success

Examining Zero-Shot Vulnerability Repair with Large Language Models

Metadata

Authors: Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, Brendan Dolan-Gavitt

Published: 2023

Read: 09/22

Summary

- Investigates the use of LLMs to fix cyber security issues in code, such as over subscription of arrays in C and improperly sanitized SQL statements in Python

Thoughts

- Could this strategy be used to “fix” portions of suspicious assembly? IE if one particular block of assembly is contributing greatly towards a models assessment that the executable is malware, could that block be “fixed”

Cited

-

Cited By

-

Notes

1. Introduction

- Investigates the usage of LLM for code to repair identified cyber security bugs in code
- Other works attempt to predict the exact tokens that a developer would use to fix the issue
- This work attempts to leverage the inherent understanding of LLMs
- Asks the following research questions:
 - (a) Can LLMs generate code to fix security vulnerabilities
 - (b) Does varying the mount of context in comments affect LLMs abilities
 - (c) What are the challenges in the real world
 - (d) How reliable are LLMs at generating repairs

2. Related Work

-

3. RQ1/RQ2: Synthetic Experimentation

- Tries to fix out-of-bounds errors for arrays and improper sanitation errors for SQL statements

4. Conclusions

-

Towards Automated Malware Creation: Code Generation and Code Integration

Metadata

Authors: Cani, Andrea and Gaudesi, Marco and Sanchez, Ernesto and Squillero, Giovanni and Tonda, Alberto

Published: 2014

Read: 09/22

Summary

- Very brief paper.
- Use EA algorithms to inject malware into benign program.

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

- Uses μ GP, an EA toolkit, to 1) make the malware undetectable and optimize the injection of the malware by creating a Trojan Horse
- Checks the generated malware for correct behavior
- Uses EA algorithm to find the optimal position to inject malicious code inside of a benign executable
- Uses a single simple malware program, Timid
- Uses an ensemble of freeware AV applications
-

2. Related Work

-

3. Methods

(a)

4. Conclusions

-

Auditing Anti-Malware Tools by Evolving Android Malware and Dynamic Loading Technique

Metadata

Authors: Yinxing Xue, Guozhu Meng, Yang Liu, Tian Huat Tan, Hongxu Chen, Jun Sun, and Jie Zhang

Published: 2017

Read: 09/22

Summary

-

Thoughts

-

Cited

- 6 - GENOME existing benchmarks for android
- 7 - DREBIN existing benchmarks for android
- 2,3 - DroidChamelion integrated three transformations for AEA android
- 11 - previous study focuses on the modelling and code generation for the attack of privacy leakage
-

Cited By

-

Notes

1. Introduction

-

2. Related Work

-

3. Methods

(a)

4. Conclusions

-

Evaluating Large Language Models Trained on Code

Metadata

Authors:

Published: 2021

Read: 09/22

Summary

- Briefly discusses using codex to create polymorphic malware

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

-

2. Related Work

-

3. Methods

(a)

4. Conclusions

-

The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement

Metadata

Authors:

Published: 2022

Read: 09/22

Summary

- Briefly discusses how the non determinism of codex could be an issue for malware defense

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

-

2. Related Work

-

3. Methods

(a)

4. Conclusions

-

Pop Quiz! Can a Large Language Model Help With Reverse Engineering?

Metadata

Authors:

Published: 2022

Read: 09/22

Summary

- Uses LLMs on malware source code, not disassembled binaries

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction

-

2. Related Work

-

3. Methods

(a)

4. Conclusions

-

Single-Shot Black-Box Adversarial Attacks Against Malware Detectors: A Causal Language Model Approach

Metadata

Authors: James Hu, Mohammadreza Ebrahimi, Hsinchun Chen
Published: 2021
Read: 09/22

Summary

- Trains a GPT CLM on malware bytecode to perform single-shot AMG method, MalGPT
- MalGPT appends file-specific bytes to an executable
- Outperforms other append-style attacks at fooling MalConv

Thoughts

- Seems to “test functionality” using VirusTotal...unsure what that means

Cited

- “Exploring adversarial examples in malware detection” (2019) single shot learning AMG
- “Binary black-box evasion attacks against deep learning-based static malware detectors with adversarial byte-level language model” (2020) DL LMs in malware analysis
- “Black-box attacks against rnn based malware detection algorithms” (2018) DL LMs in malware analysis

Cited By

-

Notes

1. Introduction

- Adversarial Malware Generation (AMG) is crucial to the success of ML systems
- Single shot evasion - one query to the defensive model
- Many append-based attacks require numerous queries to the detector before an example can be generated
- Detectors usually have query limits, so this is unrealistic
- Need for single shot AMG methods as these are the most realistic kind
- Casual Language Models (CLMs), eg GPT, are a solution for dealing with extremely long sequences, as is the case if malware is modeled as a byte sequence
- Attempts to evade a detector in a single query

2. Related Work

(a) AMGs

- Most studies use VirusTotal dataset
- Very few studies perform single shot AMG

(b) CLMs

- Some works have attempted to model malware as language
- CLMs incorporate temporal aspects into language modeling and are more well suited to long sequences than traditional LLMs

(c) GPT

- state of the art casual language model

3. Methods

(a) Threat Model

- Uses a single shot black box setting
- Uses append attacks ; 10 KB, ie, finds a good place to append certain bytes

(b) MalGPT Architecture

- malware is fed into trained GPT2 model
- model generates file-specific byte sequence
- sequence appended to original malware sample
- malware variant examined for functionality on VirusTotal

(c) GPT2 Model Training

- Trains the GPT model on a set of benign files

4. Conclusions

- MalGPT significantly outperformed other append-based attacks on fooling MalConv

Leveraging Automated Unit Tests For Unsupervised Code Translation

Metadata

Authors: Baptiste Roziere, Jie M. Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, Guillaume Lample

Published: 2022

Read: 09/22

Summary

- Introduced a novel method to grow a parallel corpus for automated code translation, from completely monolingual data

Thoughts

- Can use parallel functions extracted from GeeksForGeeks along with their units tests. Can compile these functions to obtain correct assembly for verification
- Generally speaking, it seems like most unsupervised training methods use DAE + backtranslation to perform tanscompilation
- Instead of using unit tests, could use malware detection as the correct/incorrect input

Cited

- Code Completion
 - “Code completion with neural attention and pointer networks” (2018)
 - “A self-attentional neural architecture for code completion with multi-task learning” (2020)
 - “Code prediction by feeding trees to transformers” (2021)
 - “Fast and memory-efficient neural code completion” (2021)
- “Measuring coding challenge competence with apps” (2021) evaluated the competence of several language models for solving coding challenges
- Roziere et al. (2020) proposed TransCoder
- Later, Roziere et al. (2021) (DOBF)
- **“Unsupervised translation of programming languages”** (2020) TransCoder
- **“DOBF: A deobfuscation pre-training objective for programming languages”** (2021) improved transcoder

Cited By

-

Notes

1. Introduction

-

2. Related Work

- Unit Test Generation
 - Unit test generation has been of interest for decades
 - Recently, NN have been used to successfully generate unit tests (Tufano et al., 2020)
 - This work uses EvoSuite because its open source, widely used, and currently being developed
 - Uses mutation testing to ensure the unit tests work (deliberately breaks the code to ensure test triggers)

Machine Learning For Programming Languages

- A variety of pretrained models have been used for code tooling
- Translation of Programming Languages

- TransCoder is pretrained with Masked Language Modeling and trained with Denoising Auto Encoder and Back Translation
- Using a deobfuscation objective (DOBF) with MLM improves the model

3. Method

(a) Parallel Data Creation

- EvoSuite uses evolutionary algorithms to create tests that maximize some score
- only keeps unit test suites with a mutation score larger than 90% for building parallel dataset
-

(b) Training Method

- Uses a pretrained translation model
- proposes method for online and offline training

(c) Evaluation

- Uses parallel functions extracted from GeeksForGeeks along with associated unit tests

4. Experiments

- uses encoder decoder with transformers (exact same as TransCoder)
- uses Google Big Query datasets

Metadata

Authors:
Published:
Read:

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction
 -
2. Related Work
 -
3. Methods
 - (a)
4. Conclusions
 -

Metadata

Authors:
Published:
Read:

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction
 -
2. Related Work
 -
3. Methods
 - (a)
4. Conclusions
 -

Metadata

Authors:
Published:
Read:

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

- 1. Introduction
 -
- 2. Related Work
 -
- 3. Methods
 - (a)
- 4. Conclusions
 -

Metadata

Authors:
Published:
Read:

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction
 -
2. Related Work
 -
3. Methods
 - (a)
4. Conclusions
 -

Metadata

Authors:
Published:
Read:

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction
 -
2. Related Work
 -
3. Methods
 - (a)
4. Conclusions
 -

TEMPLATE

Metadata

Authors:
Published:
Read:

Summary

-

Thoughts

-

Cited

-

Cited By

-

Notes

1. Introduction
 -
2. Related Work
 -
3. Methods
 - (a)
4. Conclusions
 -