University of Idaho

CS 404 / CS 504

Special Topics: Adversarial Machine Learning

Dr. Alex Vakanski

Lecture 10

AML in Cybersecurity – Part I: Malware Detection and Classification

Lecture Outline

- Machine Learning in cybersecurity
- Adversarial Machine Learning in cybersecurity
  - Taxonomy of AML attacks in cybersecurity
  - AML in cybersecurity versus computer vision
- Malware detection and classification
  - Malware analysis systems
    - Static malware analysis systems
    - Dynamic malware analysis systems
  - ML models for malware classification
    - Static and dynamic features
    - Deep Learning-based malware classification
  - Adversarial attacks on ML-based malware classifiers
    - Traditional ML models
    - Deep Learning approaches

ML in Cybersecurity

Machine Learning in Cybersecurity

- The cybersecurity domain is marked with a perpetual battle between security analysts and adversaries
  - Adversaries continually innovate and adapt their attack approaches, resulting in ever-increasing complexity of cyber attacks
  - Security analysts attempt to quickly respond to new attacks, and they try to be one step ahead of cyber adversaries
- Machine Learning (ML) models have a potential for addressing the complexity of recent attacks, and are increasingly used in cybersecurity
  - Yet, all ML models are vulnerable to adversarial attacks
  - Investigating adversarial attacks and defenses against ML models in cybersecurity applications is crucial for this domain
- Examples of adversarial ML attacks in cybersecurity:
  - Spam messages designed to avoid ML-based spam filters
  - Ransomware developers evading anti-malware ML-based systems
  - Malware worms evading ML classifiers, and spreading across the network
  - Crypto software evading ML systems, and using resources for mining crypto-currency

Cybersecurity Challenges

Machine Learning in Cybersecurity

- Traditional cyber defense relied predominantly on signature-based and heuristic-based methods
  - ***Signature*** is a unique set of features that identifies a specific file (e.g., malware)
  - ***Heuristic*** is a set of rules developed by security analysis for protection against specific attacks
- Challenges: both signature- and heuristic-based methods require knowledge about the malicious files, in order to determine the signature or heuristic rules
  - E.g., these approaches have difficulties detecting unknown variants of malware
- Other challenges in cybersecurity:
  - Traditional defense methods based on manually crafted signatures or heuristic rules are unable to keep pace with recent attacks, which are becoming more complex and sophisticated
  - Organizations are also experiencing a shortage of cybersecurity skills and talent
- These cybersecurity challenges can be addressed by ML solutions, due to the ability to automatically identify signature features or rules for attack identification, and have capacity to handle large volumes of data

ML Specifics in Cybersecurity

Machine Learning in Cybersecurity

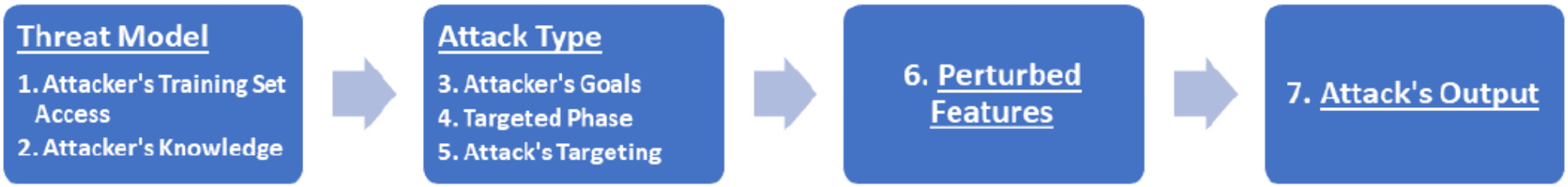
- Application of ML in cybersecurity also introduces unique challenges, including:
  - Requirement for large representative datasets for model training
    - Acquisition of cybersecurity datasets and sample labeling is expensive and time-consuming
    - Small or imbalanced datasets can lead to poor performance (e.g., missing harmful files, or high false alarms rate)
  - Requirement for interpretability of trained ML models
    - Current best performing ML models (deep neural nets, SVMs, ensembles) are the least interpretable
      - E.g., it is difficult to understand the parameters' importance in a deep NN with millions of parameters
      - Interpretable ML provides transparency to the internal decision-making process by the models, and explains models' predictions in human-understandable terms
  - Requirement for low false negatives
    - Unlike other ML applications, in cybersecurity even a single false negative (i.e., missed malicious file) can have significant consequences
    - Requires different evaluation approaches, e.g., different metrics to ensure low false negatives
  - Requirement for updating the models continuously
    - The fast evolving pace of adversarial attacks requires updated and more capable models
    - Otherwise, model performance degrades over time

Kaspersky Lab (2020) – ML Methods for Malware Detection

AML in Cybersecurity

Adversarial Machine Learning in Cybersecurity

- ***Adversarial ML in cybersecurity*** is the modeling of non-stationary adversarial settings, where an adversary manipulates (perturbs) the input data, exploiting specific vulnerabilities of ML algorithms in order to compromise the security of the targeted system
- Rosenberg et al. (2021) proposed the taxonomy of AML attacks in cybersecurity shown in the figure below
  - The taxonomy is based on 7 characteristics of AML attacks that are unique to the cybersecurity domain, listed under 4 categories (threat model, attack type, perturbed features, and attack's output)
  - The taxonomy is explained further on next pages

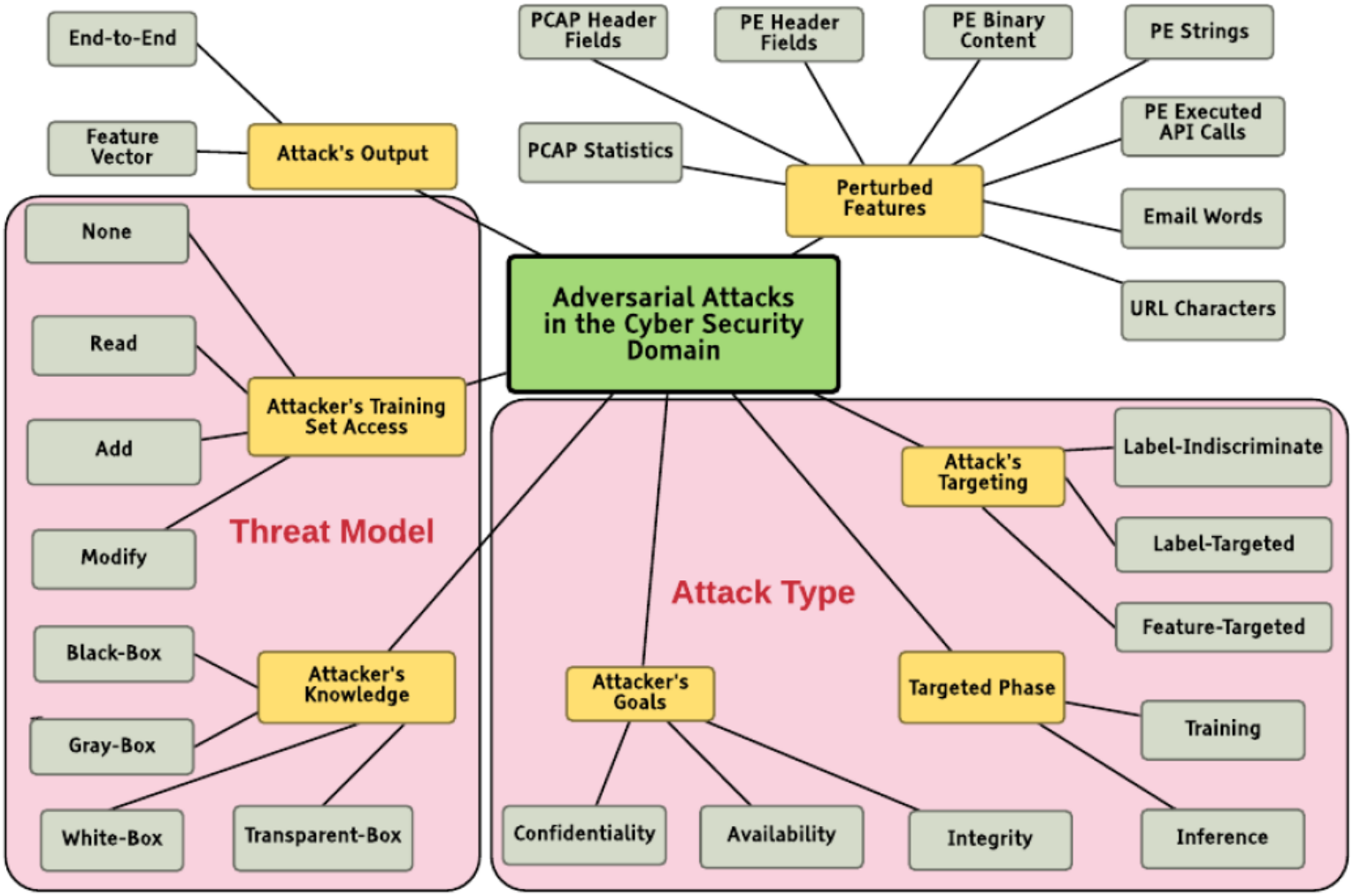


Rosenberg (2021) – AML Attacks and Defense Methods in the Cyber Security Domain

Taxonomy of AML Attacks in Cybersecurity

Adversarial Machine Learning in Cybersecurity

- A detailed overview of the proposed taxonomy by Rosenberg et al. (2021)



Picture from: Rosenberg (2021) – AML Attacks and Defense Methods in the Cyber Security Domain

Taxonomy of AML Attacks in Cybersecurity

Adversarial Machine Learning in Cybersecurity

- ***Threat model*** includes information about: (1) attacker's access to the training set, and (2) attacker's knowledge of the ML model
  - The attacker's training set access can be described as: no access, read data, add new samples, and modify existing samples
  - Based on the attacker's knowledge of the ML model, the attacks can be classified into black-box, white-box, gray-box, and transparent-box attack
    - **Gray-box attack** refers to having access to the confidence scores provided by the classifier (i.e., score-based attack)
    - **Transparent-box attack** means that the adversary has complete knowledge of the ML model, as well as knowledge about the defense methods used by the model
- ***Attacker's goals*** can include:
  - Confidentiality - acquire private information by querying the ML model
    - E.g., stealing the classifier's model
  - Integrity - cause the ML system to perform incorrectly for some or all inputs
    - E.g., causing an ML-based malware classifier to misclassify a malware file as benign
  - Availability - cause the ML system to become unavailable
    - E.g., generate malicious sessions which resemble regular network traffic, causing the ML system to classify legitimate traffic sessions as malicious, and block legitimate traffic

Rosenberg (2021) – AML Attacks and Defense Methods in the Cyber Security Domain

Taxonomy of AML Attacks in Cybersecurity

Adversarial Machine Learning in Cybersecurity

- Based on ***attack's targeting***, the attacks are categorized as:
  - Label-indiscriminate attack - minimize the probability of correctly classifying a perturbed sample (i.e., non-targeted attack)
  - Label-targeted attack – maximize the probability that a specific class is predicted for the perturbed sample (i.e., targeted attack)
  - Feature-targeted attack – input features in the perturbed sample act as triggers for malicious behavior (i.e., backdoor attack)
- In cybersecurity, ML-based systems often use more than one feature type, and hence, attackers often modify more than a single feature
  - ***Perturbed features*** depend on the attacked system, and can include PE header files, PCAP features, words in an email, characters in a URL, etc.
- Based on the ***attack's output***, the attacks can be divided into:
  - Feature-vector attacks, where the input is a feature vector, and the output is a perturbed feature vector
  - End-to-end attacks, where a functional sample is generated as an output (e.g., a spam email, runnable PE file, a phishing URL, etc.)

AML in Cybersecurity vs Computer Vision

Adversarial Machine Learning in Cybersecurity vs Computer Vision

- Most AML research has focused on the computer vision (CV) domain
  - AML in cybersecurity is even more relevant, since there are so many adversaries with specific goals and targets
  - On the other hand, AML in cybersecurity is more challenging
- Differences between ***adversarial attacks in CV versus cybersecurity***
  - Preserving the functionality of perturbed files
    - Any adversarially-perturbed executable file in cybersecurity must preserve its malicious functionality after the modification
    - E.g., modifying an API call or arbitrary byte value might cause the modified executable file to perform a different functionality, or even crash
      - Conversely, in CV modifying pixels' values does not result in an invalid image
  - Small perturbations generated by gradient-based attacks (FGSM, PGD) are difficult to be directly applied to input features in many cybersecurity applications
  - Input samples (e.g., executables) are more complex than images
    - Image files typically have a fixed size (e.g., 28×28 pixels MNIST images), and are easily resized, padded, or cropped
    - Executable files contain different types of input information, and have variable files size (that can range from several KB to several GB)

AML Applications in Cybersecurity

Adversarial Machine Learning in Cybersecurity

- The main AML applications in cybersecurity are in the following areas:
  - Malware detection and classification
  - Network intrusion detection
  - URL detection
  - Spam filtering
  - Cyber-physical systems
  - Industrial control systems
  - Biometric systems
    - Face recognition
    - Speaker verification/recognition
    - Iris and fingerprint systems

Malware Detection and Classification

Malware Detection and Classification

- Malicious software is also known as malware
- ***Malware*** is any kind of software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system or network
  - Malware is constantly evolving and presents significant threat to computer systems
  - Forces security analysts to keep pace by improving cyber defenses
- Automated malware detection versus classification
  - *Malware detection* systems: predict whether an executable file is malware or not a malware
    - Output: 1 or 0
  - *Malware classification* systems: predict the malware type of an executable
    - Output: 1 to *N*, where *N* is the number of different malware families
    - I.e., malware classification systems differentiate between different kinds of malware (virus, adware, or Trojan), in order to provide a better understanding of their capabilities

Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Malware Categories

Malware Detection and Classification

- Depending on the purposes, malware can be divided into various categories
  - Virus: attaches itself to a program and infects a device
  - Worm: self-replicates and propagates copies of itself to other devices over a network
  - Adware: generates/displays unsolicited online advertisements on user's screen
  - Ransomware: locks down an infected device, and demands payment to unlock it
  - Backdoor: allows unauthorized access to functionality
  - Trojan: a class of backdoor malware disguised as legitimate software, to trick users into installing it
  - Bot: distributes malware to other devices, and it is typically part of a network (botnet)
  - Keyloggers: captures keystrokes
  - Rootkit: gains root-level access to conceal the existence of other malware
  - Logic bomb: explodes when a condition occurs

Malware Analysis Systems

Malware Analysis Systems

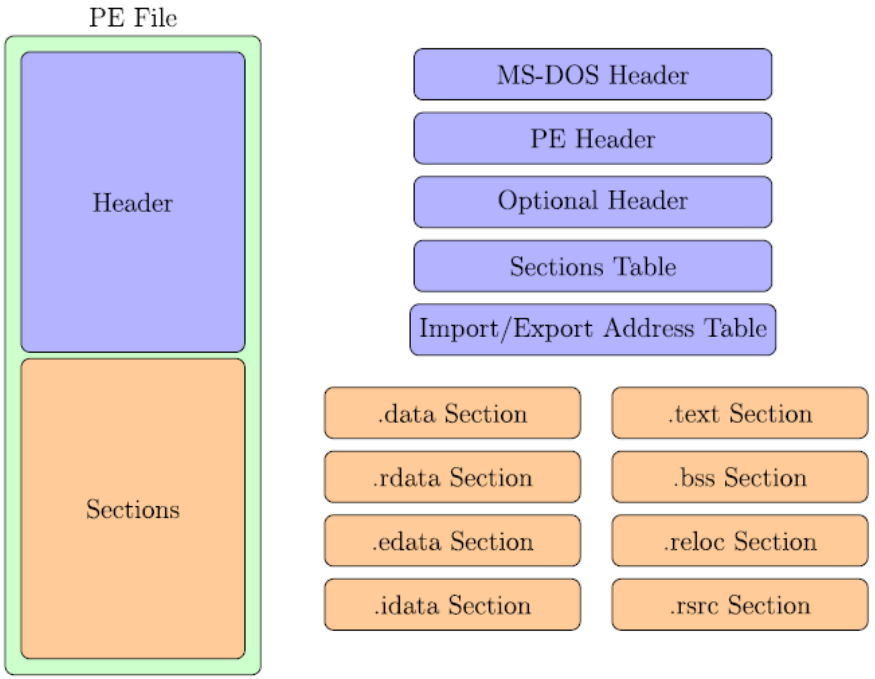
- ***Malware analysis*** involves dissecting malware to understand how it works, and determine its functionality, origin, and potential impact
  - Malware analysis is essential for any business and infrastructure that responds to cybersecurity incidents
- Malware analysis systems can be classified into two broad categories
  - ***Static analysis systems*** (pre-execution analysis)
    - Process malware without running it, and extract features to be used for malware detection and classification
  - ***Dynamic analysis systems*** (post-execution analysis)
    - It involves running the malware either in a physical or virtual environment, and searching for indicators of malicious activities
- Some references also add a class of ***hybrid analysis systems***, that combine static and dynamic analysis

Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Portable Executable (PE) File Format

Malware Analysis Systems

- In Windows systems, ***Portable Executable*** (PE) is a file format for executables
  - Analogous formats are Executable and Linkable Format or ELF (Linux, Unix) and Mach-O (macOS and iOS)
  - Most existing malware targets Windows systems
- A PE file consists of a number of a *header* and *sections* that inform the Windows OS how to manage the executable file

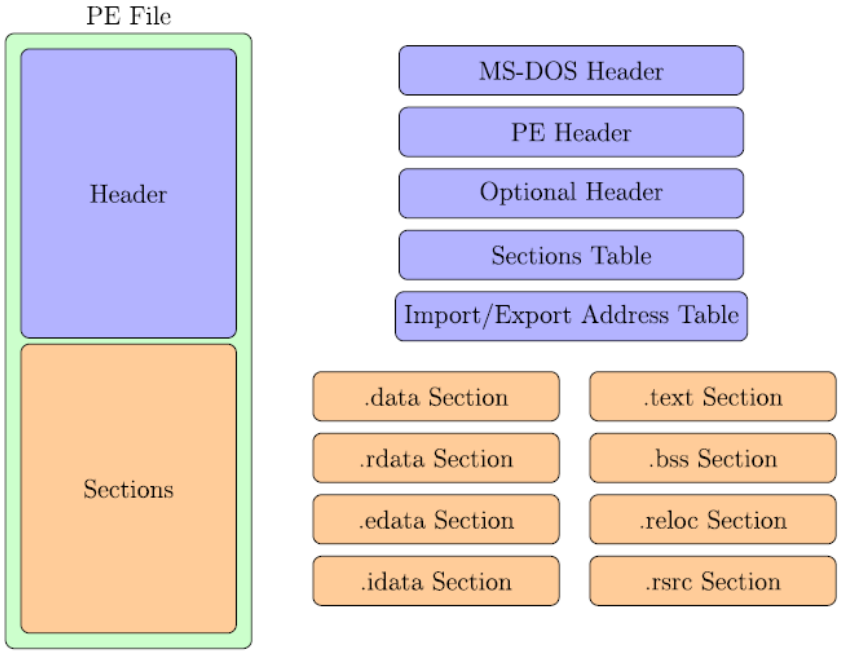


Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Portable Executable (PE) File Format

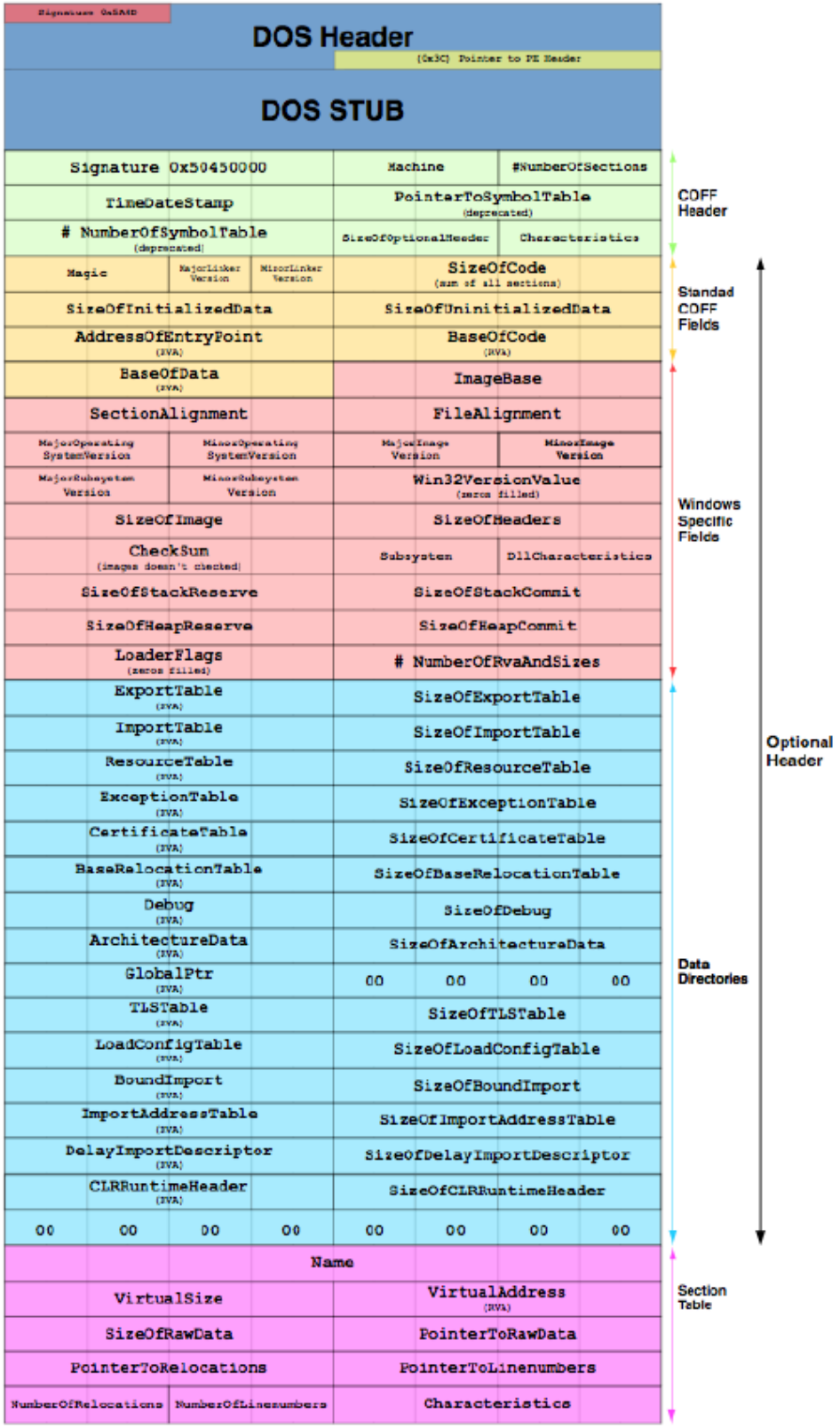
Malware Analysis Systems

- PE file format
  - PE file header
    - The header of the PE file is composed of additional headers (MS-DOS, PE, and Optional Header) and various tables and fields (Sections Table, Import/Export Address Table)
  - PE file sections
    - The sections are either code sections (machine instructions), data sections (holding variables and constants), or resource sections (holding embedded fonts, images, etc.)



Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

- The format for a 32-bit PE file header
- Note the MS-DOS Header on the top, followed by the PE Header (COFF Header, or Common Object File Format Header), Optional Header, Sections Table, etc.
- Detailed description of the PE format can be found at https://docs.microsoft.com/en-us/windows/win32/debug/pe-format



- Extracted features from a PE file for ML analysis in the EMBER dataset
- Note again the header, optional header, imports and exports, sections, etc.


```
"sha256": "000185977be72c8b007ac347b73ceb1ba3e5e4dae4fe98d4f2ea92250f7f580e",
"appeared": "2017-01",
"label": -1,
"general": {
  "file_size": 33334,
  "vsize": 45056,
  "has_debug": 0,
  "exports": 0,
  "imports": 41,
  "has_relocations": 1,
  "has_resources": 0,
  "has_signature": 0,
  "has_tls": 0,
  "symbols": 0
},
"header": {
  "coff": {
    "timestamp": 1365446976,
    "machine": "I386",
    "characteristics": [ "LARGE_ADDRESS_AWARE", ..., "EXECUTABLE_IMAGE" ]
  },
  "optional": {
    "subsystem": "WINDOWS_CUI",
    "dll_characteristics": [ "DYNAMIC_BASE", ..., "TERMINAL_SERVER_AWARE" ],
    "magic": "PE32",
    "major_image_version": 1,
    "minor_image_version": 2,
    "major_linker_version": 11,
    "minor_linker_version": 0,
    "major_operating_system_version": 6,
    "minor_operating_system_version": 0,
    "major_subsystem_version": 6,
    "minor_subsystem_version": 0,
    "sizeof_code": 3584,
    "sizeof_headers": 1024,
    "sizeof_heap_commit": 4096
  }
},
"imports": {
  "KERNEL32.dll": [ "GetTickCount" ],
  ...
},
"exports": []
"section": {
  "entry": ".text",
  "sections": [
    {
      "name": ".text",
      "size": 3584,
      "entropy": 6.368472139761825,
      "vsize": 3270,
      "props": [ "CNT_CODE", "MEM_EXECUTE", "MEM_READ"]
    },
    ...
  ]
},
"histogram": [ 3818, 155, ..., 377 ],
"byteentropy": [0, 0, ... 2943 ],
"strings": {
  "numstrings": 170,
  "avlength": 8.170588235294117,
  "printabledist": [ 15, ... 6 ],
  "printables": 1389,
  "entropy": 6.259255409240723,
  "paths": 0,
  "urls": 0,
  "registry": 0,
  "MZ": 1
},
}
```


Anderson (2018) – EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models

Static Analysis Systems

Malware Analysis Systems

- ***Static analysis*** provides information about the functionality of the file, and it produces a set of signature features (without executing the file)
  - The extracted information is used to predict whether the file is malicious software
  - The disadvantage of static analysis is that the "true features" of the code may be missed
- Static analysis can include:
  - Analyzing PE header and sections
    - PE header provides information about linked libraries and imported/exported functions, as well as contains metadata about the executable
    - Strings of characters can contain references to modified files or accessed file paths by the executable (e.g., URLs, domain names, IP addresses, names of loaded DLLs, registry keys, etc.)
    - Search for packed/encrypted code that is used by malware developers to make their manipulated files more difficult to analyze
  - Disassembling the program – translating machine code into assembly language code
    - Load the executable into a disassembler to translate it into assembly language, and obtain a better understanding of what the program does

Dynamic Analysis Systems

Malware Analysis Systems

- ***Dynamic analysis*** involves executing the program and monitoring its behavior
  - Is typically performed when all available static analysis techniques have been exhausted
- Dynamic analysis is run in a safe environment on dedicated physical or virtual machines (in order not to expose the users' system to unnecessary risks)
  - *Physical machines* are set up on isolated networks, disconnected from the Internet or any other network, to prevent malware from spreading
  - *Virtual machines* emulate the functionality of a physical computer, where the OS running on the virtual machine is isolated from the host OS
    - One limitation is that some malware can detect when they are running in a virtual machine, and they will execute differently than when in a physical machine
  - A related term is *sandbox*, referring to a physical or virtual environment for running malware, which isolates executables from other system resources and applications
    - Although they share characteristics with physical and virtual machines, sandboxes can be more limited (e.g., they can run in the browser), while physical and virtual machines always act as a complete system
    - For example, online sandboxes are websites where one can submit a sample file and receive a report about its behavior

ML-based Malware Classification

ML-based Malware Classification

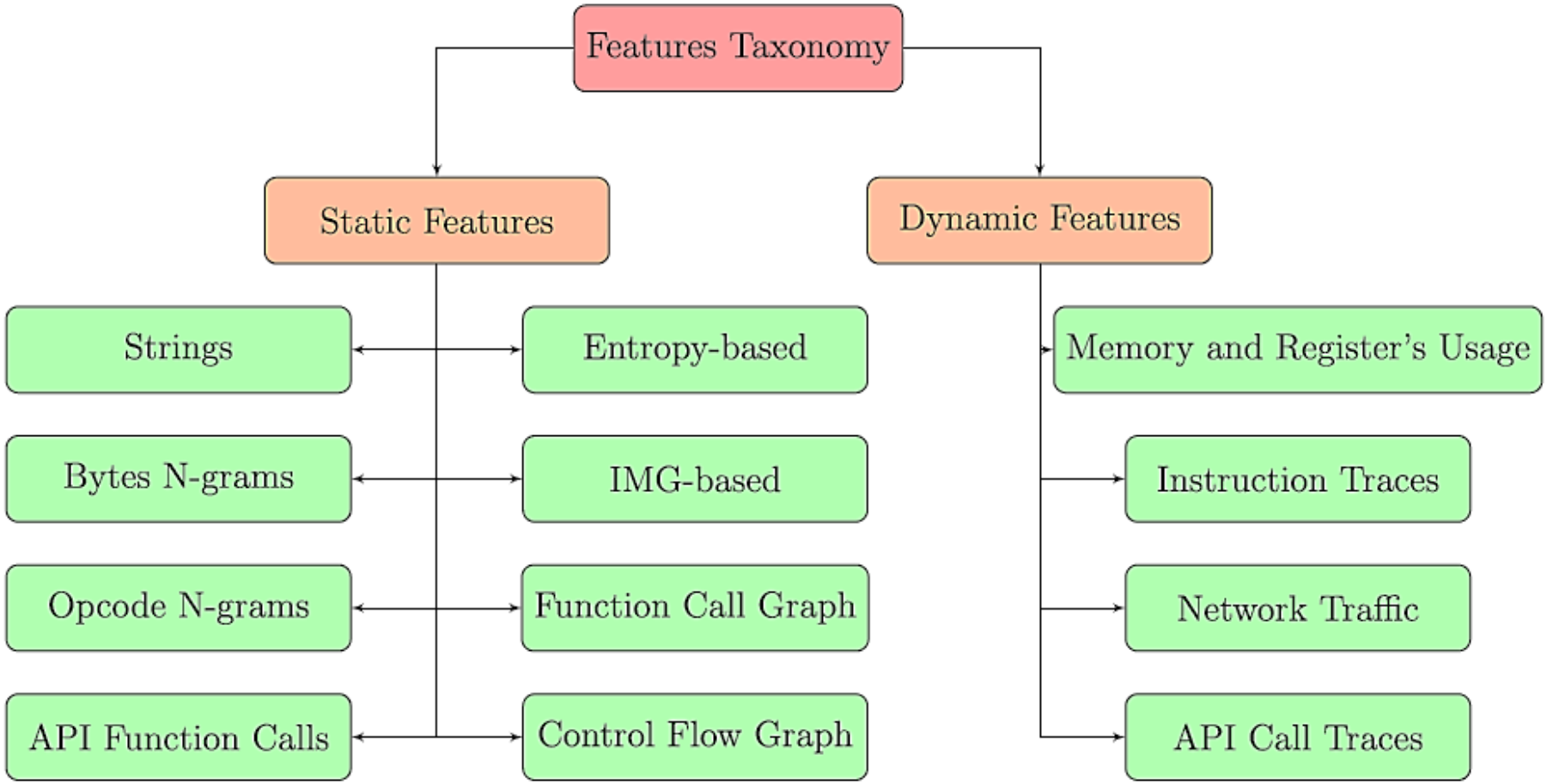
- ***ML-based systems for malware detection and classification*** employ a set of extracted features from executable files
  - As we mentioned earlier, unlike the ML models in CV that employ the intensity of image pixels for image classification, segmentation, or object detection tasks, ML models for malware classification employ a great variety of different features
  - The extracted features are used for training an ML model, and understandably, the trained model is afterwards used for detection or classification of new files
  - In the remainder of the lecture, we will use the term "classification" or "classifier" to describe both ML models for detection and classification of malware
- Based on the inputs used for malware classification, the ML-based systems can be broadly categorized into:
  - ***Raw-binary classifiers*** - use raw byte content from executables as input features
  - ***Feature-based classifiers*** - use either *static* or *dynamic* features, obtained via static or dynamic malware analysis

Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Features for Malware Classification

ML-based Malware Classification

- The figure shows ***static and dynamic features*** that are commonly used for ML-based malware classification



Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Static Features for Malware Classification

Static Features for Malware Classification

- Static features
  - In *Windows* systems, static features are extracted from either the PE file header and sections, or assembly language source file (obtained after disassembling the file)
  - In *Android* systems, static features are extracted from the disassembled APK
    - Various disassembler tools for Windows and Android are available
- Strings – sequence of characters, related to URLs, IP addresses, accessed file paths, registry keys, or names of modified files by the executable
  - Ye et al. (2008) used extracted strings from PE files as input features to an SVM ensemble with bagging model for malware detection
- Byte *n*-grams – sequence of *n* bytes in PE header or the assembly language code
  - An *n-gram* is a sequence of *n* adjacent items in sequential data
  - A large number of sequences of *n* bytes (*n* ranging from 1 to 8) are used as input features for ML model training
  - Different ML models (Decision Trees, Random Forests, Deep Belief Nets) have been implemented using byte *n*-grams , e.g., by Jain and Meena (2011), Yuxin et al. (2019)
  - Challenges include the large number of *n*-grams for each file (which often requires reducing the dimensionality of the feature vectors)

Static Features for Malware Classification

Static Features for Malware Classification

- Opcode (mnemonic) *n*-grams – *n* consecutive opcodes (i.e., operational code instructions) in the assembly language source code
  - Assembly instructions are composed of an operational code and operand
    - E.g., for the instruction sequence: "call sub_401BCD", "add eax 1", "mov ebx ebx", the 3-gram opcode is: CALL-ADD-MOV
  - Malware samples from the same family often use the same opcodes
  - Santos et al. (2013) selected the top 1,000 features using 1 or 2-gram opcodes and trained an SVM malware classifier
- API function call – request to the OS for accessing system resources, such as networking, security, file management, etc.
  - Application Programming Interfaces (API) function calls are very discriminative features, as they can provide key information to reveal the behavior of malware
  - E.g., certain sequences of API function calls are often found in malware, but rarely in benign files
  - Ahmadi et al. (2016) used the frequency of 794 API function calls to develop an ML system for classifying malware into families

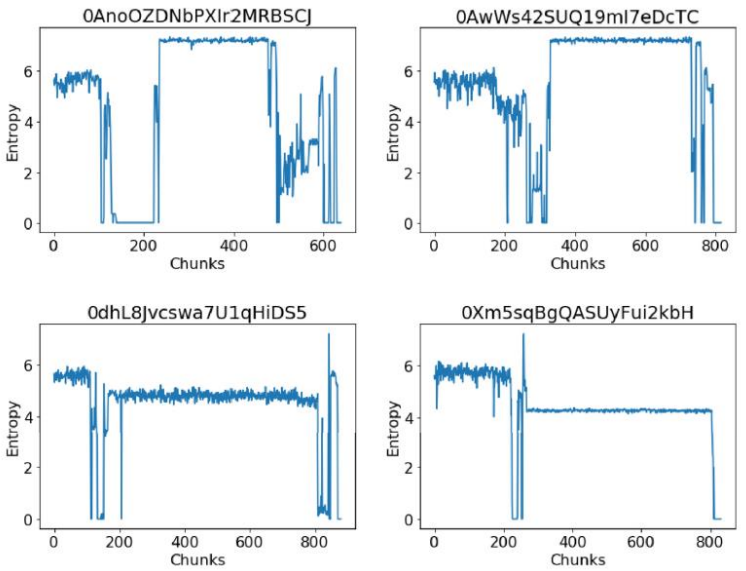
Static Features for Malware Classification

Static Features for Malware Classification

- Entropy-based features – indicate the statistical variation of bytes in a file, and are used to detect compressed or encrypted segments of codes in executables
  - Malware developers use **compression** and **encryption** to conceal malicious segments of code from static analysis
  - Files with compressed or encrypted segments have higher entropy than native code
  - **Structural entropy** represents an executable file as a stream of entropy values, where each value indicates the entropy over a small chunk of code (see the figure below)
  - A similarity score of structural entropies is used for malware classification (e.g., by Sorokin and Jun (2011))

Ramnit family malware

Gatak family malware



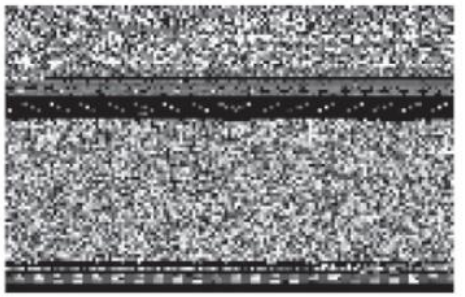
Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Static Features for Malware Classification

Static Features for Malware Classification

- IMG-based features – visualize the binary content of an executable as a gray-scale image
  - This is achieved by interpreting every byte as one pixel of a gray-scale image, and organizing the array of bytes in an executable as a 2-D image
  - Two malware families are shown as gray-scale images in the figure (note that the two families have a distinct image representation)
  - Nataraj et al. (2011) used *k*-Nearest Neighbors to classify malware families

Ramnit family malware



Lollipop family malware

Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Static Features for Malware Classification

Static Features for Malware Classification

- Function–call graph – is a directed graph whose vertices represent the functions of a program, and the edges symbolize function calls
  - Kinable et al. (2011) developed an approach for clustering malware based on the structural similarities between function-call graphs
- Control–flow graph – is a directed graph in which the nodes represent basic blocks, and the edges represent control-flow paths
  - A basic block is a linear sequence of program instructions having an entry point (the first instruction executed) and an exit point (the last instruction executed)
  - The control-flow graph is a representation of all the paths that can be traversed during a program's execution
  - Faruki et al. (2012) used a Random Forest classifier for detecting malware using control-flow graphs of various API calls

Dynamic Features for Malware Classification

Dynamic Features for Malware Classification

- ***Dynamic features*** are extracted from the execution of malware at runtime
- Memory and registers usage – values stored in the memory and different registers during the execution can distinguish benign from malicious programs
  - Ghiasi et al. (2015) monitored the memory content and register values before and after each invoked API call
  - They used similarity scores between the benign and malicious files in a training set to train an ML model for malware detection
- Instruction traces – sequence of processor instructions called during the execution of a program
  - Dynamic instruction traces are more robust indicators of the program's behavior than static traces, since compression and encryption can obfuscate code instructions from static analysis
  - Carlin et al. (2017) analyzed traces of opcodes to detect malware by Random Forest and Hidden Markov Model classifiers

Dynamic Features for Malware Classification

- Network traffic – monitoring the traffic entering and exiting the network can provide helpful information to detect malicious behavior
  - E.g., when malware infects a host machine, it may establish communication with an external server to download updates, other malware, or leak private and sensitive information from the host machine
  - Bekerman et al. (2015) extracted 972 features from the network traffic, and used them for developing Decision Tree and Random Forest malware classifiers
- API call traces – traces for accessing file systems, devices, processes, threads and error handling, and also to access functions such as the Windows registry, manage user accounts, etc.
  - Uppal et al. (2014) proposed traditional ML-based classifiers using *n*-grams of features extracted from traces of invoked API calls

Adversarial Attacks on ML Malware Classifiers

Adversarial Attacks on ML-based Malware Classifiers

- Next, a short overview of the adversarial attacks on ML-based models for malware classification is presented
  - A more detailed review can be found in ***Rosenberg et al. (2021) – Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain***
- AML attacks approaches can generally be divided into two groups:
  - Attacks on traditional ML-based malware classifiers
  - Attacks on deep learning-based malware classifiers

Attacks on Traditional ML Models

Adversarial Attacks on Traditional ML-based Malware Classifiers

- ***Ming et al. (2015) Replacement Attacks: Automatically Impeding Behavior-Based Malware Specifications***
  - This work introduced is an inference integrity gray-box attack
    - In other words, it is an evasion attack, where the adversary's goal is to alter the malware file, in order to result in misclassification by the ML model at inference time
    - Recall that integrity attacks cause the ML system to perform incorrectly (as opposed to attacks that aim to make the system unavailable, or steal private information)
  - The authors modified malware code by replacing API calls with functionality-preserving API calls
  - The attack was employed to evade an ML classifier using Function-Call Graphs features as inputs

Attacks on Traditional ML Models

Adversarial Attacks on Traditional ML-based Malware Classifiers

- ***Xu et al. (2020) MANIS: Evading Malware Detection System on Graph Structure***
  - Inference gray-box attack against Android APK malware classifiers
  - Uses *n*-strongest nodes and FGSM in a Function-Call Graph to generate perturbed samples
- ***Anderson et al. (2018) Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning***
  - Inference black-box attack that employs static features (extracted from the PE header, sections table, and import/export address table) against a Gradient Boosted Decision Tree classifier
  - A reinforcement learning approach was used to generate perturbed samples, by learning the sequence of operations that are likely to result in detection evasion

Attacks on Traditional ML Models

Adversarial Attacks on Traditional ML-based Malware Classifiers

- **Data poisoning attacks**
  - One should note that data poisoning attacks are more challenging in cybersecurity, because they require injecting samples into the training set of the malware classifier
  - But they can have significant consequences, when the adversary has the ability to tamper with the training set
- ***Siciu et al. (2018) When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks***
  - Training integrity gray-box attack, requires data adding access to the training set
  - The attack is against a linear SVM classifier for Android malware detection
  - Data poisoning was done by adding static features (API calls, URL requests) to benign samples
- ***Munoz Gonzalez et al. (2017) Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization***
  - Training integrity black-box attack, requires read and add access to the training set
  - Against Logistic Regression and an NN model for spam and ransomware detection
  - A substitute model was trained and poisoned, and by employing transferability, it was demonstrated that the poisoned samples were effective against the target classifiers

Attacks on Traditional ML Models

Adversarial Attacks on Traditional ML-based Malware Classifiers

- **Attacks on PDF classifiers**
  - Note that PDF malware is less common, and most malware are PE files
- ***Srndic and Laskov (2014) Practical Evasion of a Learning-Based Classifier: A Case Study***
  - Inference integrity gray-box attack, against Random Forest classifier (named PDFRATE) that uses static binary strings as features for detecting malicious PDF files
  - The attack adds new strings features to malicious PDF files, to evade being detected as malware by PDFRATE
- ***Li et al. (2020) A Feature-vector Generative Adversarial Network for Evading PDF Malware Classifiers***
  - Attack on the PDFRATE classifier by using GAN-generated feature vectors for producing malicious PDF files (which were classified as benign by the classifier)
- ***Dang et al. (2017) Evading Classifiers by Morphing in the Dark***
  - Gray-box attack against SVM and Random Forest PDF malware classifiers
  - A hill-climbing optimization approach was used to minimize the score for the rate of feature modifications from malicious and benign PDFs

Deep Learning for Malware Classification

Deep Learning Approaches for Malware Classification

- Besides traditional ML approaches for malware classification that rely on manually selected features based on expert knowledge, recent work has emerged that applied ***Deep Learning methods*** for malware classification
  - DL approaches are more successful in detecting unseen and unsigned malware, but are also more vulnerable to adversarial attacks
- Numerous DL approaches have been introduced, employing raw bytes and static/dynamic malware input features and various network architectures
  - Encoder architectures have often been used in these approaches for extracting salient features and dimensionality reduction of *n*-gram features
  - Convolutional NN models employing IMG-based features and bytes-based features have been applied for malware classification
  - Recurrent NN models have been introduced for capturing dependencies in API call traces, network traffic, and instruction traces
  - Architectures with both convolutional and recurrent layers have also been developed for dealing with both the spatial and sequential nature of static and dynamic features in executables

Gilbert (2020) – The Rise of Machine Learning for Detection and Classification of Malware

Attacks on Deep Learning Models

Adversarial Attacks on Deep Leaning-based Malware Classifiers

- **Attacks on Deep Learning classifiers using raw bytes (raw-binary classifiers)**
  - A limitation of these approaches is that raw byte content is rarely used as features in the next generation anti-virus (NGAV) products
- ***Kreuk et el. (2018) Adversarial Examples on Discrete Sequences for Beating Whole-Binary Malware Detection***
  - Inference white-box attack against MalConv - a CNN model for malware detection using raw byte inputs
  - FGSM attack is used to modify bytes that were inserted between the file's sections
- ***Koloshnaji et al. (2018) Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables***
  - Implemented a similar attack to Kreuk et al. (2018)
  - Analyzed the byte features that are the most impactful for the attack, found that most of these features belong to the PE header
- ***Siciu et al. (2018) Exploring Adversarial Examples in Malware Detection***
  - Developed a related black-box attack, where adversarial files were created by appending bytes from benign PE headers to malware

Attacks on Deep Learning Models

Adversarial Attacks on Deep Leaning-based Malware Classifiers

- **Attacks on Deep Learning classifiers using static or dynamic features (feature-based classifiers)**
- ***Abusnaina et al. (2019) Adversarial Learning Attacks on Graph-based IoT Malware Detection Systems***
  - Inference white-box attack, against a CNN-model for IoT malware classification
  - Uses Control-Flow Graph features of the malware disassembly source code
  - Malicious samples were generated by concatenating CFGs of benign samples
- ***Hu and Tan (2017) Black-Box Attacks against RNN based Malware Detection Algorithms***
  - Inference gray-box attack, against an LSTM classifier trained on the dynamic API call traces of the malware
  - A GAN model with RNN layers was used to generate invalid API calls, which were inserted into the original API call traces

Attacks on Deep Learning Models

Adversarial Attacks on Deep Leaning-based Malware Classifiers

- ***Rosenberg et al. (2020) Generating End-to-End Adversarial Examples for Malware Classifiers Using Explainability***
  - Gray-box attack using 2,351 static features extracted from PE files
  - Transferability in AML is employed, where a substitute ML model is first trained, and it is hoped that the adversarial samples will be transferred to a target ML model
  - Used feature importance approaches from explainable ML to select the minimal set out of 2,351 features that have high impact on the malware classification
  - Algorithm:
    1) Train a substitute NN model on a training set believed to accurately represent the attacked ML-based malware classifier
    2) Select a malware executable file that needs to bypass the attacked malware classifier
    3) Use explainable ML algorithm to calculate features importance for the classification of the malware on the substitute model
    4) For each feature in the set of features that are the easiest to modify, change the feature using the list of predefined values, and select the value that result in the lowest confidence score by the substitute malware classifier
    5) Repeat until a benign classification is achieved by the target malware classifier

Attacks on Deep Learning Models

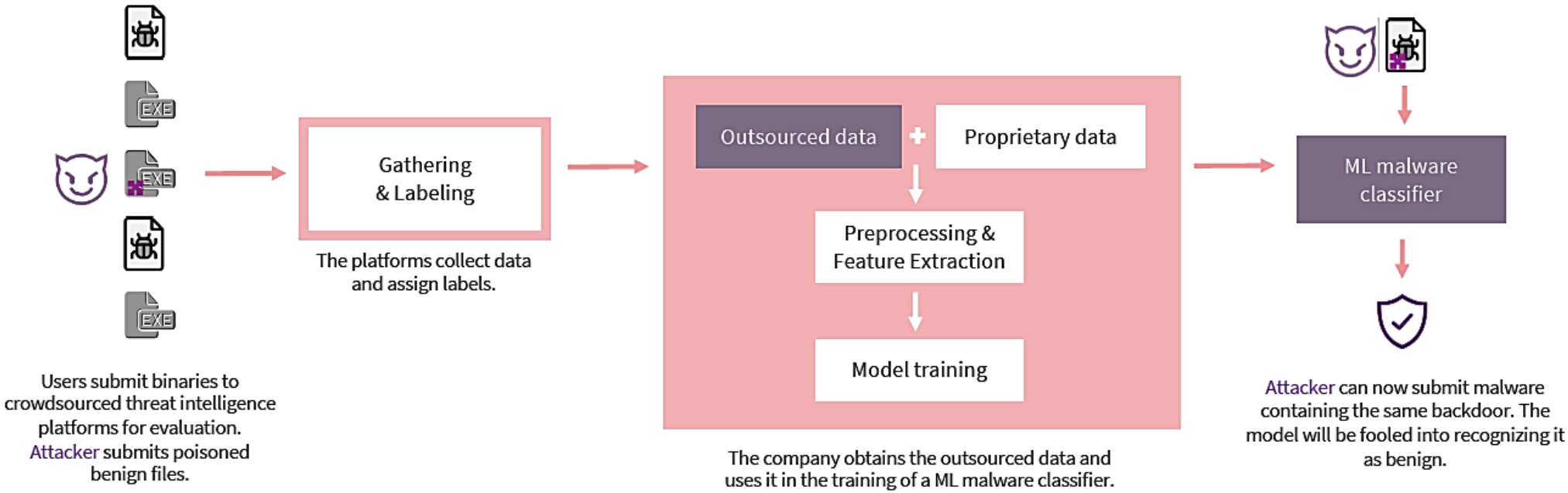
Adversarial Attacks on Deep Leaning-based Malware Classifiers

- ***Severi et al. (2021) Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers***
  - Data-poisoning attacks using a backdoor trigger, on Windows PE files, PDFs, and Android APK files
  - The explainable ML approach SHAP (Shapley Additive Explanations) was used to select a small set of relevant static features for creating the backdoor trigger
  - The attacks were evaluated against several ML malware classifiers: GBM (gradient boosting model), NNs, Random Forest, and SVM
  - Different datasets for malware classification were used
    - Ember dataset (Windows executables) – consisting of 1.1 million PE files, having 2,351 dimensional features
    - Contagio dataset (PDFs) – consisting of 10 thousand PDF files, having 135 dimensional features
    - Drebin dataset (Android APK executables) – consisting of 130 thousand apps, having 545,000 dimensional features

Attacks on Deep Learning Models

Adversarial Attacks on Deep Leaning-based Malware Classifiers

- Severi et el. (2021) cont'd (backdoor poisoning attack)
  - Large volumes of executables are acquired from third-party platforms, and are labeled by pools of existing antivirus engines
  - The outsourced data is combined with some proprietary data to create a training set
  - Data preprocessing and feature extraction steps are followed by model training
  - Malware with backdoor features is classified as benign by the trained model

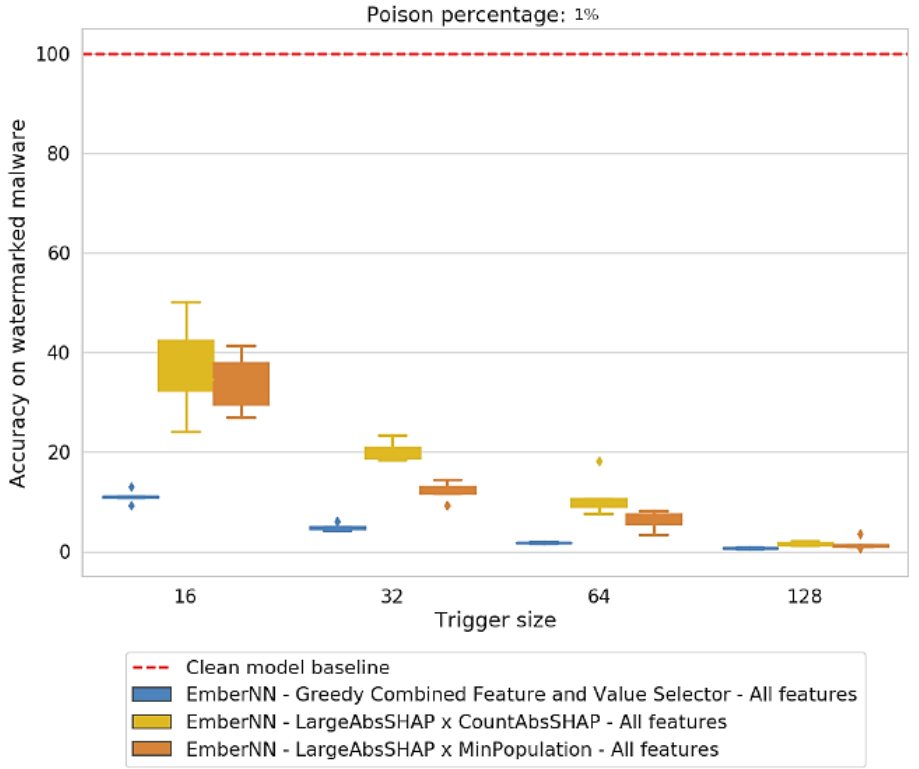


Attacks on Deep Learning Models

Adversarial Attacks on Deep Leaning-based Malware Classifiers

- Severi et el. (2021) cont'd (backdoor poisoning attack)
  - F1 score standard accuracy for four ML-based malware classifiers: GBM, NN, Random Forest, and SVM
    - The accuracy on the original (not-poisoned) files is over 99%
  - Corresponding F1 score accuracy of the attacked NN-classifier
    - With a fixed poisoning rate of 1% of the training set, and varying the trigger size (i.e., the number of modified features) from 16 to 128
    - Accuracies drop to below 1% for 128 modified features

Model	F1 Score	FP rate	FN rate	Dataset
LightGBM	0.9861	0.0112	0.0167	EMBER
EmberNN	0.9911	0.0067	0.0111	EMBER
Random Forest	0.9977	0.0025	0.0020	Contagio
Linear SVM	0.9942	0.0026	0.07575	Drebin



Additional References

1. Rosenberg et al. (2021) – Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain, https://arxiv.org/abs/2007.02407
2. Gilbert et al. (2020) – The Rise of Machine Learning for Detection and Classification of Malware: Research Developments, Trends and Challenges, link
3. Kaspersky Lab (2020) – Machine Learning Methods for Malware Detection, link