University of Idaho

Department of Computer Science

# CS 487/587 Adversarial Machine Learning

*Dr. Alex Vakanski*

# Lecture 10

# AML in Cybersecurity – Part I: Network Intrusion Detection

# Lecture Outline

- Adversarial Machine Learning in cybersecurity
  - Taxonomy of AML attacks in cybersecurity
  - AML in cybersecurity versus computer vision
- Network intrusion detection
  - Goals of NIDS
- Datasets for network intrusion detection
- Anomaly detection with Machine Learning
  - One-class SVM
  - Autoencoders
  - Variational autoencoders
  - GANs
  - Sequence-to-sequence models
- Adversarial attacks on ML-based NIDS
  - Feature-level attacks
  - Packet-level attacks

# ML in Cybersecurity

*Adversarial 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 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

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

# Cybersecurity Challenges

*Adversarial 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 capacity to handle large volumes of data, and ability to automatically identify signature features or rules for attack identification

# ML Specifics in Cybersecurity

*Adversarial Machine Learning in Cybersecurity*

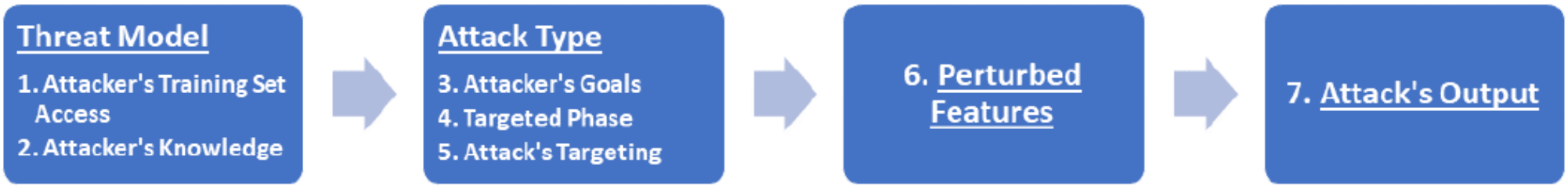
- Application of ML in cybersecurity also introduces unique challenges, including:
  - Requirement for large representative datasets for model training
    - Acquisition of cybersecurity datasets and sample labeling is expensive and time-consuming
    - Small or imbalanced datasets can lead to poor performance (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

Slide credit: Kaspersky Lab (2020) – ML Methods for Malware Detection

# AML in Cybersecurity

*Adversarial Machine Learning in Cybersecurity*

- ***Adversarial ML in cybersecurity*** refers to the setting where an adversary manipulates (perturbs) the input data, in order to exploit specific vulnerabilities of ML algorithms and compromise the security of the targeted system
- Rosenberg et al. (2021) proposed the following 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

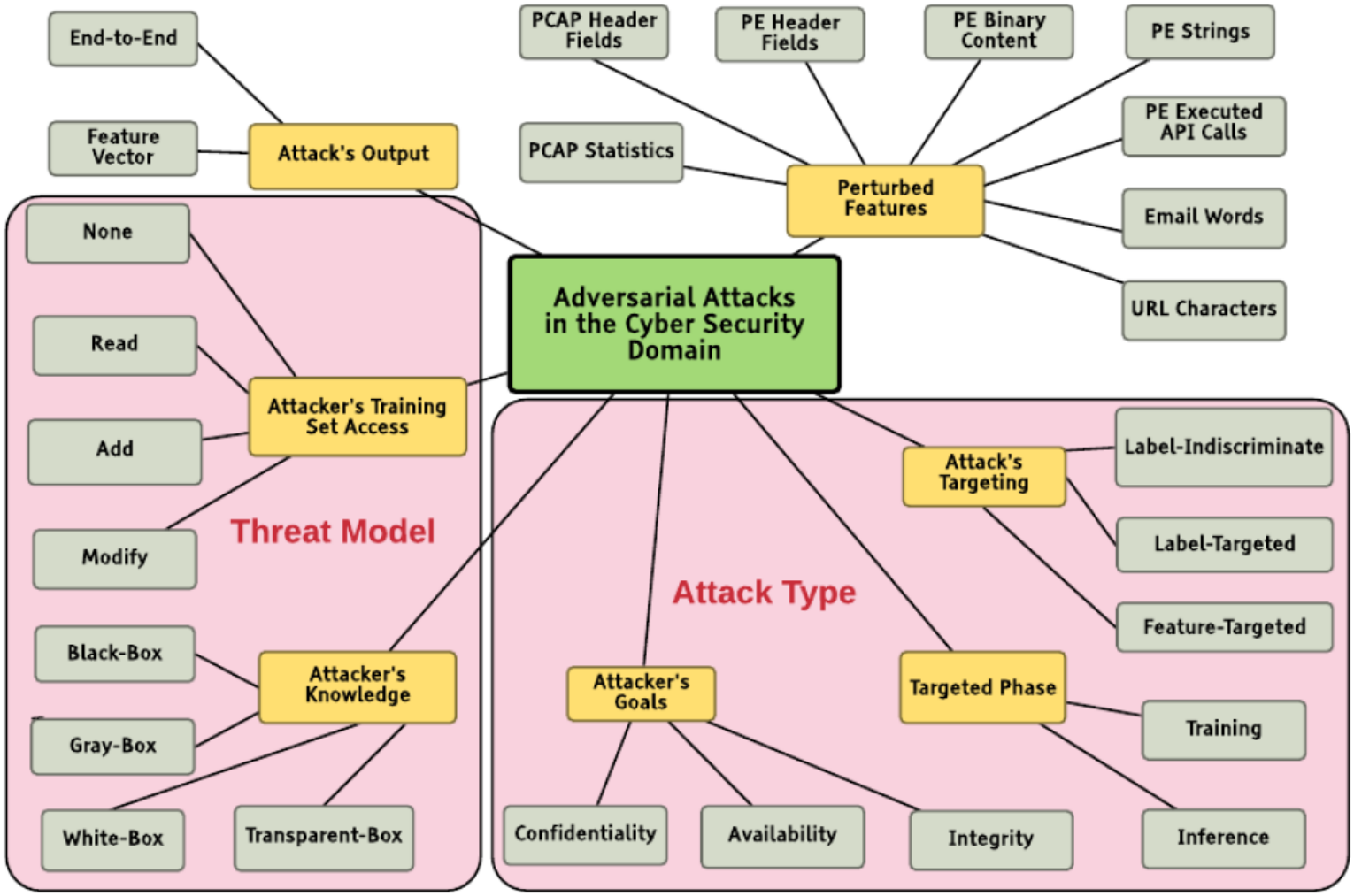



Rosenberg (2021) – AML Attacks and Defense Methods in the Cyber Security Domain

# 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

# Taxonomy of AML Attacks in Cybersecurity

*Adversarial Machine Learning in Cybersecurity*

- Based on ***attack's targeting***, the attacks are categorized as:
  - Label-indiscriminate attack (non-targeted attack) - minimize the probability of correctly classifying a perturbed sample
  - Label-targeted attack (targeted attack) – maximize the probability that a specific class is predicted for the perturbed sample
  - Feature-targeted attack (backdoor trigger attack) – input features in the perturbed sample act as triggers for malicious behavior
- 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 output of the attack is a perturbed feature vector (i.e., a perturbed vector of extracted features from a malware file)
  - End-to-end attacks, where the output of the attack is a generated functional sample (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., in CV modifying pixels' values does not result in an invalid image
      - Conversely, modifying an API call or arbitrary byte value might cause the modified executable file to perform a different functionality, or even crash
  - 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:
  - Network intrusion detection
  - Malware detection and classification
  - URL detection
  - Spam filtering
  - Cyber-physical systems
  - Industrial control systems
  - Biometric systems
    - Face recognition
    - Speaker verification/recognition
    - Iris and fingerprint systems

# Network Intrusion Detection

*Network Intrusion Detection*

- ***Network security*** is critical to every organization, as all computer systems suffer from security vulnerabilities
  - Network security requires solutions in place for protection from the increasing number of cyber threats
  - It is essential for every organization to implement some form of intrusion detection systems that can discover potential threat events early and in a reliable manner
- An ***intrusion*** is a deliberate unauthorized attempt, successful or not, to break into, access, manipulate, or misuse some valuable property, which may result into or render the property unreliable or unusable
- An ***intrusion detection system (IDS)*** is a security tool for detecting unauthorized intrusions into computer systems and networks
  - A security system used to secure networks from unauthorized intrusions is a ***network intrusion detection system (NIDS)***
  - NIDS should prevent possible intrusions by continuously monitoring the network traffic, to detect any suspicious behavior that violates the security policies and compromises the network confidentiality, integrity, and availability

Slide credit: Ahmad (2020) – Network Intrusion Detection System: A Systematic Study of ML and DL Approaches

# Network Intrusion Detection

*Network Intrusion Detection*

- NIDS is implemented in the form of a device or software that monitors all traffic passing through a strategic point in the network for malicious activities
  - It is typically deployed at a single point, for example, it can be connected to the network switch (as in the figure)
    - If malicious behavior is detected, NIDS will generate alerts to the host or network administrators

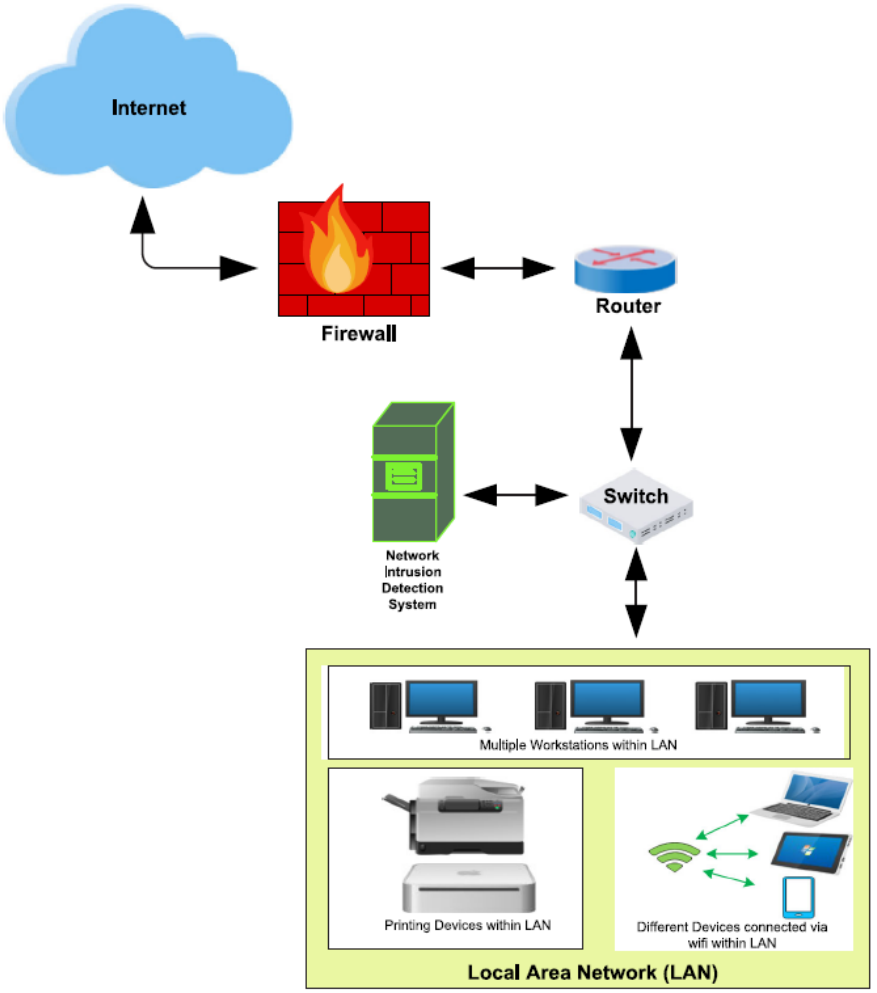



Figure from: Ahmad (2020) – Network Intrusion Detection System: A Systematic Study of ML and DL Approaches

# Goals of NIDS

*Goals of Network Intrusion Detection Systems*

- The main goals of NIDS include:
  1. Detect wide variety of intrusions
     - Previously known and unknown attacks
     - Suggests if there is a need to learn/adapt to new attacks
  2. Detect intrusions in timely fashion
     - And minimize the time spent verifying attacks
     - Depending on the system criticality, it may be required to operate in real-time, especially when the system responds to (and not only monitors) intrusions
       - Problem: analyzing commands may impact the response time of the system
  3. Present the analysis in a simple, easy-to-understand format
     - Ideally as a binary indicator (normal vs malicious activities)
     - Usually the analysis is more complex than a binary output, and security analysts are required to examine suspected attacks
     - The user interface is critical, especially when monitoring large systems
  4. Is accurate
     - Minimize false positives, false negatives

# IDS Categories

*IDS Categories*

- The figure depicts an IDS taxonomy based on the **deployment methods** or **detection methods**
  - ***Deployment methods***
    - ***Host-based IDS*** – deployed to monitor the activities of a single host and scan for security policy violations and suspicious activities
      - Requires information processing for each single node in a network
    - ***Network-based IDS*** – deployed to monitor the activities of all devices connected to a network

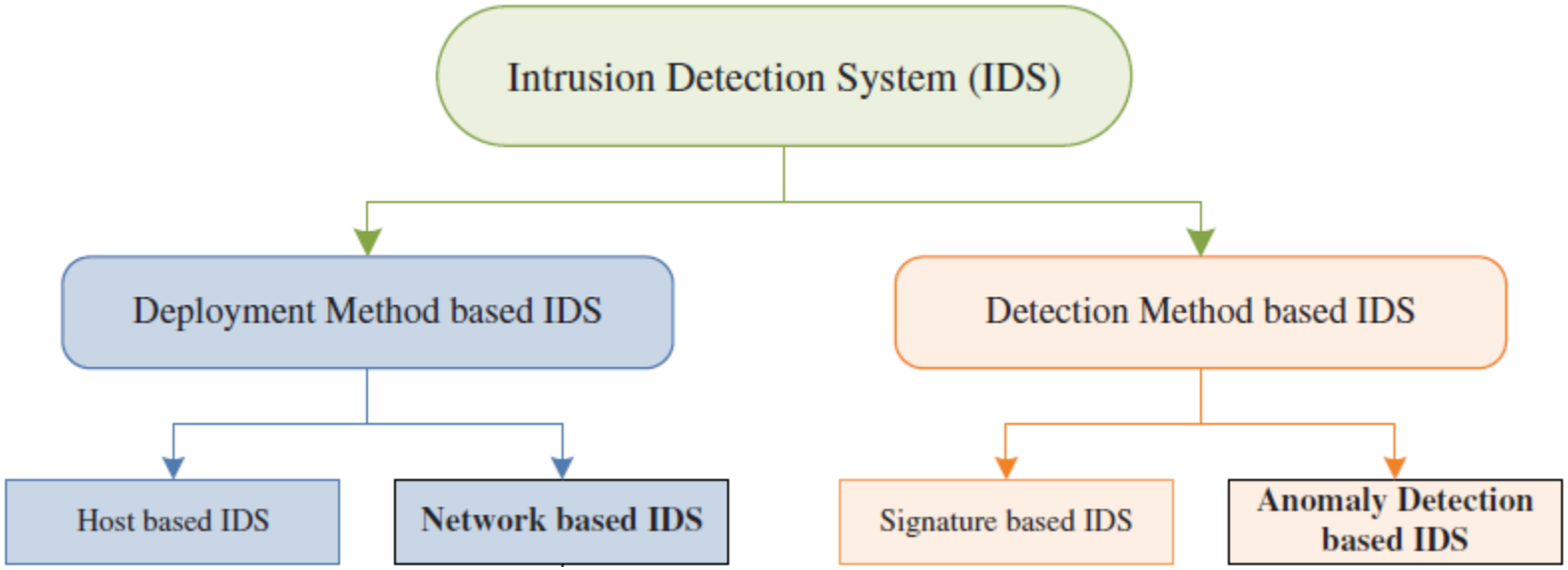



Figure from: Ahmad (2020) – Network Intrusion Detection System: A Systematic Study of ML and DL Approaches

# IDS Categories

*IDS Categories*

- Based on the used ***detection methods***, IDS can be broadly divided into:
  - ***Signature-based systems***
    - These systems are also known as misuse intrusion detection
    - The system compares the incoming traffic with a pre-existing database containing signatures of known attacks
    - Signature databases need to be continuously updated with the most recent attacks
    - Detecting new attacks, for which a signature does not exist, is difficult
  - ***Anomaly-based systems***
    - The system uses statistics to form a baseline (normal) usage of the network at different time intervals
    - Deviations from the baseline usage are considered anomalies
    - The advantage of these systems is that they can detect unknown attacks
    - The main challenge is the high false alarms rate (as it is difficult to find the exact boundary between normal and abnormal behavior)

Cuelogic Technologies Blog - Evaluation of Machine Learning Algorithms for Intrusion Detection System

# NIDS with Machine Learning

*Network Intrusion Detection with Machine Learning*

- Enormous increase in network traffic in recent years and the resulting security threats are posing many challenges for detecting malicious network intrusions
- To address these challenges, ML and DL-based NIDS have been implemented for detecting network intrusions
  - Anomaly detection has been the main focus of these methods, due to the potential for detecting new types of attacks
- In the remainder of the lecture, we will first overview the datasets that are commonly used for training and evaluating ML-based NIDS, followed by a description of the ML models used for anomaly detection, and followed by adversarial attacks on ML models for NIDS

# Datasets for Network Intrusion Detection

*Datasets for Network Intrusion Detection*

- There are several public datasets consisting of records of normal network traffic and network attacks
  - Each record in these datasets represents a network connection data packet
  - The data packets are collected between defined starting and ending times, as data flows to and from a source machine and a target machine under a distinct network communication protocol
- Network connection data packets are saved as ***PCAP (Packet Capture)*** files (i.e., .pcapfile)
  - PCAP files have different formats, e.g., Libpcap (Linux and macOS), WinPcap (Windows), and Npcap (Windows)
  - PCAP files are used for network analysis, monitoring network traffic, and managing security risks
    - The data packets allow to identify network problems
      - E.g., based on data usage of applications and devices
      - Or, identify where a piece of malware breached the network, by tracking the flow of malicious traffic and other malicious communications

# NSL-KDD Dataset

*Datasets for Network Intrusion Detection*

- The most popular dataset for benchmarking ML models for NIDS has been the ***NSL-KDD dataset***
  - It is an updated, cleaned-up version of the original KDD Cup'99 dataset (released in 1999)
- NSL-KDD contains 150 thousand network data from packet records (PCAP files)
- Each record has 41 features, shown in the table
  - The features include duration of the connection, protocol type, data bytes send from source to destination, number of failed logins, etc.
  - The 41 features are either categorical (4), binary (6), discrete (23), or continuous (10)
    - Many approaches use a subset of the 41 features
  - Every record has an associated label (indicating whether it is a normal traffic or attack) and a score (the severity of the traffic, on a scale from 0 to 21)

| No. | Feature Name | No. | Feature Name |
|---|---|---|---|
| 1 | Duration | 22 | is_guest_login |
| 2 | protocol type | 23 | **count** |
| 3 | service | 24 | srv_count |
| 4 | **flag** | 25 | **serror_rate** |
| 5 | src_bytes | 26 | **srv_serror_rate** |
| 6 | **dst_bytes** | 27 | **rerror_rate** |
| 7 | land | 28 | srv_rerror_rate |
| 8 | wrong_fragment | 29 | **same_srv_rate** |
| 9 | urgent | 30 | diff_srv_rate |
| 10 | hot | 31 | srv_diff_host_rate |
| 11 | num_failed_logins | 32 | dst_host_count |
| 12 | logged_in | 33 | **dst_host_srv_count** |
| 13 | num_compromised | 34 | dst_host_same_srv_rate |
| 14 | root_shell | 35 | dst_host_diff_srv_rate |
| 15 | su_attempted | 36 | dst_host_same_src_port_rate |
| 16 | num_root | 37 | dst_host_srv_diff_host_rate |
| 17 | num_file_creations | 38 | **dst_host_serror_rate** |
| 18 | num_shells | 39 | **dst_host_srv_serror_rate** |
| 19 | num_access_files | 40 | dst_host_rerror_rate |
| 20 | num_outbound_cmds | 41 | dst_host_srv_rerror_rate |
| 21 | is_host_login | 42 | |

Table from: Gerry Saporito – A Deeper Dive into the NSL-KDD Data Set

# NSL-KDD Dataset

*Datasets for Network Intrusion Detection*

- Examples of 15 PCAPs (each row is one PCAP file)

| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot | ... | dst_host_diff_srv_rate | dst_host_same_src_port_rate | dst_host_srv_diff_host_rate | dst_host_serror_rate | dst_host_srv_serror_rate |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | tcp | ftp_data | SF | 491 | 0 | 0 | 0 | 0 | 0 | ... | 0.03 | 0.17 | 0.00 | 0.00 | 0.00 |
| 1 | 0 | udp | other | SF | 146 | 0 | 0 | 0 | 0 | 0 | ... | 0.60 | 0.88 | 0.00 | 0.00 | 0.00 |
| 2 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.05 | 0.00 | 0.00 | 1.00 | 1.00 |
| 3 | 0 | tcp | http | SF | 232 | 8153 | 0 | 0 | 0 | 0 | ... | 0.00 | 0.03 | 0.04 | 0.03 | 0.01 |
| 4 | 0 | tcp | http | SF | 199 | 420 | 0 | 0 | 0 | 0 | ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0 | tcp | private | REJ | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.07 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.05 | 0.00 | 0.00 | 1.00 | 1.00 |
| 7 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.07 | 0.00 | 0.00 | 1.00 | 1.00 |
| 8 | 0 | tcp | remote_job | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.05 | 0.00 | 0.00 | 1.00 | 1.00 |
| 9 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.06 | 0.00 | 0.00 | 1.00 | 1.00 |
| 10 | 0 | tcp | private | REJ | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.07 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.07 | 0.00 | 0.00 | 1.00 | 1.00 |
| 12 | 0 | tcp | http | SF | 287 | 2251 | 0 | 0 | 0 | 0 | ... | 0.00 | 0.12 | 0.03 | 0.00 | 0.00 |
| 13 | 0 | tcp | ftp_data | SF | 334 | 0 | 0 | 0 | 0 | 0 | ... | 0.00 | 1.00 | 0.20 | 0.00 | 0.00 |
| 14 | 0 | tcp | name | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.07 | 0.00 | 0.00 | 1.00 | 1.00 |

continued

| dst_host_srv_serror_rate | dst_host_rerror_rate | dst_host_srv_rerror_rate | attack | attack score | label |
|---|---|---|---|---|---|
| 0.00 | 0.05 | 0.00 | normal | 20 | 0 |
| 0.00 | 0.00 | 0.00 | normal | 15 | 0 |
| 1.00 | 0.00 | 0.00 | neptune | 19 | 1 |
| 0.01 | 0.00 | 0.01 | normal | 21 | 0 |
| 0.00 | 0.00 | 0.00 | normal | 21 | 0 |
| 0.00 | 1.00 | 1.00 | neptune | 21 | 1 |
| 1.00 | 0.00 | 0.00 | neptune | 21 | 1 |
| 1.00 | 0.00 | 0.00 | neptune | 21 | 1 |
| 1.00 | 0.00 | 0.00 | neptune | 21 | 1 |
| 1.00 | 0.00 | 0.00 | neptune | 21 | 1 |
| 0.00 | 1.00 | 1.00 | neptune | 21 | 1 |
| 1.00 | 0.00 | 0.00 | neptune | 21 | 1 |
| 0.00 | 0.00 | 0.00 | normal | 21 | 0 |
| 0.00 | 0.00 | 0.00 | warezclient | 15 | 1 |
| 1.00 | 0.00 | 0.00 | neptune | 19 | 1 |

# NSL-KDD Dataset

*Datasets for Network Intrusion Detection*

- The attacks in the NSL-KDD dataset are categorized into 4 classes
  - DoS - Denial of Service, by flooding the server with abnormal amount of traffic
  - Probing - Surveillance and other probing attacks to get information from a network
  - U2R (User to Root) - Unauthorized access of a normal user as a super-user (root)
  - R2L (Remote to Local) - Unauthorized access from a remote machine to gain local access
- The subclasses for each attack are shown below, resulting in 39 attacks

| Classes: | DoS | Probe | U2R | R2L |
|---|---|---|---|---|
| Sub-Classes: | • apache2<br>• back<br>• land<br>• neptune<br>• mailbomb<br>• pod<br>• processtable<br>• smurf<br>• teardrop<br>• udpstorm<br>• worm | • ipsweep<br>• mscan<br>• nmap<br>• portsweep<br>• saint<br>• satan | • buffer_overflow<br>• loadmodule<br>• perl<br>• ps<br>• rootkit<br>• sqlattack<br>• xterm | • ftp_write<br>• guess_passwd<br>• httptunnel<br>• imap<br>• multihop<br>• named<br>• phf<br>• sendmail<br>• Snmpgetattack<br>• spy<br>• snmpguess<br>• warezclient<br>• warezmaster<br>• xlock<br>• xsnoop |
| Total: | 11 | 6 | 7 | 15 |

Table from: Gerry Saporito – A Deeper Dive into the NSL-KDD Data Set

# NSL-KDD Dataset

*Datasets for Network Intrusion Detection*

- The records are divided into Train (125 K instances) and Test subsets (25 K instances)
  - As well as a smaller subset Train+20%, containing 20% of the train records (25 K)
- The number of records per attack class is shown in the table
  - Majority of the records in the Train set are normal traffic (53%)
  - The most common attack in the Train set is DoS (37%), while U2R and R2L occur rarely
  - The Test set contains attack subclasses not seen in the Train set

| Dataset | Number of Records: | | | | | |
|---|---|---|---|---|---|---|
| | Total | Normal | DoS | Probe | U2R | R2L |
| KDDTrain+20% | 25192 | 13449 (53%) | 9234 (37%) | 2289 (9.16%) | 11 (0.04%) | 209 (0.8%) |
| KDDTrain+ | 125973 | 67343 (53%) | 45927 (37%) | 11656 (9.11%) | 52 (0.04%) | 995 (0.85%) |
| KDDTest+ | 22544 | 9711 (43%) | 7458 (33%) | 2421 (11%) | 200 (0.9%) | 2654 (12.1%) |

Table from: Gerry Saporito – A Deeper Dive into the NSL-KDD Data Set

# CSE-CIC-IDS2018 Dataset

*Datasets for Network Intrusion Detection*

- ***CSE-CIC-IDS2018 dataset*** was collected with an attacking infrastructure consisting of 50 machines, and a victim infrastructure of 420 machines and 30 servers
  - The testbed includes both Windows and Linux machines
  - It is a collaborative project between the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC)
  - Link to the dataset
  - It is a more recent dataset, in comparison to the most popular KDD Cup'99 dataset
- The dataset includes the network traffic records (PCAP files) and system logs of each machine, captured with the CICFlowMeter-V3 device
  - The records have 80 network traffic features, which include duration, number of packets, number of bytes, length of packets, etc.
- There are 7 types of attack (details about the attacks are presented on the next two pages)

Table from: https://www.unb.ca/cic/datasets/ids-2018.html

# CSE-CIC-IDS2018 Dataset

*Datasets for Network Intrusion Detection*

- Brute-force attack – submit many passwords to guess login information
- Heartbleed attack – scan for vulnerable applications (e.g., OpenSSL), and exploit them to retrieve the memory of the web server (can include passwords, credit card numbers, private email or social media messages)
- Botnet attack - Zeus and Ares malware used for requesting screenshots from infected devices every 7 minutes, and stealing information by keystroke logging
- DoS attack - Slowloris Denial of Service attack allows a single device to take down the web server of another device, by overwhelming it with network traffic
- DDoS attack - Low Orbit in Cannon (LOIC) Distributed Denial of Service attack used 4 devices to take down the web server of a target device
- Web attacks – scan a website for vulnerable applications, and conduct SQL injection, command injection, and unrestricted file upload
- Infiltration of the network from inside attack – a vulnerable application (e.g., PDF Reader) is sent via a malicious email attachment, and if exploited, it is followed by IP sweep, full port scan, and service enumerations

# CSE-CIC-IDS2018 Dataset

*Datasets for Network Intrusion Detection*

- Attacks in the CSE-CIC-IDS2018 dataset

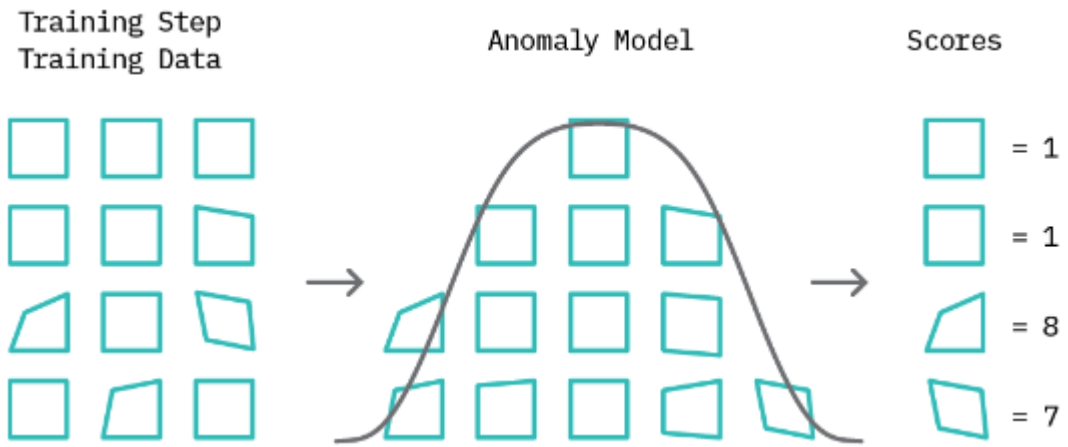
| Attack | Tools | Duration | Attacker | Victim |
|---|---|---|---|---|
| Bruteforce attack | FTP – Patator<br><br>SSH – Patator | One day | Kali linux | Ubuntu 16.4 (Web Server) |
| DoS attack | Hulk, GoldenEye,<br><br>Slowloris, Slowhttptest | One day | Kali linux | Ubuntu 16.4 (Apache) |
| DoS attack | Heartleech | One day | Kali linux | Ubuntu 12.04 (Open SSL) |
| Web attack | • Damn Vulnerable Web App (DVWA)<br>• In-house selenium framework (XSS and Brute-force) | Two days | Kali linux | Ubuntu 16.4 (Web Server) |
| Infiltration attack | • First level: Dropbox download in a windows machine<br>• Second Level: Nmap and portscan | Two days | Kali linux | Windows Vista and Macintosh |
| Botnet attack | • Ares (developed by Python): remote shell, file upload/download, capturing<br>• screenshots and key logging | One day | Kali linux | Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit) |
| DDoS+PortScan | Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests | Two days | Kali linux | Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit) |

Table from: https://www.unb.ca/cic/datasets/ids-2018.html

# Anomaly Detection with Machine Learning

*Anomaly Detection with Machine Learning*

- An ***anomaly*** is a data point or pattern in data that does not conform to a notion of normal behavior
  - Anomalies are also often referred to as outliers, abnormalities, or deviations
- ***Anomaly detection*** is finding such patterns in data that do not adhere to expected normal behavior, given previous observations
  - Anomaly detection has applications in many other domains besides network intrusion detection, including medical diagnostics, financial fraud protection, manufacturing quality control, marketing and social media analytics, etc.
- Approach: first model normal behavior, and then exploit it to identify anomalies




Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Anomaly Detection with Machine Learning

*Anomaly Detection with Machine Learning*

- Anomaly detection can be addressed as:
  - ***Supervised learning*** task – train a classification model using labeled normal and abnormal samples
    - E.g., signatures of normal and abnormal samples can be used as features for training a classifier, and at inference, the classifier can be used to flag abnormal samples
    - This approach assumes access to labeled examples of all types of anomalies that could occur
  - ***Unsupervised learning*** task – train a model using only unlabeled normal samples, to learn the structure of the normal data
    - At inference, any sample that is significantly different than the normal behavior is flagged as an anomaly
  - ***Semi-supervised learning*** task – train a model using many unlabeled samples and a few labeled samples
    - E.g., train a model in unsupervised way using many samples (presumably most of which are normal), and afterward fine-tune the model by using a small number of labeled normal and abnormal samples

Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Anomaly Detection with Machine Learning

*Anomaly Detection with Machine Learning*

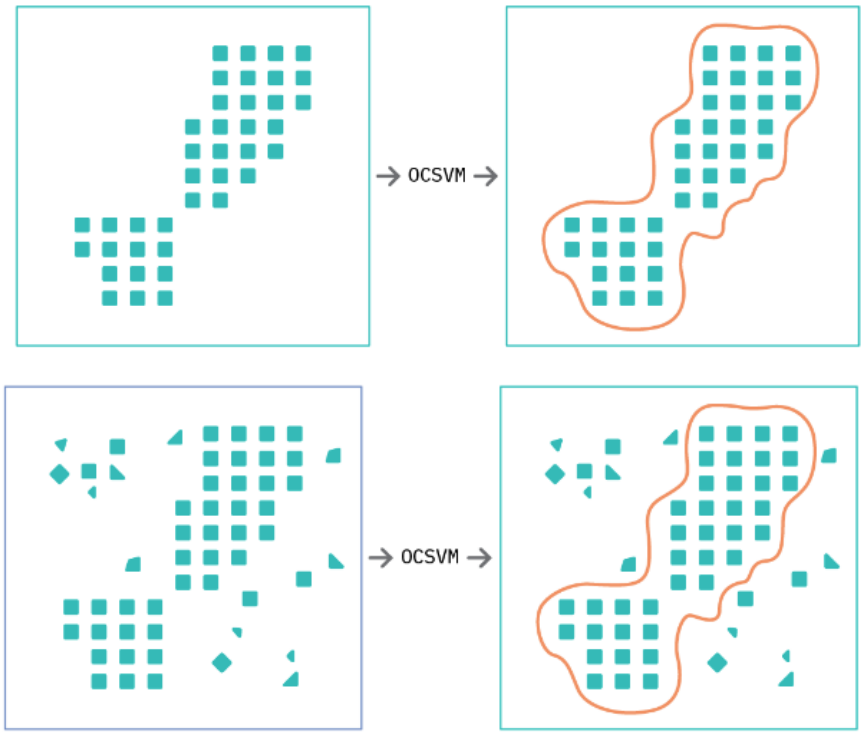
- Various conventional Machine Learning approaches have been employed for anomaly detection
  - Clustering approaches: *k*-means clustering, SOM (self-organizing maps), EM (expectation maximization)
  - Nearest neighbor approaches: *k*-nearest neighbors
  - Classification approaches (One-class SVM)
  - Statistical approaches (HMM, regression models)
- State-of-the-art results in anomaly detection have been typically reported by Deep Learning approaches
  - Due to the capacity to model complex dependencies in multivariate and high-dimensional data
  - These approaches commonly fall in the following categories:
    - Autoencoders
    - Variational autoencoders
    - GANs
    - Sequence-to-sequence models

Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# One-Class SVM for Anomaly Detection

*Anomaly Detection with Machine Learning*

- ***One-class SVM (OCSVM)*** for anomaly detection is a variant of SVM designed for learning a decision boundary around normal data instances
- Approach:
  1. Train the OCSVM model on normal data (to model normal behavior)
  2. At inference, for an input instance calculate the distance to the decision boundary (i.e., the separating hyperplane)
  3. If the distance is positive then label the instance as normal data, and if it is negative then label it as abnormal data (anomaly)

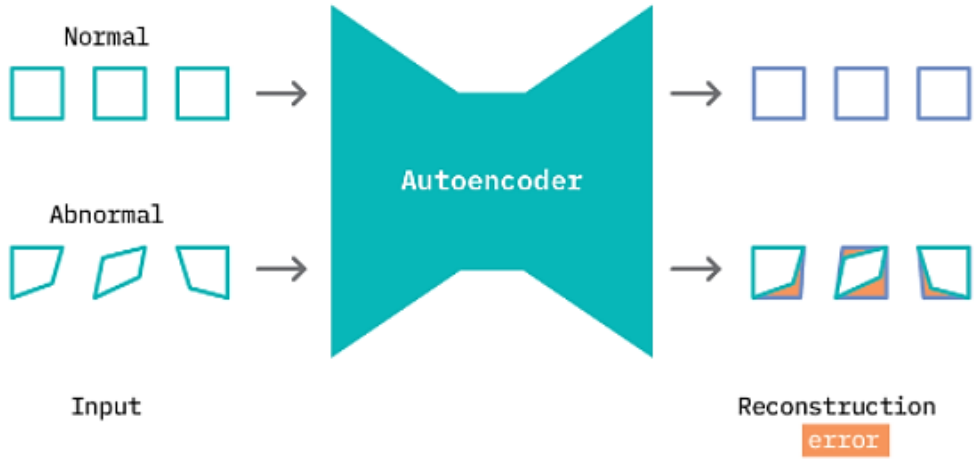



Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Autoencoders for Anomaly Detection

*Anomaly Detection with Machine Learning*

- ***Autoencoders*** (AE)
  - An encoder maps inputs into a lower-dimensional representation (code, latent or encoded representation, embedding), and a decoder reconstructs the original inputs
- Approach:
  1. Train the autoencoder on normal data (to model normal behavior)
  2. At inference, calculate the reconstruction error: e.g., RMSE deviation between the input instance and the corresponding reconstructed output
  3. If the reconstruction error is less than a threshold then label the instance as normal data, if it is greater than the threshold then label it as abnormal data (anomaly)
     - The manually-selected threshold value allows the user to tune the "sensitivity" to anomalies




Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Autoencoders for Anomaly Detection

*Anomaly Detection with Machine Learning*

- Use of autoencoder model for anomaly detection: airspeed during a takeoff
  - The orange line is anomalous speed, the green lines are normal speeds

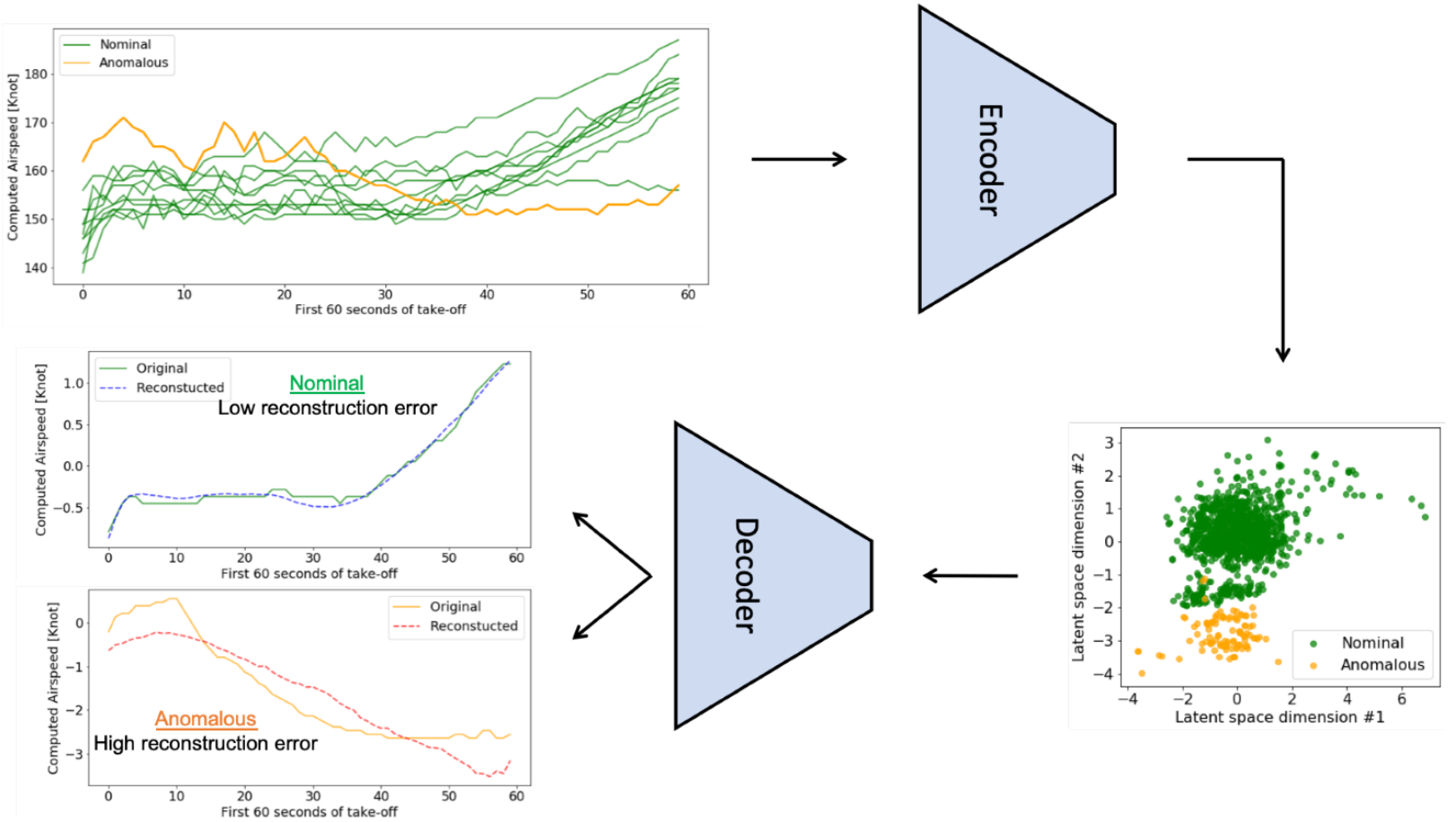
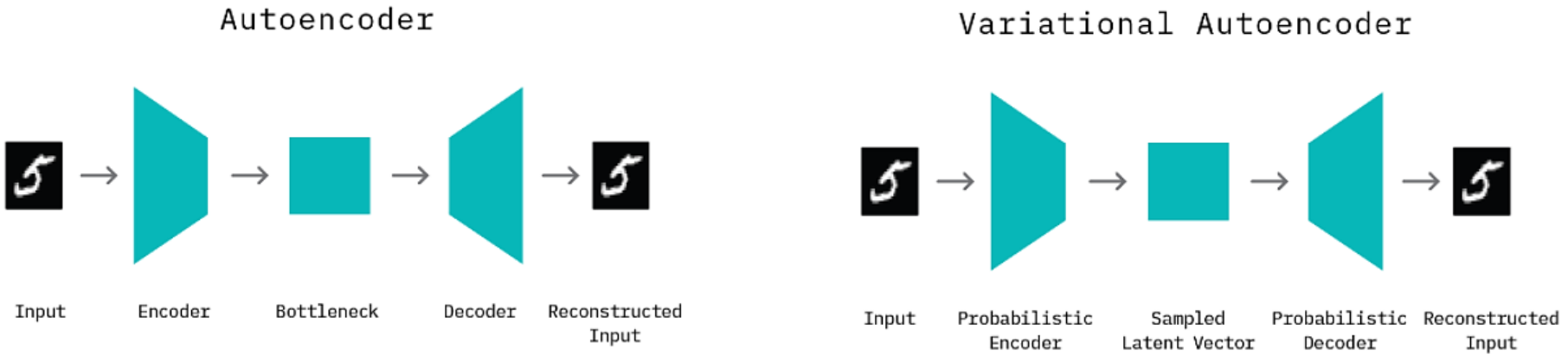



Figure from: Memarzadeh (2020) Unsupervised Anomaly Detection in Flight Data Using Convolutional Variational Auto-Encoder

# Variational Autoencoders for Anomaly Detection

*Anomaly Detection with Machine Learning*

- ***Variational autoencoders*** (VAE) learn a mapping from input data to a distribution
  - I.e., the encoder network learns the parameters (mean and variance) of a distribution
  - The decoder network learns to reconstruct the original data by sampling from the distribution
  - Typically, a Gaussian distribution is used to model the reconstruction space
- VAE are trained by minimizing the KL-divergence between the estimated distribution by the model and the distribution of the real data
  - VAE are also generative models, since they can generate new instances (by sampling from the latent code and reconstructing the sampled data)

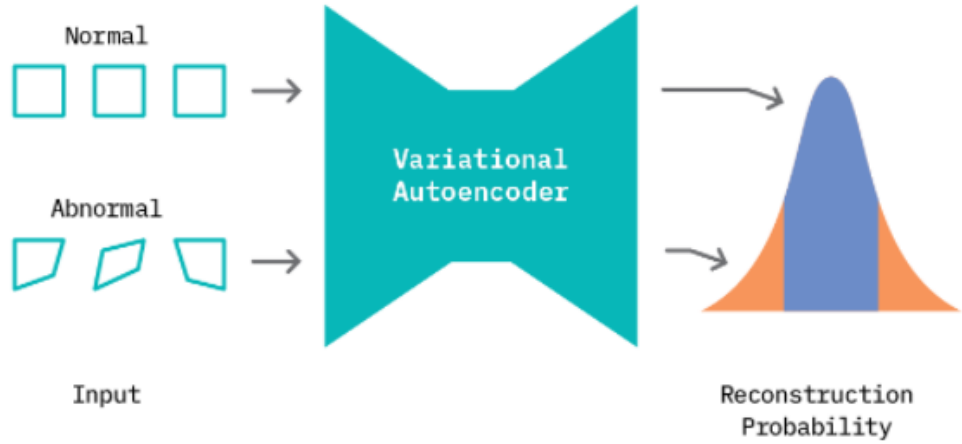



Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Variational Autoencoders for Anomaly Detection

*Anomaly Detection with Machine Learning*

- Approach 1 (similar to the AE approach):
  1. Train the VAE model on normal data instances (to model normal behavior)
  2. At inference, calculate the reconstruction error: e.g., RMSE deviation between the input instance and the reconstructed output of the corresponding sample code
  3. If the reconstruction error is less than a threshold then label the instance as normal data, if it is greater than the threshold then label it as abnormal data (anomaly)
- Approach 2:
  1. Train the VAE model on normal data instances (to model normal behavior)
  2. At inference, calculate the mean and variance from the decoder, and calculate the probability that a new instance belongs to the distribution
  3. If the data instance lies in a low-density region (i.e., below some threshold), it is labeled as abnormal data (anomaly)

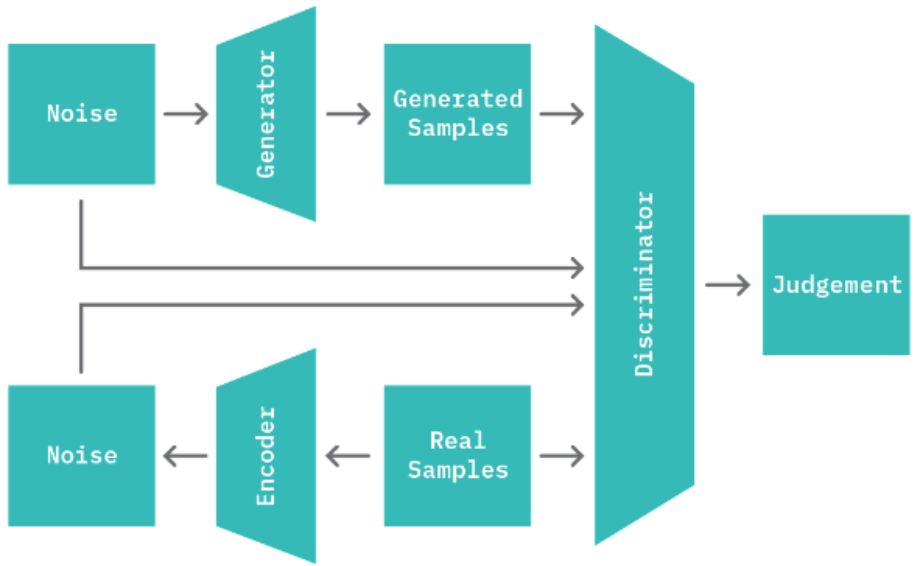



Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# GANs for Anomaly Detection

*Anomaly Detection with Machine Learning*

- Several works used GANs for learning the distribution of normal samples
  - The architecture called ***BiGAN (Bidirectional GAN)*** is commonly used for anomaly detection
  - E.g., ***Akcay et al. (2018) GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training*** (link)
- In BiGAN:
  - A Generator takes as inputs random noise vectors $Z$, and generate synthetic samples $\bar{X}$
  - An additional Encoder is added that learns the reverse mapping – how to generate a fixed noise vector $\bar{Z}$ given a real sample $X$
  - The Discriminator takes as inputs both real samples $X$ and synthetic samples $\bar{X}$, as well as latent noise vectors $Z$ (from the Generator) and $\bar{Z}$ (from the Encoder)

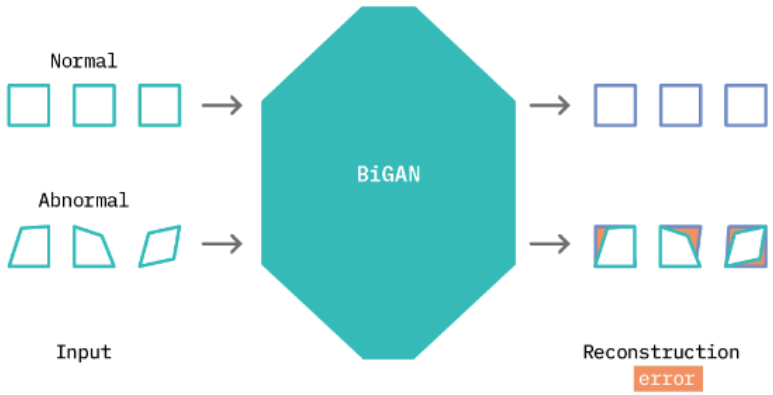



Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# GANs for Anomaly Detection

*Anomaly Detection with Machine Learning*

- Approach:
  1. Train the BiGAN model on normal data instances (to model normal behavior)
  2. At inference, for a real data instance $X$, from the Encoder obtain a latent vector $\bar{Z}$
  3. The noise vector is $\bar{Z}$ is fed to the Generator to yield a synthetic sample $\bar{X}$
  4. Calculate the reconstruction error: e.g., RMSE deviation between the real data instance $X$ and the corresponding synthetic sample $\bar{X}$
  5. Calculate the loss of the Discriminator, i.e., cross-entropy of predictions for $X$ and $\bar{X}$
  6. Calculate an anomaly score as a weighted sum of the reconstruction error and the loss of the Discriminator
  7. If the anomaly score is less than a threshold then label the instance as normal data, if it is greater than the threshold then label it as abnormal data (anomaly)

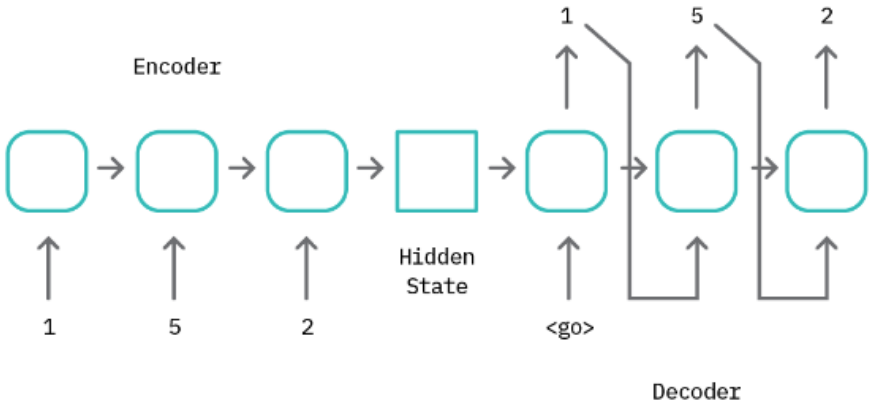



Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Sequence-to-sequence Models for Anomaly Detection

*Anomaly Detection with Machine Learning*

- ***Sequence-to-sequence models*** are designed to learn mappings between sequential data (e.g., time-series signals)
- Sequence-to-sequence models typically consist of an Encoder that generates a hidden representation of the input tokens, and a Decoder that takes in the encoder representation and sequentially generates a set of output tokens
  - The encoder and decoder are typically composed of ***recurrent layers***, such as RNN, LSTM, or GRU, since recurrent networks are particularly suitable for modeling temporal relationships within input data tokens
  - Transformer Networks are also sequence-to-sequence models
- The anomaly detection approach is similar to the Autoencoder models, i.e., based on the reconstruction errors for new examples




Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Anomaly Detection with Machine Learning

*Anomaly Detection with Machine Learning*

- The table lists the pros and cons of the described ML approaches for anomaly detection

| Model | Pros | Cons |
|---|---|---|
| AutoEncoder | • Flexible approach to modeling complex non-linear patterns in data | • Does not support variational inference (estimates of uncertainty)<br>• Requires a large dataset for training |
| Variational AutoEncoder | • Supports variational inference (probabilistic measure of uncertainty) | • Requires a large amount of training data, training can take a while |
| GAN (BiGAN) | • Supports variational inference (probabilistic measure of uncertainty)<br>• Use of discriminator signal allows better learning of data manifold[7] (useful for high dimensional image data).<br>• GANs trained in semi-supervised learning mode have shown great promise, even with very few labeled data[8] | • Requires a large amount of training data, and longer training time (epochs) to arrive at stable results[9]<br>• Training can be unstable (GAN mode collapse) |
| Sequence-to-Sequence Model | • Well suited for data with temporal components (e.g., discretized time series data) | • Slow inference (compute scales with sequence length which needs to be fixed)<br>• Training can be slow<br>• Limited accuracy when data contains features with no temporal dependence<br>• Supports variational inference (probabilistic measure of uncertainty) |
| One Class SVM | • Does not require a large amount of data<br>• Fast to train<br>• Fast inference time | • Limited capacity in capturing complex relationships within data<br>• Requires careful parameter selection (kernel, nu, gamma) that need to be carefully tuned.<br>• Does not model a probability distribution, harder to compute estimates of confidence. |

Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Benchmarking Models for Anomaly Detection

*Anomaly Detection with Machine Learning*

- Performance by the presented models evaluated using the NSL-KDD dataset
  - The best performance was achieved by BiGAN and Autoencoder

| Method | Encoder | Decoder | Other Parameters |
|---|---|---|---|
| PCA | NA | NA | 2 Component PCA |
| OCSVM | NA | NA | Kernel: Rbf, Outlier fraction: 0.01; gamma: 0.5. Anomaly score as distance from decision boundary. |
| Autoencoder | 2 hidden layers [15, 7] | 2 hidden layers [15, 7] | Latent dimension: 2Batch size: 256Loss: Mean squared error |
| Variational Autoencoder | 2 hidden layers [15, 7] | 2 hidden layers [15, 7] | Latent dimension: 2<br>Batch size: 256<br>Loss: Mean squared error + KL divergence |
| Sequence to Sequence Model | 1 hidden layer, [10] | 1 hidden layer [20] | Bidirectional LSTMs<br>Batch size: 256<br>Loss: Mean squared error |
| Bidirectional GAN | Encoder: 2 hidden layers [15, 7], Generator: 2 hidden layers [15, 7] | Generator: 2 hidden layers [15, 7] Discriminator: 2 hidden layers [15, 7] | Latent dimension: 32<br>Loss: Binary Cross Entropy<br>Learning rate: 0.1 |

| Method | Model Size (KB) | Inference Time (Seconds) | # of Parameters | Total Training Time (Seconds) |
|---|---|---|---|---|
| BiGAN | 47.945 | 1.26 | 714 | 111.726 |
| Autoencoder | 22.008 | 0.279 | 842 | 32.751 |
| OCSVM | 10.77 | 0.029 | NA | 0.417 |
| VAE | 23.797 | 0.391 | 858 | 27.922 |
| Seq2Seq | 33.109 | 400.131 | 2741 | 645.448 |
| PCA | 1.233 | 0.003 | NA | 0.213 |

| Method | ROC AUC | Accuracy | Precision | Recall | F1 score | F2 Score |
|---|---|---|---|---|---|---|
| BiGAN | 0.972 | 0.962 | 0.857 | 0.973 | 0.911 | 0.947 |
| Autoencoder | 0.963 | 0.964 | 0.867 | 0.968 | 0.914 | 0.945 |
| OCSVM | 0.957 | 0.949 | 0.906 | 0.83 | 0.866 | 0.844 |
| VAE | 0.946 | 0.93 | 0.819 | 0.836 | 0.827 | 0.832 |
| Seq2Seq | 0.919 | 0.829 | 0.68 | 0.271 | 0.388 | 0.308 |
| PCA | 0.771 | 0.936 | 0.977 | 0.699 | 0.815 | 0.741 |

Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Considerations for Anomaly Detection

*Anomaly Detection with Machine Learning*

- Imbalanced datasets
  - Normal data samples are more readily available than abnormal samples
  - Consequently, the model may perform poorly on abnormal samples
  - Remedy: collect more data, or consider using precision, recall, F1 metrics
- Definition of anomaly
  - The boundary between normal and anomalous behavior can evolve over time
  - It may require retraining the models to adopt to the changes in the data distribution
- False alarms
  - Many of the found anomalies could correspond to noise in the data
  - False alarms require human review of the cases, which increases the costs
- Computational complexity
  - Anomaly detection can require low latency (DL models are computationally intensive)
  - This may impose a trade-off between performance and accuracy

Blog: Cloudera Fast Forward – Deep Learning for Anomaly Detection

# Adversarial Attacks on NIDS



- ***Feature-level (feature vector) attacks on ML-based NIDS***
  - Feature-level attacks are achieved by perturbing a vector of extracted features from PCAP files: the generated adversarial samples are feature vectors
  - Although such adversarial attacks can be successful in evading ML models trained on datasets of extracted features, these attacks are less useful in practice
    - Since the inputs to the ML model for network intrusion detection are PCAP files
    - Also, typically it is not known what type of features were used by the ML model
- ***Packet-level (end-to-end) attacks on ML-based NIDS***
  - Packet-level attacks generate full PCAP files, rather than network features
    - In the taxonomy by Rosenberg et al. (2021), these attacks are end-to-end attacks based on the attack's output
  - Such attacks are more practical, because the generated adversarial samples can be used to directly evade ML models for network intrusion detection
  - Limitation of current packet-level methods: most attacks focus on evaluating the ability to evade ML models used for network intrusion detection
    - Less attention is paid to evaluating the functionality of adversarial samples (i.e., whether a perturbed benign sample has preserved its functionality and its malicious behavior)

# Feature-level Adversarial Attacks on NIDS

*Feature-level Adversarial Attacks on ML-based NIDS*

- ***Warzinsky et al. (2018) Intrusion Detection Systems Vulnerability on Adversarial Examples*** (link)
  - White-box evasion attack against a three-layer MLP classifier using the NSL-KDD dataset
  - FGSM (Fast Gradient Sign Method) was used to create perturbed samples by modifying input features
    - The adversarial samples were misclassified as normal samples by the MLP model
  - The outputs of the attack are modified feature vectors
- ***Clements et al. (2019) Rallying Adversarial Techniques against Deep Learning for Network Security*** (link)
  - White-box evasion attack against Kitsune – a NIDS comprising an ensemble of autoencoders
    - An anomaly score is calculated based on a weighted RMSE deviation of the ensemble of autoencoders
  - The authors implemented 4 attacks: FGSM, JSMA (Jacobian-based Saliency Map Attack), Carlini & Wagner, and ENM (Elastic Net Method) attack
    - It has the same limitation, as only the feature vectors were perturbed

# Feature-level Adversarial Attacks on NIDS

*Feature-level Adversarial Attacks on ML-based NIDS*

- ***Huang et al. (2019) Adversarial Attacks on SDN-Based Deep Learning IDS System*** (link)
  - White-box evasion attack on port scanning NIDS classifiers in a software-defined network (SDN)
    - SDNs use software-based controllers to control network traffic (instead of using dedicated hardware-based devices, such as routers or switches)
  - Attacked are three NIDS deep learning models, employing LSTM, CNN, and MLP architectures
  - FGSM and JSMA attacks were performed on regular traffic packets to generate adversarial samples
  - Besides the evasion attack, this work also demonstrated an availability attack
    - JSMA was applied on regular traffic data packets, which were classified by the port scanning NIDS as attacks, resulting in blocked legitimate traffic

# GANs for Adversarial Attacks on NIDS

*Feature-level Adversarial Attacks on ML-based NIDS*

- ***Lin et al. (2018) Generative Adversarial Networks for Attack Generation against Intrusion Detection*** (link)
  - Against seven traditional ML-based NIDS: SVM, naïve Bayes, MLP, logistic regression, decision tree, random forest, and *k*-NN classifier
  - A GAN architecture called IDS-GAN (GAN attacks against Intrusion Detection Systems) is proposed
  - NSL-KDD dataset was used for training the classifiers, and for evaluating the adversarial samples (with perturbed feature vectors)
- ***Yang et al. (2018) Adversarial Examples Against the Deep Learning Based Network Intrusion Detection Systems*** (link)
  - Against a deep NN model using the same features from the NSL-KDD dataset as in Lin et al. (2018)
  - C&W, ZOO (Zeroth Order Optimization), and a GAN-based attack were used to add small perturbations to the input feature vectors, so as to deceive the deep NN model and misclassify malicious network packets as benign

# Packet-level Adversarial Attacks on NIDS

*Packet-level Adversarial Attacks on ML-based NIDS*

- ***Homoliak (2019) Improving Network Intrusion Detection Classifiers by Non-payload-based Exploit-independent Obfuscations: An adversarial approach*** (link)
  - Packet-level attacks against five traditional ML classifiers: naïve Bayes, decision trees, SVM, logistic regression, and naïve Bayes with kernel density estimation
  - Evaluated on a dataset collected by the authors called ASNM-NPBO
  - The attack approach involve applying random obfuscations and modifications to the network packets
    - Examples of modifications are: adding time delay to a packet, reordering a packet, damage parts of a packet, duplicate parts of a packet, and fragmenting a packet
    - The modified network packets behave similar to normal traffic, and can evade ML models used in NIDS
  - The attack generated network packets, and not just modified feature vectors

# Packet-level Adversarial Attacks on NIDS

*Packet-level Adversarial Attacks on ML-based NIDS*

- ***Kuppa et al. (2019) Black Box Attacks on Deep Anomaly Detectors*** (link)
  - Query-efficient gray-box (score-based) evasion attack
  - Attacks against seven anomaly detectors: autoencoder, One-Class SVM, autoencoder with Gaussian Mixture Model, anoGAN, deep SVM, isolation forests, and an adversarially learned model
  - The seven classifiers were trained on the CSE-CIC-IDS2018 dataset
  - The work employs a manifold approximation algorithm to project pcap files into a subspace where an adversarial sample is found that is the closest to the original clean file
    - Afterward, the adversarial sample is projected back into a pcap file

# Additional References

1. Rosenberg et al. (2021) – Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain (link)
2. Ahmad (2020) – Network Intrusion Detection System: A Systematic Study of Machine Learning and Deep Learning Approaches (link)
3. Cloudera Fast Forward – Deep Learning for Anomaly Detection (link)
4. Blog Post by Cuelogic Technologies – Evaluation of Machine Learning Algorithms for Intrusion Detection System (link)
5. Intrusion Detection – Chapter 22 in "Introduction to Computer Security"
6. Blog Post by Gerry Saporito – A Deeper Dive into the NSL-KDD Data Set (link)