CSCD58 Final Project Report

Packet Sniffer + Intrusion Detection Using Machine Learning

UTSC Fall 2023

Team Members

Name	Student Number	UTORID	Email
Vincent Li	1006036889	livinc10	vinc.li@mail.utoronto.ca
Jiayu Lu	1005706960	lujiayu4	jiay.lu@mail.utoronto.ca
Yusuf Khan	1006330329	khanyus7	yusuf.khan@mail.utoronto.ca

The video can be found with the source code, titled:
"CSCD58_Final_Project_Video"

Team Contributions

Vincent Li

- In charge of training the IDS
    - Used the kddcup99 dataset and scikit-learn to train the IDS model (**train_ids.py**)
    - Created a user friendly setup script (**setup.py**) to allow users to initiate the training process themselves
- Created the IDS object (**ids.py**) for the packet sniffer to use to classify TCP connections
- Integrated the packet sniffer with the IDS by creating objects **tcp_connection.py**, **trimmed_packet.py** to transfer data between the packet sniffer and the IDS

Jiayu Liu

- In charge of implementing Packet Sniffer
    - Used scapy library to captures packets selectively based on specific IP and ports
    - Employed multithreading to manage packet sniffing and data analysis simultaneously

Yusuf Khan

- In charge of implementing CLI
    - Used click and repl_click python libraries to create autocompletion and documentation for all commands
    - Used IDS and Packet Sniffer implemented by other members to simulate attacks
    - Created different attack types for simulation

About this Project

Description

This project is a Python packet sniffer with an integrated machine learning algorithm that uses scikit-learn and the kdcup99 dataset to train a machine learning algorithm in order to detect suspicious TCP connections. More specifically, the project is a CLI tool that users can run to watch incoming traffic to a server they are running on their computer, the tool will sniff out packets whose destination is the running server, and display their details.

Running parallel to the packet sniffer is the **Intrusion Detection System (IDS)** which will keep track of incoming TCP connections, and flag any it deems suspicious. Types of attacks that can be detected include but are not limited to denial of service (dos) and probing. Once a connection is flagged, a warning is shown on the CLI, warning the user of a potential attack on their server.

Goals

While building this project, we had the following goals in mind:

- Detect and display network traffic coming to/from a server on a computer
- Automatically detect suspicious connections
    - Figure out what kind of features of a TCP connection make a packet suspicious.
    - Figure out patterns for different types of attacks

The general goal of this project is to build a tool that can monitor traffic to and from a specific server running on a machine, and furthermore pick out any suspicious traffic automatically. The method we chose to pick out suspicious traffic is to train a machine learning algorithm trained off of a dataset of TCP connections. Since we chose to use a machine learning model, more goals related to construction the model arose:

- Figure out which model performs the best
- What kind of attacks is the model best suited for?
- What are the model's strengths and weaknesses?

Reaching these goals, and figuring out the answers will help us understand more about network attacks, and how we detect and defend against them.

Setup & Documentation

Installation

1. Clone the repo at: https://github.com/Jiayu-Lu/CSCD58_Final_Project
2. Make sure you have Python3
3. Install the requirements by using **pip install -r requirement.txt** at the root of the repo

Running the Application

In order to run the application, run the CLI user interface with: **python cli.py start**, refer to the **CLI** section below for a more detailed information on the different settings and features of the CLI.

Documentation

All the documentation can be found in markdown format and in HTML format in the **documentation** folder. We recommend looking at the HTML version by just opening **index.html** in your preferred browser.

Implementation Details

This project can be broken up into 3 main parts: the CLI, packet sniffer, and the IDS. On top of these main parts, we have also written simulations to simulate attacks on our test server. For this project we used Python3 along with scapy as our packet sniffing library. The model was trained using scikit-learn along with the kddcup99 dataset, many models were tested which will be explained in the later section.

CLI

The CLI brings everything together. We used the python libraries click and click-repl to build our CLI. It allows you to simulate different types of attacks on an http test server and then classify each connection to the server using an IDS.

Start the CLI with the command:
python cli.py start

You will be greeted with a welcome message. The CLI has a few commands which let you configure the simulation, you can type in --help to view all of them, but a summary can be also found here:

Command	Description
print-attack-types	Prints out the available attack types that can be simulated.
set-algo ALGORITHM	Set the algorithm/model to use for the IDS when classifying a connection. ALGORITHM is the number of the algorithm that is used for simulations Valid numbers are: 0: Gaussian Naive Bayes 1: Decision Tree Classifier 2: Random Forest Classifier 3: Support Vector Classifier 4: Logistic Regression 5: Gradient Descent
print-algo	Prints out the current detection algorithm used in the simulation.
quit	Exit the CLI
start-ids	Start the IDS (This starts an http server and captures packets it receives. It uses these packets to classify each connection using the IDS) In another terminal use the start-attack command to start an attack to this server
start-attack ATTACK_TYPE	Starts an attack of type ATTACK_TYPE on the server. Available attack types are: **probe**: A general probe attack **port_scan**: Uses nmap to check if server ports are open (probe) **ack_dos**: Sends thousands of TCP acks to dos server (dos) You can also add new attack types. Each attack type is just a function. Add a new function to the simulation/attack_types.py file and then add the function name to the array at the top called __all__.

To view the options for a specific command you can type in command-name --help, just like a regular linux command.

Packet Sniffer

The Packet Sniffer (PacketCapture class) is a part of an Intrusion Detection System (IDS) designed to monitor and analyse network traffic. This class leverages the scapy library for its core functionality of packet sniffing, selectively capturing packets based on defined IP addresses and ports. Once captured, these packets are queued for further processing and analysis.

Initialization and Configuration

The class is initialised with a set of configurable parameters that include a **detection timeout**, a **detection interval**, and an **Algorithm object** for intrusion detection. The **detection timeout** parameter sets the duration for the packet capture window, while the **detection interval** dictates the frequency at which the IDS's analytical model evaluates the network traffic. The class instantiation also involves the creation of an IDS instance, utilising an **Algorithms** object, which forms the backbone of the intrusion detection mechanism.

Concurrency Operations

To efficiently manage the dual tasks of packet sniffing and data analysis, the class employs a multithreading approach. This concurrency ensures that network packets are continuously captured and queued for processing without interruption, thereby maintaining a real-time monitoring environment.

Packet Processing Mechanics

The class manages incoming network packets by placing them in a queue and processing them at set intervals. It handles TCP connections by starting new ones when SYN packets are detected and closing them upon receiving FIN packets, thus keeping track of ongoing and past TCP connections.

Statistics Calculation

The class computes various statistics related to TCP connections and individual packets for IDS, which are crucial for analysing network behaviour and identifying potential anomalies.

IDS

The Intrusion Detection System (IDS) uses the kddcup99 dataset as its training dataset, and can be configured to use 1 of 6 different models:

- Decision Tree
- Logistic Regression
- Support Vector
- Random Forest
- Gaussian Naive Bayes
- Gradient Descent

All of these models are pre-trained and can be found in `attack_detection/trained_models`.
The dataset is a collection of TCP connections, with each row containing properties of the particular connection ranging from physical properties such as its size to domain knowledge such

as failed or successful login attempts. A full list of features and their descriptions can be found here, however they can be mainly broken down into 3 categories: physical properties (size of packets, duration of connection, etc), domain knowledge (login attempts, is root/guest user, etc), and statistics regarding connections within the past 2 seconds (number of connections within the last 2 seconds, number of connections from the same host, etc).

Training the IDS

The IDS was trained using the standard methods of training any statistical model, we first pruned features with high correlation, then split the dataset 70/30, where 70% of it was used for training. Using this split we trained each of the 6 different models, and determined their accuracy with the testing set. In our case, we only used 10% of the kddcup99 dataset to save on training time. (file: **attack_detection/data/kddcup.data_10_percent.gz**)

You can train the model yourself by running `setup.py` and going into `settings.py` to configure which models you want to train.

We have tested 6 different models for this IDS:

- Decision Tree
- Logistic Regression
- Support Vector
- Random Forest
- Gaussian Naive Bayes
- Gradient Descent

And judging purely by the test scores, the Random Forest model scored the highest accuracy.

Using the IDS

The IDS uses properties of TCP connections as its features, therefore in order to use the models we must collect data on each of the TCP connections that occur at our server.

- We create a new connection whenever we see a SYN packet
- We close the connection if we see a FIN packet
- For every packet that belongs to a particular connection, add it to the connection data, updating the feature values of the connection as packets are added (ex. As a packet is added, add to the number of bytes sent).
- Every x milliseconds, run the open connection on the IDS, and check whether the connection looks like an attack or not.
- If the detection is an attack, determine what kind of attack it is.

This process is automatically performed by the packet sniffer, and the results are shown on the CLI while the application is running.

Discussion/Analysis

IDS Accuracy and Choosing an Effective Model

The accuracy scores of our IDS were very high when used against the testing dataset during training, values were in the range of >95% accuracy. This is to be expected considering our relatively smaller dataset, and an even smaller testing dataset. Because of this, the different algorithms only differed by a couple percent at best, which made choosing the most effective algorithm not a trivial task, which is why we opted to make algorithm choice a customizable feature. For future improvements, the model should be trained on a larger dataset, potentially with a broader set of algorithms, in order to clearly see the differences between different algorithms.

Model Limitations

Integrating the model with our packet sniffer, and simulations proved tricky, due to many factors. These problems led to our model not performing as well as expected, in the following sections we elaborate on these factors and suggest improvements.

Dataset Quality

The kddcup99 dataset is a widely known dataset in the field of network attacks, but is often criticised for its quality. Online forums criticise the dataset for not reflecting real world attacks in 1998, let alone the modern age. This paper published by researchers at TMU also points out flaws in the method the data was gathered, and the quality of the data itself, including a major bias towards DoS attacks where 71% of the data is classified as a DoS attack. Therefore although the kddcup99 dataset is a well known dataset, it is not necessarily of high quality and should not be used when constructing production level detection systems. To improve on this flaw, it would make sense to use a more up to date and higher quality dataset such as the UNSW-NB15 dataset or the NSL-KDD dataset, which are more up to date, and reflect modern attacks more accurately.

Simulation Construction and Accuracy

Another limitation in the model is not directly about the model itself, but is in our ability to recreate scenarios that led to the data points in the dataset. Due to the smaller scale and timeframe of our project, it is not realistic to create a full fledged attack simulation environment where we can perform real-world attacks on our IDS. (Even if we did, the above section highlights why the IDS may not perform well even in that scenario) Therefore we opted to create a simple simulation, by modelling domain knowledge with keywords in packet payloads, and attacking a simple web server. However, without any knowledge on what kind of scenarios were captured in the kddcup99 dataset, it is impossible to create an accurate simulation where the IDS is in a familiar environment and can perform optimally. Therefore, while we tried to develop our simple simulation in a way that allows the IDS to be used in its fullest capacity, it definitely is not reflective of attack scenarios of the 1990s. One method to improve this is tied with the section, update the dataset to one of a high quality which in detail explains its data collection methodology. This way, creating a simulation to demonstrate the IDS' full performance would be an achievable task, in addition this simulation would be more reflective of the real world.

Conclusion

In conclusion, this project represents a significant advancement in integrating traditional network monitoring with machine learning techniques to enhance cybersecurity measures. The high accuracy scores of our Intrusion Detection System (IDS), exceeding 95%, showcase the potential of applying advanced algorithms in network security, even though our choice of algorithms was limited by the size and scope of the datasets used. However, the integration of the model with practical network environments highlighted several limitations, notably in simulation accuracy and dataset quality.

The reliance on outdated datasets like kddcup99, criticised for its lack of modern relevance and data bias, underlines the need for more contemporary and balanced datasets, such as UNSW-NB15 or NSL-KDD, to ensure the IDS is effective against current threats. Furthermore, the project's inability to recreate complex real-world attack scenarios limits the practical testing of the IDS, suggesting a need for improved simulation methods that better reflect current network environments.

Despite these challenges, the project lays a strong foundation for future improvements in network security. Emphasising the need for larger, more relevant datasets and more realistic simulation environments, this project underscores the potential of machine learning in revolutionising network security, setting a pathway for developing more dynamic, adaptive, and effective security systems.