AI-Driven Threat Detection and Response System

Alex Lappin
George Mason University
Fairfax, VA
alappin@gmu.edu

Joshua Puyat
George Mason University
Fairfax, VA
jpuyat@gmu.edu

Sumayah Alomari
George Mason University
Fairfax, VA
salomar3@gmu.edu

Mahi Khan
George Mason University
Fairfax, VA
mkhan209@gmu.edu

Hussam Al Bakhat
George Mason University
Fairfax, VA
halbakha@gmu.edu

Bibhu Paudyal
George Mason University
Fairfax, VA
bpaudya@gmu.edu

Zhengdao Wang
George Mason University
Fairfax, VA
zwang52@gmu.edu

Alexandre Barreto
George Mason University
Fairfax, VA
adebarro@gmu.edu

***Abstract*— The increasing sophistication and diversity of cyber threats pose significant challenges to organizations relying on traditional, signature-based intrusion detection systems (IDS). Modern cybersecurity demands advanced, AI-driven threat detection and response. This paper proposes a novel approach utilizing a Deep Neural Network (DNN) for dynamic threat classification, addressing the limitations of static, rule-based systems in the face of adaptive, AI-powered cybercriminals. Our proof-of-concept integrates a multi-class supervised DNN with the open-source IDS Suricata within an AWS environment. By training the DNN on a diverse dataset of MITRE ATT&CK examples (specifically, the CICIDS2017 dataset), we demonstrate its ability to classify and label Suricata alerts and Metadata, effectively bridging the gap between signature-based detection and dynamic threat adaptation. Suricata replays and generates real-world attack scenarios, providing realistic data for training and evaluating the DNN. We detail the preprocessing pipeline for Suricata data and the architecture of our DNN, highlighting the statistically significant improvements achieved in training and testing accuracy. This proof-of-concept establishes a foundation for a network of enhanced AI-driven security tools, paving the way for future research exploring specific AI algorithms, algorithm-to-tool mappings, and the development of proactive IDS capabilities. The paper concludes by discussing the potential for automated deployment and scaling of this approach within organizational settings.**

***Index Terms*—Cybersecurity, Deep Neural Network (DNN), Intrusion Detection System (IDS), Suricata, Amazon Web Service (AWS), Deep Learning.**

I. INTRODUCTION

In today's hyper-connected world, cyber-attacks have not only become more frequent, but they have also become more complex, posing major threats to critical systems and sensitive data. Cybersecurity has long depended on traditional detection methods such as signature-based and rule-based systems to identify malicious activities. With the rise of artificial intelligence (AI), there is now an opportunity to build more adaptive and intelligent security solutions that can detect and respond to both known and unknown threats in real time.

A. *Problem Statement*

Traditional intrusion detection systems (IDS) primarily use rule-based or signature-based techniques to detect malicious activities. While effective against known threats, these methods can fall short when confronting threats such as zero-day attacks or advanced persistent threats (APTs). For example, they could generate high false-positive rates, making them inefficient and unsustainable in dynamic environments. Moreover, many current intrusion detection systems (IDS) lack the ability to learn from new data or adapt to changing attack patterns, leading to decreased detection accuracy over time. This creates a critical need for more intelligent and adaptive detection systems.

This project addresses these limitations by proposing an AI-Driven Threat Detection System that uses machine learning algorithm dense neural networks (DNNs) to classify network traffic as either benign or malicious. The system is designed to integrate seamlessly with existing cybersecurity infrastructure, operate with minimal latency, and continuously improve its detection capabilities through ongoing learning and exposure to new data.

B. *Overview of the Approach*

The proposed AI-Driven Threat Detection System uses a supervised machine learning approach centered on (DNNs) to classify network traffic as benign or malicious. The pipeline consists of five main stages: data collection, preprocessing, model training, evaluation, and deployment. The system is trained on labeled datasets such as CICIDS2017 or NSL-KDD, ensuring exposure to diverse attack patterns. Raw traffic data is transformed through feature extraction and normalization into a format suitable for deep learning. The DNN model learns to identify complex patterns across multiple layers, enabling it to detect subtle indicators of compromise.

Model performance is evaluated using standard metrics such as precision, recall, F1-score, and Area Under the Curve (AUC), ensuring high detection accuracy and low false-positive rates. Once trained, the model is integrated into a real-time detection engine that analyzes live network data and flags threats automatically. The system also supports periodic retraining to adapt to evolving threats, making it a scalable and future-ready solution for dynamic cybersecurity environments.

II. BACKGROUND

The work on this project is a novel approach, which is, however, extremely expansive and requires a basic understanding of the components that all make up the workflow within the main use case. These fall under two primary categories: the AI itself and the AWS server which aims to bridge the AI and the IDS.

A. Amazon Web Services (AWS)

The AWS platform serves as the foundational infrastructure for this project, allowing for the development of the AI-Driven Detection and Response System within a secure cloud environment. AWS has several key features and tools which allow for remote access, resource organization, and scalable deployment. One major reason why AWS is optimal for the IDS portion of this plan is because simulating malicious network attacks call for an isolated environment. A fully contained and controlled sandbox is necessary so that there is no chance of high-risk activities escaping the targeted network and impacting the host machine or associated networks. Traditional virtual machines (VMs) are often hosted on local hardware and lack network isolation and other innate safety features that AWS provides like Virtual Private Clouds (VPCs) and easily managed security groups. This cloud-based architecture provides the comfort needed to conduct malicious cybersecurity testing. Specifically, Elastic Compute Cloud (EC2) is the VPC instance where the relevant tools were deployed. These instances are highly customizable, allowing users to create a virtual machine with select memory, CPU, storage, operating system, security settings, network configurations, and more. The flexibility does not end with the creation of one instance, however, these instances can be copied and launched in mass. Meaning that projects that utilize AWS EC2 have the benefit of effortless yet legitimate scalability. Another integral service is Simple Storage Service or S3. S3 buckets serve as repositories for storing data and offer a simple way to export essential files like packet capture (PCAP) logs. Essentially, AWS ensures that any system that utilizes their ecosystem is given a unified, secure environment.

- ***Graphical Network Simulator-3 (GNS3) Server:*** GNS3 is a powerful tool for creating and monitoring detailed network topologies in real-time. This method of emulation is useful for simulating real environments with virtual routers, switches, endpoints, and firewalls. Each allows for a degree of flexibility when in the building process, resulting in complex systems that mirror physical or cloud environments. The software allows the user to control the flow of network traffic and other data throughout different nodes of the system. Cyberattacks sent to an GNS3 instance would propagate through these different nodes and entry points, giving a more three-dimensional understanding of the way certain attacks impact certain network topologies. When the goal is to gather training data for a machine learning model, this contextual behavior would be a more perceptive input.

- ***Suricata:*** Created by the Open Information Security Foundation (OISF), Suricata is an open-source IDS/IPS that has strong packet inspection and real-time alerting capabilities. This tool supports a wide range of protocols, multi-thread activities, and has a plethora of uses in both online and offline modes. One of the core strengths is its logging functionalities. The primary output log goes into the `eve.json` file where alert signatures and other select traffic flow information are written. Other available logging formats include `fast.log, unified2, syslog`, and PCAP. Each logging method can be further configured in the `suricata.yaml` file where settings critical for simulation and testing can be found. One such setting is conditional PCAP logging. This option transitions the IDS to only log packets that match specific conditions or rules. Conditions can be set using keywords or through events. Namely when certain alerts are triggered. Only logs related to the set signatures or rules are written and there is no noise from unrelated background traffic. A feature essential for fast and focused analysis, as sifting through massive PCAP files may be unnecessary for simulation research purposes. Additionally, conditional PCAP logging is disk and memory efficient for environments with limited capabilities like cloud instances or virtual machines.

Offline mode is fundamental for testing complex network traffic flows on the system. In this, Suricata analyzes pre-captured PCAP files. A feature mainly used for replaying attack scenarios by TCPreplay or other tools that can feed historical traffic data for Suricata to read as if it were happening live [1]. Malicious attack replay can be used when there is a lack of attacking tools or technologies available to the user. A method that poses no risk to the host machine and does not require network access. The rule engine in Suricata is compatible with other common detection and prevention systems, allowing for an overlap of resources and rule customizing. Suricata remains an excellent option for any level of use from simple research to enterprise-level security.

- ***Snort:*** Snort is an IDS that specializes in live packet analysis and attack signature detection. This tool was released 12 years before Suricata and was one of the original open-source detection systems. Snort has been used widely in enterprise environments and is currently maintained and developed by Cisco. For capabilities, Snort falls short to Suricata on a few metrics. Snort has basic protocol parsing, dated logging, moderate performance, and only recently got multi-threading with Snort 3. Rulesets in snort have been industry defining with a simple syntax that is widely used and understood. It operates by inspecting packets in network traffic and matching it against a set of detection rules set by the user. These rules target certain information like header values, protocols, and payload content. Multiple operational modes offer the user a flexible detection tool that can fit into various use cases. Sniffer mode displays packets in real time. Packet logger mode is used for offline analysis as Snort captures and stores raw traffic into a disk. Network Intrusion Detection Mode (NIDS) is when its signature-based detection engine produces alerts based on matches with predetermined rules.

- ***CICFlowmeter:*** The Canadian Institute for Cybersecurity (CIC) designed this traffic flow analysis tool with the purpose of converting raw PCAP data into readable, structured formats like `.csv` files. Once transformed, the data easily fits into statistical analysis processes and machine learning. With a more advanced method than basic packet logging tools, CICFlowMeter relates sequences of captured packets into bidirectional flows that share a few common properties. Eighty-four features, or categories, are extracted for each flow. Features include duration, protocol, packet counts, time-based statistics, flow-specific size/rate metrics, and flow behavior.

- ***MITRE Caldera:*** IDS testing requires one of two things: an attacking machine where malicious traffic is being sent to the target, or a tool that carries out attack simulations on the user's behalf. Caldera is an attack emulation platform that simulates real cyberattacks using the MITRE ATT&CK framework. Functionalities include automated pinpoint attacks, scheduling, scripting, and integration with external tools. The primary use case is to generate consistent, known attack traffic in isolated environments. Caldera and other similar tools are integral for training and evaluating higher level machine learning models working on intrusion detection systems.

B. Involved AI Models

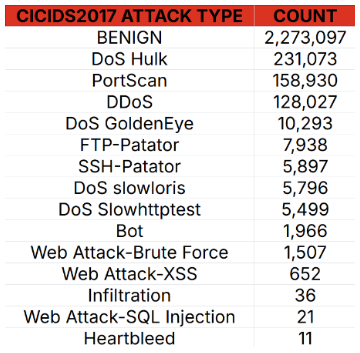
In today's world the number of options regarding machine learning or AI include a wide variety of variations and special use cases. The effectiveness of a statistic model relies primarily on the quality of the dataset it is training from, and relevance of the algorithm that is analyzing the data [2]. This makes knowing the background of all competing models crucial in making a model with significant results. The IDS models in any environment will yield high volumes of network flow statistics, creating high dimensional feature sets. When examining large feature spaces there are a multitude of features that a model may learn from, especially when looking at PCAP network flow. A traditional machine learning model (e.g., SVM, Decision Trees) will require domain knowledge on select features that need to be represented as priority. This means that a good idea of a feature hierarchy would be necessary when creating the model, however this paper aims to try to understand this since PCAP network flows are not as novel. Additionally, in the case of multi-class data, non-linear class boundaries will be exhibited, and a traditional machine learning approach would be unable to scale in comparison to anything aside from a binary classification. When analyzing large datasets, speed and time are an invaluable resource, making the need for GPU utilization and parallel processing a necessity for completion of this task. These factors along with the idea of scalability in an adaptive learning environment such as cyber security, where attacks will never stick to a single pattern, create the need for advanced AI in our system as opposed to traditional machine learning.

- ***Convolutional Neural Network:*** A Convolutional Neural Network (CNN) is characterized by convolutional layers, which apply filters that learn local patterns [3]. Given that PCAP files will always be in chronological order, and attacks will usually occur over multiple packets being sent back and forth, the idea of local pattern analysis is a very plausible idea in theory. This is why, originally, it was our first model in development. CNNs were initially brought about for analysis of image processing, since these were 2D relationships that were pioneered on local features. CNNs, however, can be adapted for 1D text processing as well. By setting a numeric base for filter range, the formula can look at similar feature columns inside of a CSV and take local relationships in that number into account. This makes it necessary to have similar columns in the data, or features, close to one another since they will have a relationship. After preprocessing the data, the formula looks at $X \in \mathbb{R}(\mathrm{totalPackets}, 1, N)$ where N is the number of features, in our case 79. These [1, N] vectors are the rows since this is in 1D mode. After each convolution, ReLU (Rectified Linear Unit) is applied elementwise, introducing non-linearity in order to capture the complex pattern relationships. To combat overfitting of data, which means the data will train only to the specific dataset it has been given and unable to expand to new datasets, there is a max pooling layer afterwards to reduce feature maps and introduce a slight amount of variance [3]. At the very end of these layers there is a SoftMax flattening layer which turns the data into a single vector in order to classify the data.

- ***Deep Neural Network:*** A Deep Neural Network (DNN) or multi-layer perceptrons (MLP) is distinguished by fully connected dense layers. These connect neurons in one layer to the next neuron in another layer. This design allows the network to directly learn non-linear feature interactions across the entire feature space, without relying on local neighborhood assumptions in the case of CNNs. Each input, no matter the location of column, is equally accessible to every neuron of the previous layer. Each row of the dataset becomes represented accurately as X∈R(totalPackets,N) where N is the total number of features in the dataset, in our case 79. Each of these rows enters an input layer and from there, dense layers will apply a learned weight and bias term, which will be talked about more in depth within our exact application. This then passes through an activation function, ReLU (Rectified Linear Unit), which helps the model learn complex high-dimensional decision boundaries through the introduction of non-linearity. To reduce the overfitting of the training dataset, the introduction of dropout and early stopping are usually put into place. Dropout takes a fraction of neurons and removes them to then force the model to learn redundant representations of the data and not rely on any single neuron [3]. Early stopping also stops training once the improvement comes to a logistic slowdown and prevents any additional epochs, or runs, from overtraining the data on the exact features of the dataset. Once these hidden layers have finished, there is a SoftMax layer which determines the probability distribution of the classes similar to CNNs.

C. CICIDS2017 Dataset

The CICIDS2017 dataset was developed by the Canadian Institute for Cybersecurity (CIC), and it known as a benchmark for modern intrusion detection simulation [4]. This dataset emulates a real network environment by incorporating various realistic attacks alongside legitimate traffic flows. It focuses on 14 main attacks along with a large benign dataset to recreate actual traffic flow.



CICIDS2017 ATTACK TYPE	COUNT
BENIGN	2,273,097
DoS Hulk	231,073
PortScan	158,930
DDoS	128,027
DoS GoldenEye	10,293
FTP-Patator	7,938
SSH-Patator	5,897
DoS slowloris	5,796
DoS Slowhttptest	5,499
Bot	1,966
Web Attack-Brute Force	1,507
Web Attack-XSS	652
Infiltration	36
Web Attack-SQL Injection	21
Heartbleed	11

TABLE 1: CICIDS2017 Attacks and Counts

The table above provides the exact attack names and variations of each attack involved in the CICIDS2017 dataset as well as the corresponding attack numbers of each attack packet which show the dataset weight balancing. These represent real world threats that cyber-related fields face every day. It also represents day-to-day traffic that is captured in the benign category. These include browsing, working, emailing, and file transfers that would present itself in any real-world IDS. Tools such as tcpdump captured these raw packets and then processed them into flow statistics with 85 features including IP addresses and exact timestamps. They also focused on modern systems as a big motive for CICIDS2017 and subsequent datasets that CIC released is the lack of large modern network flows that would truly represent up-to-date systems [4]. One big takeaway from this dataset is the high imbalance towards certain attack classes such as DoS being a couple hundred thousand attack packets whereas an SQL injection only contains 21 total packets. These imbalances represent the likelihood of these attacks in the real world, however, will cause statistical models to ignore them due to the lack of representation. Additionally, this data is not automatically readable by an AI, so there must be development of a tailored preprocessing code that is detailed later in the implementation.

D. Multi-Class Supervised Learning

As seen in the CICIDS2017 dataset, we are working with 15 total classes. Unlike binary classification, the model will output one of several possible labels for each input instance. In the context of our IDS, this finer-grained approach enables a single model to detect and classify multiple attack types. Since the classes are imbalanced, weighing these different classes accordingly is a necessary hyper-tuning that will be required for them to represent the AI output. This presents itself with the cross-entropy loss function which is a standard for multi-class neural networks and compares predicted probability of one-hot or label-encoded targets. We also need to adjust the layers of the neural network to match numerically with the 15 classes, hence our output layer size being 15. With the case of multiple classes, it is extremely important to look at each individual class metrics when weighing performance in the form of precision and recall.

Our models also utilize supervised learning since the CICIDS2017 dataset includes multiple explicit labeled attack classes rather than a simple yes/no attack present [4]. We can calculate model loss on how closely the model is able to match the initial labels when checking itself. In contrast with unsupervised, anomaly-based attack detection, the model will define its own notion of an attack without any direct guidance leading to errors and incorrect anomalies especially in a large dataset with benign traffic. This supervision will produce actionable alerts based on prior datasets and even allows a self-propagating model wherein the model is retrained on its own correct classifications.

E. Weakness of CNN Algorithm

When looking for a model to begin research our team saw potential in the more nuanced relationships that a CNN may uncover as well as similar examples of a CNN being used when looking at PCAP data [5]. This led us to create a fully developed CNN. The model has three convolution layers which include a ReLU in every layer to prevent overfitting. This is followed by a flattening layer and a cross-entropy loss layer which is meant to combat the imbalanced CICIDS2017 dataset by weighing underrepresented classes with a higher weight. The competing DNN at the time had three dense layers followed by a flattening layer and cross-entropy loss layer at the end. Our team spent months hyper-tuning CNN only for the best results to be underwhelming.

Model accuracy is the most common metric when looking at AI's capabilities.

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Using equation (1), TP represents the correct positive results, TN represents the true negative results, FP denotes false positive, and FN denotes false negative [5]. The highest accuracy after testing many different variations was 0.85. This on its own, however, does not show the whole story as a model with 90% of its data being benign would then be 90% accurate if it never recognized a single attack.

$$PR = \frac{TP}{TP+FP} \quad (2)$$

$$DR = \frac{TP}{TP+FN} \quad (3)$$

Equation (2) denotes precision and equation (3) denotes detection rate where the variables are recognized from equation (1) [5]. Precision shows how many of the predicted positive cases were actually correct. Recall shows how well the model captures actual positive cases. Our highest precision in the CNN with equal weighting in each class was a macro average of 0.49 among the 15 classes along with a detection rate of 0.88. This was due to extremely low precisions such as 0.01 regarding SQL injection and 0.04 regarding infiltration. These low count classes, among others, were essentially being ignored by CNN. No matter how much weight we would give the low represented class in the cross-entropy function, the low classes would only peak at around 0.3 precision, which would tank the rest of the precision in return.

Knowing these statistics, the model itself indicates that CNN is biased towards majority classes and underperforms when the attack has fewer samples. Our overall accuracy was inflated by the majority classes and when looking at equal averages, this model does not meet our statistical needs. This suggests that the local relationships that initially attracted our team to CNNs may not exist in PCAP data which is notoriously tabular. Thus, the convolution filters are not finding patterns to distinguish minority classes even with bias and weight manually applied to them.

F. Appeal of DNN Algorithm

The absence of spatial data within the CICIDS2017 as well as k-column window sharing within the CNN, which weights columns near each other similarly led to an overcomplication of the algorithm and subpar results. The DNN's fully connected layers treat every feature as first-class and learn global, non-local interactions by default. The DNN also has heavier minority class amplification since rare IDS flows are not combined within a convolution but instead contribute gradient weight to all first layer weights. This is necessary when seeing the imbalance of CICIDS2017. Additionally, max pooling does not exist within a DNN and can save the loss of information gained when invariance is introduced into the system.

This is the point where the team decided to continue development within the form of a DNN which is heavily expanded upon in the proof-of-concept and further proven efficiency when SHAP is introduced.

G. Explainability (XAI)

U.S. Executive Orders & NIST AI Risk Management Framework strongly enforce explainability within AI. This is defined as the ability to effectively understand why the AI is making the decisions that it is [6]. Our model aims to provide this through showing our best model after training and having SHapley Additive exPlanations (SHAP) explain which features it is relying on as well as the impact they made on the outcome with inclusive statistics such as false positives and false negatives which both impact a real-world deployment of the AI-driven IDS system. Any operational version of this model will provide administrative logging to explain the reasoning behind any incorrect assumptions.

III. PROOF-OF-CONCEPT

This section is dedicated to the conceptual system that our team originally came up with as a remediation of the solution. We propose the idea of having a fully self-propagating AI system linked to a red team setup within the IDS which simulates attacks consistently and feeds them to the AI on to keep updating it with new threats. It is worth noting that some sections are intentionally vague as they will be explained further in the methodology of implementation.

A. *Environment Setup*

The environment for this project was a t2.large EC2 instance made within AWS. This virtual cloud interface provides 2vCPUs and 8 GB of memory, meeting the requirements for running

Suricata with passive PCAP logging or initiating large scale PCAP replays. Ubuntu Linux is used as the operating system due to its high compatibility with most security tools and low resource overhead. This isolated VPC ensures that any simulated malicious traffic won't leave the contained environment. To ensure fill connectivity and support inbound and outbound traffic flows, a public IPv4 address was enabled. Additionally, security group settings associated with the instance were modified to allow inbound HTTP (port 80), HTTPS (port 443), and ICMP traffic. Essential configurations for testing commands like `curl, wget, and ping`. Outbound rules were set to allow for unrestricted internet access.

B. *Attack Simulation*

Our traffic simulation component utilizes a scheduled attack model, allowing for both benign and malicious traffic within select time windows. This scheduling allowed for post-capture labeling of flows with a Python script ran through the IDE Spyder. This method resembles the semi-automated approach used in the original CICIDS2017 dataset [7]. Benign traffic was generated with Linux tools and common network utilities to emulate realistic user behavior. ICMP pings, DNS queries, HTTP/HTTPS requests, file downloads, and NTP synchronization are some of the benign included:

```
ping –c [#repetitions] example.com

nslookup example.com

Curl http://example.com

ntpdate time.google.com

wget [download site] -O [file path]
```

These commands are simple methods to generate a diverse array of traffic within the scope of a singular terminal interface operating in a one machine simulation system. A VM may be used for adversary emulation but would add chronological deviation, which is an unwelcome factor if the limited flexibility of a schedule-based labeling system is considered. This project adopted the 15 label outputs that were used in the CICIDS2017 dataset but only some could be simulated due to few factors. Web attacks, for example, would lack a compatible web server on the receiving end, making it impossible for a realistic simulation of HTTP related vulnerabilities. Additionally, more complex attacks that rely on lateral movement or have a multi-stage process would require a much more complex network topology. This is one reason why incorporating a tool like GNS3 that allows for multiple nodes could be significant for future iterations of the project. More realism and richness in the input data for a ML algorithm leads to a stronger model. For malicious traffic generation tools like Nmap, HULK, hping3, and scripted HTTP payloads were used. Each attack type had clearly defined time blocks for when they were to be executed. Of the 14 possible attack patterns our ML model could decipher, we simulated Port Scanning, DoS Hulk, DoS GoldenEye, DDoS, and Cross-Site Scripting.

C. *Deep Neural Network*

As mentioned before our team found the DNN to be the best conceptual match to the tabular data found within PCAP format [4]. We proposed a complex DNN model with many components in order to utilize the power of our model to the maximum potential. This came in the form of PyTorch after discussion with industry professionals about the latest neural network technology.

Beginning with the data preprocessing we first put seeding into our model due to the research nature of the project, we wanted to have reproducible results in the form of `torch.manual_seed=42`. The dataset is read into a Pandas Data frame, then shuffled and scaled in order to stabilize our gradient-based training. The training validation split to support this architecture was 90%–10% split with stratification [5].

Our model architecture is the most important part of the system. Lightning is a PyTorch library that abstracts boilerplate aspects of AI design into high level classes leading to concise, maintainable code [8]. Additionally, Lightning allows hyperparametric management which helps checkpoint good model results and experiment with multi-GPU training. Our DNN was defined in a DenseClassifier with four fully connected layers (78 → 128 → 64 → 32 → 15) for each of the 15 attack classes with a ReLU layer in between for non-linear, complex, pattern recognition. To address the class imbalance of the CICIDS2017 dataset, we proposed the use of FocalLoss which is a step above standard CrossEntropyLoss in the CNN. This formula represents equation (4).

$$FocalLoss = -\alpha(1-p_t)^{\gamma}\log(p_t) \qquad (4)$$

Equation (4) will focus on penalizing misclassification of underrepresented samples more heavily with p_t being the probability of a ground-truth (correct) class. Additionally, the sample's selection probability is scaled inversely to the class frequency raised to a power of 0.4 to ensure higher minority class appearance in the batches.

Regarding the training and evaluation of our model, a forward pass takes our 79 features and passes them through the network with 15 output logics. The loss computation is then found through the focal loss of the ground-truth labels. Finally, there is a backward pass which updates the network weights and allows the model to more accurately look at minority classes for the next forward pass. After each epoch (run) of the model, the best model path is updated if the current validation accuracy is surpassed in that run and then saved as a checkpoint. After x number of epochs are run, the evaluation function loads the best model checkpoint and shows the final statistics. This proposition of DNN logic is implemented in section IV and will cover the statistics gained.

D. *High Level Architecture*

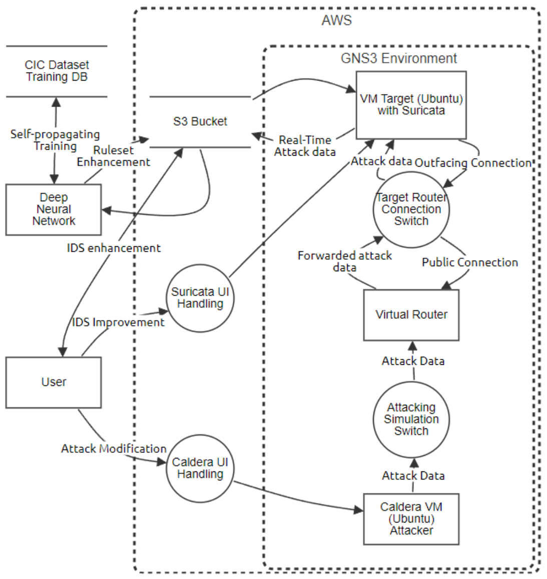


Fig. 1: Proof-of-concept Data Flow Diagram

Figure 1 depicts our integrated concept environment where real-time traffic from a virtual simulation is routed through a GNS3-based network with a router and two switches, to a target host running Suricata. Simultaneously, the Deep Neural Network interacts with Suricata's rulesets and detection data to further improve overall detection accuracy and responsiveness. The user in this scenario will be in the form of a system administrator or similar role. They will be outside of the system, configuring the Caldera attack simulations as well as the updating of the Suricata rulesets and configuration in order to better defend and detect attacks. They can additionally view the S3 bucket for the AI output for XAI purposes. Finally, the DNN will be taking IDS attack data in this instance and adding it to its database for retraining as well as evaluation, so the network will grow as more Caldera attacks occur.

IV. DESIGN IMPLEMENTATION

The following section is designed in detail the real-world partial implementation of the proof-of-concept in as much as our team was able to complete given the time frame. This does not, however, include the caldera setup or GNS3 server since the red team environment became unfeasible as we continued development on the IDS and AI. Instead, our team has implemented the use case of attack replays within the IDS to mimic the red team environment to the best of our ability.

A. *Suricata IDS*

Suricata is the central IDS system used for this project and the tool is improved by our AI model. This tool was responsible for monitoring traffic, triggering alerts, and logging packets for further analysis [9]. The `suricata.yaml` file is where configurations like passive packet capture mode are enabled. Captures were written to `/var/log/suricata/`, with the filenames `team1.pcap.[timestamp]`. To enhance detection abilities, Emerging Threats (ET) Open Ruleset was enabled which gave the modern rules for a wide range of known attack signatures. This essentially allowed Suricata to detect common malicious traffic behaviors without the need for custom rule iteration. We experimented with changing rule signature names to match the CICIDS2017 labels, but this process helped log readability more than classification. Another challenge was to allow Suricata to monitor traffic created from the host source. By default, Suricata does not see loopback interface traffic, so this configuration was necessary for the purpose of this scheduled attack system. The configuration to allow "lo" and "any" traffic in testing scenarios allowed for the generation of detailed alert logs in the `eve.json` file and raw PCAP captures that aligned with our scheduled traffic testing.

B. *AWS S3 Bucket*

Amazon Simple Storage Service (Amazon S3) is a highly scalable, durable, and secure object storage solution offered by Amazon Web Services (AWS). In the implementation of our Intrusion Detection System (IDS), Amazon S3 was leveraged as a centralized medium to facilitate data exchange between the cloud-hosted IDS and the local development environment. The IDS was deployed within an AWS Elastic Compute Cloud (EC2) instance, necessitating a method for transferring packet capture (pcap) files to and from the virtual environment. To achieve this, an S3 bucket was configured to act as a temporary storage container for relevant network data. Each object within the S3 bucket encapsulates the pcap data, corresponding metadata, and a globally unique key identifier.

To upload pcap files from the IDS to the S3 bucket, the AWS Command Line Interface (CLI) was utilized. The following command was executed within the EC2 terminal: `aws s3 cp`. This operation ensured that the files were securely transferred to the bucket, making them accessible to us. Once stored in the S3 bucket, these pcap files were downloaded onto a local machine for offline inspection and modification aimed at refining the IDS detection algorithms. Conversely, when additional data or updated files needed to be introduced into the IDS environment, they were first uploaded to the same S3 bucket from the local system. The EC2 instance then accessed the contents of the bucket using `aws s3 ls`. This allowed enumeration of all stored objects. The required file was extracted into the virtual machine using `aws s3 cp`.

Through this process, Amazon S3 effectively enabled secure, reliable, and scalable data synchronization between cloud and local environments, significantly contributing to the efficient development and maintenance of the IDS framework.

C. *Supervised Training and Results*

As previously discussed, our team implemented a fully connected Deep Neural Network which is highly detailed in Section III subsection C. This model can analyze the CICIDS2017 dataset and using PyTorch Lightning framework, implements four dense layers (nn.Linear) and four rectified linear unit layers (nn.ReLU). Additionally, we implemented minority classification representation in the form of FocalLoss and a scaled selection probability as mentioned before.

With the model architecture in mind, our team found the Google Collab environment to be the best place to implement this code and work on it due to the online GPU and simultaneous change functionality. Additionally, our team had to develop a preprocessing code that would turn the CICIDS2017 dataset into readable training data. This was done through online literature and code that similarly used the CICIDS2017 dataset. We began by mounting the google drive to the collaborative environment to keep a database available for the preprocessed data. Next, we had to change the Latin-1 encoding of the "Thursday-WorkingHours-Morning-WebAttacks.pcap" file to UTF-8 using a script. We then remove all the NaN columns in the data since they contain no advantages as well as renaming the mis-encoded and unrecognized characters in the ASCII file labels. These CICIDS files are then concatenated into one singular large data frame called "MachineLearningCVE.csv". For the final preprocessing, we fit a LabelEncoder function onto the attack label column so that the included attack classes are saved as integers 0 through 14. We also replace any infinity or negative with a NaN and then convert all NaN values to the mean of that column to ensure consistent numeric data within each label category. Then, split the chunk's data into train (90%) and test (10%) sets, stratified by label. All these steps are completely necessary to avoid any errors or data miscalculation within the DNN architecture.

Once all of the preprocessing steps have been completed, we employ the environment's L4 GPU Tensor Core for AI, featuring 24 GB of VRAM. This allows us to begin parsing the model architecture which includes the previous proof-of-concept structure in the form of 125 epoch runs as well as graphs and SHAP for XAI.

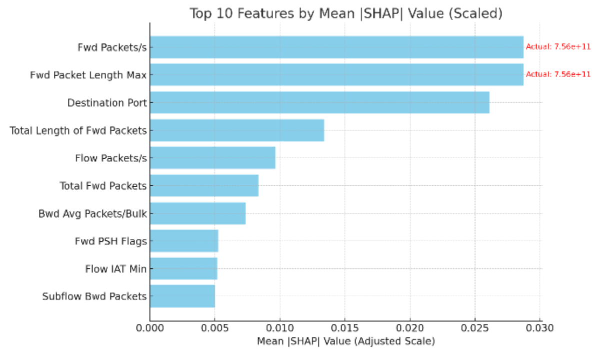
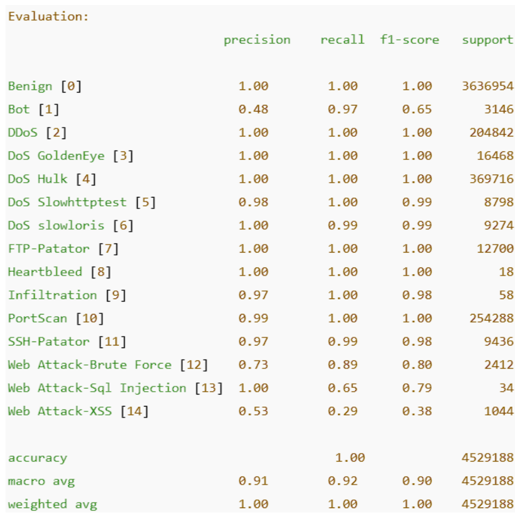


Fig. 2: SHAP Feature Values

Regarding the policy behind Explainable AI, our model proactively implements SHAP to identify the heaviest feature contributions to the results of the model. It does this through an advanced Shapley formula which measures how the inclusion or exclusion of a feature affects prediction. Figure 2 shows these features within our PCAP file and ranks the top 10 notable features. Two features that were exponentially useful to the model and took up most of the weight as seen by how they were so large they did not fit in the graph were the Fwd Packets/s and Fwd Packet Length Max. This means that many of the predictions came solely from the intensity (rate) and magnitude (size) of packets flowing in the network. This makes sense as things like DoS or DDoS are characterized through the packet timing as well as port scan and infiltration through the packet size. This was followed by metrics such as the destination port and the total length which is the sum of the previous packets.

The additional explainability helps any operator attempting to use this XAI in a real-world scenario to understand where certain evaluations are coming from based on the data it is given [6]. It helps to combat any black box decision making as well as provide future research teams with the metrics needed to further hyper tune the datasets that the model may require.



```
Evaluation:
                               precision    recall  f1-score   support

Benign [0]                        1.00      1.00      1.00   3636954
Bot [1]                           0.48      0.97      0.65      3146
DDoS [2]                          1.00      1.00      1.00    204842
DoS GoldenEye [3]                 1.00      1.00      1.00     16468
DoS Hulk [4]                      1.00      1.00      1.00    369716
DoS Slowhttptest [5]              0.98      1.00      0.99      8798
DoS slowloris [6]                 1.00      0.99      0.99      9274
FTP-Patator [7]                   1.00      1.00      1.00     12700
Heartbleed [8]                    1.00      1.00      1.00        18
Infiltration [9]                  0.97      1.00      0.98        58
PortScan [10]                     0.99      1.00      1.00    254288
SSH-Patator [11]                  0.97      0.99      0.98      9436
Web Attack-Brute Force [12]       0.73      0.89      0.80      2412
Web Attack-Sql Injection [13]     1.00      0.65      0.79        34
Web Attack-XSS [14]               0.53      0.29      0.38      1044

accuracy                                    1.00   4529188
macro avg                         0.91      0.92      0.90   4529188
weighted avg                      1.00      1.00      1.00   4529188
```

Fig. 3: DNN Final Scores

Figure 3 details the predictions of 4,529,188 traffic instances across 15 distinct attack categories. As stated before, recall shows how well the model captures actual positive cases and precision shows how many of the predicted positive cases were actually correct. F1 score takes these two metrics and balances them using a harmonic mean such that if one is low, the score will stay around that. F1 score is the true measure of our model's capabilities since both the precision and recall need to remain high in a real-world scenario [10]. The macro avg scores are taking each of the 15 classes as equal weights and are the most important metric since as noted in the weighted avg, the scores are 1.0 due to the high-class imbalances. We are proud that our model can achieve an overall 0.9 F1 score since this is considered excellent and implies the model balances precision and recall well, effectively identifying positive cases while minimizing false positives and negatives [10].

Though this model has outstanding results, it is not perfect. It is evident that the main attack case weighing this down is the XSS web attack with only a 0.38 F1 score, meaning the model struggles to pinpoint this pattern difference with other classes. Additionally, the Bot attack class has high recall but lowered 0.48 precision which indicates that it may have a false positive trend. These two attack classes will cause operators to see possible cases of an attack that truly is not there. This dataset is also inherently flawed with certain classes only having tens or hundreds of instances. Statistical testing in these small classes might yield wide confidence intervals. Although the classes with small representations and good results show promise, statistically these can be prone to change in a real-world case as there is much less context for the model to be trained on. These flaws will be more detailed in the future directions as a team could most easily fix these issues and fine tune the model.

D. Post-Evaluation Labeling

Once the model has finished its initial training on the CICIDS dataset, it is then ready to receive unlabeled data from the IDS. This is in the form of a post evaluation function within Google Collab's active runtime. Our implementation included using a google drive folder full of unlabeled CSVs that have been transformed from PCAP files by CIC Flowmeter. This will most accurately simulate real time data coming in from the IDS in order to replicate a real use case.

First, the post evaluation process loads this data and then preprocesses it similarly to the last section. It additionally drops unwanted columns and matches them up to the CICIDS2017 dataset by label name to preserve consistency. This includes removal of data such as IP addresses since this would be noise for the model's evaluation. Once these columns are in order it labels packet by packet and puts them into the new Label column in the csv file which is stored in the google drive for access of the results from Suricata. Our team was able to generate multiple attack types including DoS Hulk, Goldeneye, and port scans. The model was able to detect packets that matched up nearly identical to these without knowing any labeling beforehand. We also ran benign packets just to check for false positives and the model was able to label them entirely as benign. This presented us with the model's usefulness in the real-world scenario of an AI Driven Threat modeling software.

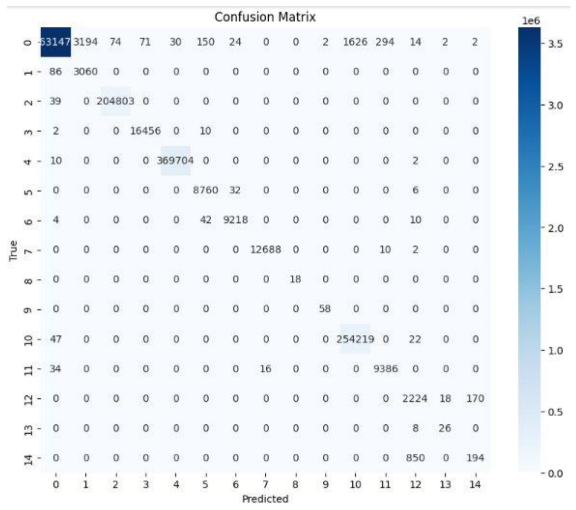


Fig. 4: DNN Confusion Matrix

Although our packet labeling came out as best as it possibly could, there is some notable improvement that is presented in Figure 4. This shows the confusion matrix that our model shows us after running the full training process. The X axis shows the predicted attack classes, and the Y axis shows the actual classes. If the model were perfect the only packets that would be presented would lie in a diagonal line matching up each predicted class with its corresponding true class. This is not the case, more so with packets being labeled as false positives in the top row. Our IDS is geared to be more hyper-sensitive such that we have many more false positives than false negatives which would harm a system much more. False positives, although in a small amount, are still an annoyance for a system administrator. These false positives are seen more so with Bot and Port scan attacks. This also highlights the exact flaws seen within the XSS classification 14 being commonly mislabeled as a Brute Force web attack at 12. Similarly, the Brute force is commonly mislabeled as XSS attacks.

These minor drawbacks highlighted in the Confusion Matrix are meant to be kept in mind when using the post evaluation model to label data that has not yet been labeled. These drawbacks will manifest themselves in those labels and must be considered when deployed inside of a real system architecture. More extensive testing with this post evaluation function would prove that these flaws can and will occur occasionally. However, this will be the case with any AI implementation, and the most a team can do is reduce the occurrences.

E. IDS Enhancement

To connect the AI workflow back to the IDS, our team would have liked to integrate the AI into the IDS code itself using AWS Lambda scripts and Suricata API, however with the time constraints in place we took the simplest route of having the network administrator analyze the attacks themselves.

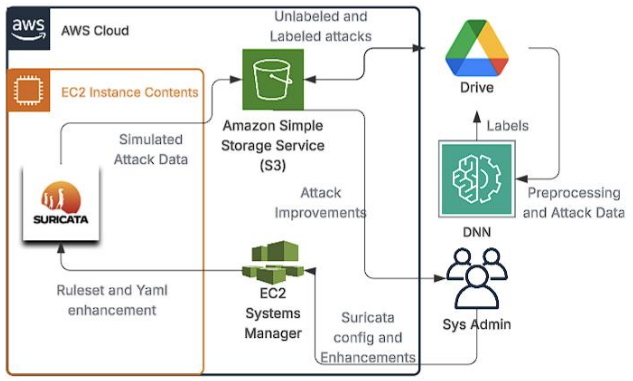

aws
AWS Cloud
Unlabeled and Labeled attacks
Drive
EC2 Instance Contents
Simulated Attack Data
Amazon Simple Storage Service (S3)
Labels
SURICATA
Attack Improvements
Preprocessing and Attack Data
DNN
Ruleset and Yaml enhancement
EC2 Systems Manager
Suricata config and Enhancements
Sys Admin

Fig. 5: Implemented System Architecture

Figure 5 describes the entire system architecture from a high level including every technology involved in the actual implementation of our conceptual idea. This aims to give an understanding of the user or System Administrator point of view within the system, which in our case is the enhancement being referenced. Given more time, the S3 bucket would be directly feeding Suricata with YAML file and ruleset improvements based on the attacks running through the AI. Instead, our team acted as the user and developed a script that would pull the attack files and then reference the attack names found in the files. We utilized ChatGPT API to mimic a research script and gave a high-level description of each attack present in the now labeled PCAP traffic from Suricata. These scripts would then call the API to detail exact ruleset enhancements which included modifying the current Suricata YAML file to now detect signatures from the uncaptured attacks. For example, our DoS Hulk originally went undetected, however, this script added `HTTP GET` flood signatures into the YAML file after analysis of the AI labeling. Additionally, there are firewall rules added in to the .rules and recommended analysis in the `.eve and .fast` logs which will even further optimize the IDS to catch the attack in the future.

It is worth noting that these enhancements are only an example to show a small amount of what is capable within the optimization of AI within security tools. This use case can be expanded on and tailored to corporation security needs and goals.

V. TECHNOLOGICAL FUTURE DIRECTION

Although our implementation came with great statistical success, our proof-of-concept was not fully implemented to its greatest extent. Our team's implementation was solely focused on generating attack data from replays within Suricata. In future variations of this work, a team can expand on this concept with the implementation of a fleshed out GNS3 server containing virtual router and switch nodes to replicate an enterprise environment. This concept in conjunction with the incorporation of MITRE's Caldera as a real attack simulation source would create an authentic flow of attack data that can supplement as a self-generated dataset to enhance the AI's detection over time. This would create a self-propagating environment in regard to the IDS. The System Administrator would also be able to tweak this system depending on attack requirements and evolving threats.

Suricata is an IDS but may also act as an Intrusion Prevention System (IPS). This prevention mode would add a layer of complexity into the system of the original detection and combining this idea with direct real-time labeled attack data from the AI to the IPS would make it such that the system could react in real time to any attacks present without the help of a third party. The idea of expanding the toolset functionality with regard to our model also applies to any cyber tool that has a statistical error rate. AI can be put in place as a countermeasure to statistical anomalies and evolving threats regarding many tools such as a SIEM, SOAR orchestration, or even in regard to an Endpoint Detection and Response system (EDR). This will not take the place of an administrator overseeing these tools but instead assist in being more accurate overall than if the AI had not been analyzing any packets or specific data to begin with. Our project serves as a prototype for statistical AI models and configurations that work well in unison with PCAP data flows, and we encourage modification of our GitHub code and expansion of the system with respect to these ideas to create a stronger, more secure, enterprise system [11].

Additionally, future work would include incorporating other competitive projects with a similar goal in order to benchmark our statistical results and attempt to surpass the current gold standards in the market [5].

V. CONCLUSION

The implementation of AI in cybersecurity poses a unique opportunity to enhance threat detection beyond conventional rule-based systems. Our work presents a proof-of-concept for an AI-driven threat detection and response system that leverages deep learning and explainable AI (XAI) to identify and interpret cyber threats in real time. Through extensive preprocessing, model optimization, and integration with Suricata IDS, we achieved an F1 score of 0.9 across multiple attack categories in the CICIDS2017 dataset. This high performance emphasizes the potential of supervised learning models in cybersecurity, particularly when paired with methods such as FocalLoss to handle class imbalance and SHAP for model explainability.

A key contribution of our work is the development of a post-evaluation pipeline that autonomously labels live packet captures and translates them into actionable insights for the IDS. This not only demonstrates the feasibility of integrating AI with security infrastructure but also introduces a scalable way to iteratively improve detection systems. To bridge the gap between automated threat detection and actionable system improvements, our team integrated the ChatGPT API as part of an intelligent enhancement layer. Once the AI model labeled traffic from Suricata, the ChatGPT API was used to generate high-level summaries and descriptive insights about the detected attacks. These insights were then used to suggest concrete ruleset modifications, including adjustments to the Suricata YAML file and the creation of custom detection signatures for previously undetected threats such as DoS Hulk attacks. This approach demonstrated how natural language models can serve as an intermediary between AI-driven classification and decision-making. By translating complex model outputs into understandable guidance, ChatGPT enabled more intuitive ruleset updates in refining intrusion detection systems with minimal manual effort.

Despite these successes, our model's performance on low-frequency classes like XSS and Bot attacks highlights the persistent challenges in minority class detection. This reinforces the need for continuous data enrichment, adversarial testing, and more robust evaluation environments such as a GNS3-Caldera hybrid for red team simulations. The vision moving forward is to incorporate real-time prevention capabilities, enabling the IDS to evolve into a fully autonomous intrusion prevention system (IPS). In conclusion, our project lays the groundwork for integrating machine learning and explainable AI within modern cybersecurity workflows. By combining statistical accuracy with real-world system design, we provide a blueprint for future teams to expand upon, integrating deeper automation, more adaptable learning systems, and broader interoperability with enterprise-grade security solutions.

REFERENCES

[1] Valentin. (2015). Replaying packets with tcpreplay. Blogspot.com. https://yy27.blogspot.com/2015/01/replayingpackets-with-tcpreplay.html

[2] G. Vidhani, "Understanding AI Models: A Beginner's Guide," OpenXcell, Nov. 15, 2024. [Online]. Available: https://www.openxcell.com/blog/ai-models/.

[3] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "A Detailed Analysis of the CICIDS2017 Data Set," in Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2018), 2018, [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[5] H. H. Tran, T. H. La, V. T. Vu and D. M. Vu, "Deep Inductive Transfer Learning Approach for Network Attacks Detection," 2023 International Conference on Advanced Computing and Analytics (ACOMPA), Da Nang City, Vietnam, 2023, pp. 64-69, doi: 10.1109/ACOMPA61072.2023.00020.

[6] Tabassi, E. (2023), Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST Trustworthy and Responsible AI, National Institute of Standards and Technology, Gaithersburg, MD, [Online], https://doi.org/10.6028/NIST.AI.100-1, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=936225

[7] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *(ICISSP)*, Portugal, Jan. 2018. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[8] PyTorch Lightning Documentation. [Online]. Available: https://pytorch-lightning.readthedocs.io/en/stable/.

[9] OISF, "suricata — Suricata User Guide," *Suricata Documentation*, [Online]. Available: https://docs.suricata.io/en/latest/manpages/suricata.html.

[10] N. Buhl, "F1 Score in Machine Learning," Encord Blog, Jul. 18, 2023. [Online]. Available: https://encord.com/blog/f1-score-in-machine-learning/.

[11] g8cyse492, "CICIDS2017FINAL," GitHub. Available: https://github.com/g8cyse492/CICIDS2017FINAL/blob/main/CICIDS2017FINAL.ipynb.