Comparing Rule-Based and ML-Based Detection of C2 Channels

RIT Cybersecurity MS Project

Nicholas Geigel

May 2025

1 Introduction

Networked systems are an unavoidable modern life, powering everything from critical infrastructure to everyday consumer devices. As the scale and complexity of these systems grow, so does the need to monitor and secure the traffic that flows between them. Intrusion Detection Systems (IDSs) are a key line of defense. IDSs monitor network traffic and generate alerts when suspicious or unauthorized activity is detected [1]. These systems play an important role in identifying attacks before they escalate.

One type of threat that IDSs are designed to detect is command and control (C2) traffic. C2 frameworks are used by attackers to maintain access to compromised systems, run commands, move laterally through a network, and exfiltrate data. These frameworks typically establish a communication channel between the infected device and a remote C2 server, allowing attackers to control compromised machines remotely [2]. Many of these frameworks use legitimate-looking protocols like HTTPS to blend in with normal traffic, making detection difficult. There are also a wide range of capable, open-source C2 tools readily available to attackers, which increases the risk for defenders.

Suricata is an open-source intrusion detection system that analyzes network traffic using a combination of rules, signatures, and protocol decoding [3]. It is often used with community-maintained rule sets like Emerging Threats Open to identify known patterns of malicious activity.

This project compares a behavior-based machine learning approach to Suricata's rule-based detection in identifying C2 traffic. Specifically, it evaluates a custom-trained Random Forest model against Suricata using the Emerging Threats Open ruleset. The focus is on HTTPS-based C2 frameworks, with all traffic collected in a controlled lab environment designed to simulate realistic usage.

The rest of this paper is organized as follows. Section 2 reviews related work on C2 detection and machine learning applications in intrusion detection. Section 3 describes the methodology for collecting data, extracting features, training the model, and evaluating both the model and Suricata. Section 4 presents the results. Section 5 discusses key findings, limitations, and areas for improvement. Section 6 concludes the paper. All source code and datasets can be found at: https://github.com/nickg-24/c2-ms-capstone

2 Related Works

2.1 Supervised Learning and IDS Fundamentals

Abdallah et al. present a broad survey of supervised machine learning for intrusion detection systems (IDS) [4]. They compare detection performance across four flow-based datasets: KDD'99, NSL-KDD, CICIDS2017,

and UNSW-NB15. The paper also introduces a taxonomy that helps guide algorithm selection based on dataset characteristics. Across the board, Random Forest classifiers consistently outperform other models in terms of accuracy and false positive rate, especially when paired with feature selection. Reported results include accuracies as high as 99.9% on the NSL-KDD dataset. Aside from a single outlier, the surveyed approaches achieve at least 91% accuracy.

The authors emphasize the role of feature selection in improving model performance. It's used to remove irrelevant or redundant input features and helps mitigate the "curse of dimensionality" [4], which occurs when too many features and too little training data make it harder for models to generalize. The paper also discusses recurring challenges in IDS datasets, including class imbalance, and highlights how sampling strategies can improve detection performance.

Their use of both hold-out and N-fold cross-validation was helpful in shaping the validation strategy used in this project.

2.2 Challenges in Flow-Based Detection and Generalization

Talukder et al. propose a machine learning approach for detecting network intrusions in large, unbalanced datasets [5]. Their approach combines Random Oversampling (RO), Stacking Feature Embedding (SFE), and Principal Component Analysis (PCA) to improve classification performance while keeping the model general. They evaluate their model on three datasets: UNSW-NB15, CIC-IDS-2017, and CIC-IDS-2018, achieving high accuracy with each model having 99% accuracy at the minimum.

While their model is more complex and deals with a broader scope, their findings are still relevant to this project. This work reinforces the idea that you can get strong results with relatively lightweight models, if features are appropriately selected. Although this project does not apply advanced techniques like feature embedding or principal component analysis, it shares the same goal of building a lightweight, generalizable model that can handle noisy, imbalanced datasets. Their findings support the use of Random Forest in the context of classifying network traffic.

2.3 Machine Learning for C2 Detection

Palo Alto Network's Unit 42 published an article describing the using of deep learning models to detect C2 traffic generated by malware such as Emotet and Sality. By focusing on HTTP packet headers, the Unit 42 team was able to train a model that achieved a 98% precision rate with a false positive rate of less than 0.025% [6]. Their success was due in part to a large dataset of 60 million HTTP session headers fed into a deep learning model. They note that C2 packets often evade traditional signature-based detection because they contain randomized URIs, generic User-Agents, and dynamic IP addresses. Deep learning was used to identify structure patterns rather than fixed byte sequences, which help to detect novel attacks. A notable finding is that their model was able to automatically extract significant features to aid in detection, as opposed to depending on any one field.

Similarly, Känzig et al. examine how Random Forest classifiers can be used to classify C2 traffic specifically [7]. Their model used 20 flow-based features and was trained on one year's worth of data. The model was then tested across datasets from different years, achieving 99% precision and over 90% recall, speaking to its generalization capability. An interesting note about this study is that classification occurred in real time, producing results quickly enough for use in operational scenarios. While their features are flow-based, the focus on behavioral patterns supports this project's use of structural indicators within packets to detect

C2 activity.

2.4 Limitations of Rule-Based Detection

Macedo evaluates the the effectiveness of signature-based IDS in detecting and exposes a gap in C2 traffic detection. The study focuses on the Covenant C2 framework, iteratively developing rulesets to analyze how configuration changes affect detection rates [8]. A notable takeaway from this study is that TLS encryption renders most public rulesets ineffective, as many rely on detecting fixed byte sequences in plaintext traffic.

Macedo also explored how custom Suricata scripts to detect indicators such as self-signed TLS certificates, but notes that the results are not reliable enough on their own for identifying C2 traffic. This work reinforces the limitations of relying solely on static rules, especially when dealing with modern and adaptable C2 frameworks. Overall, this paper supports the need for more generalizable detection strategies such as the machine learning approaches discussed in earlier works and applied in this project.

2.5 Related Works Overview

While much of the existing literature focuses on flow-based intrusion detection, this project applies supervised learning and packet-level features to detect C2 traffic, aiming to offer a slightly different perspective. Random Forest appears across multiple sources as a strong performer, justifying its use in this project. The reviewed works also help to identify useful validation techniques as well as place an emphasis on the weaknesses of signature-based detection, especially with encrypted traffic. These insights and limitations helped to shape the model architecture, feature selection, evaluation strategy, and test environment design, each of which will be detailed in the following sections.

3 Methodology

This section describes the process used to collect network data, extract features, train a machine learning model, and evaluate its performance. The goal of this project was to detect C2 traffic using supervised learning on packet-level features. Six open-source C2 frameworks were deployed in a test environment to generate malicious traffic. Packet captures were collected, processed, labeled, and a Random Forest classifier was trained to distinguish between C2 and normal traffic. The following subsections outline the network setup, data generation process, feature selections, model training, and evaluation procedures.

3.1 Data Collection

Data collection for this project was carried out in a Proxmox-based virtual environment. Proxmox is an open-source platform for managing virtualized systems and was selected for its flexibility and scalability. Its support for both virtual machines and containers made it easy to adapt the test environment as project needs evolved. Proxmox also allowed for rapid deployment and remote management of multiple components, which was especially useful when troubleshooting or reconfiguring parts of the environment.

The testbed included several core components: a router, firewall, DNS server, and multiple virtual machines. A simplified version of the topology is shown in Figure 1.

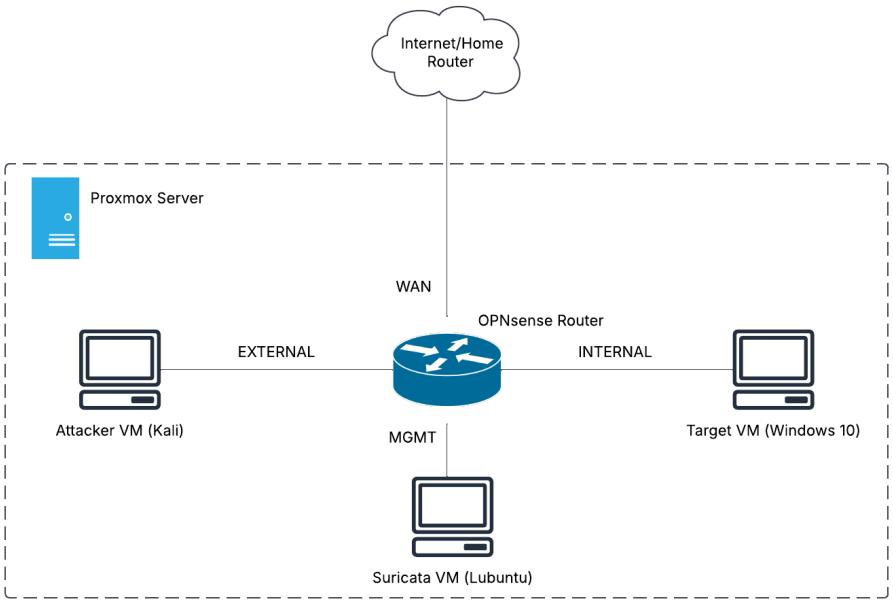

Figure 1: Simplified Topology

1. **OPNsense Router:** Directed all traffic within the Proxmox environment.
2. **Attacker VM:**A Kali Linux virtual machine used to host the C2 frameworks and issue commands to the target.
3. **Target VM:** A Windows 10 machine where C2 payloads were executed. This VM also captured traffic using Wireshark.
4. **Suricata VM:** A Lubuntu VM used to process PCAPs with Suricata and the ET Open ruleset.

The target VM was placed on its own network and used to capture all traffic. This setup allowed the system to record communication with both external (internet) services and internal components, including the attacker VM. Capturing traffic at this point simplified the environment and avoided the need for complex routing or port mirroring.

To simulate realistic network behavior, the environment generated both normal and malicious HTTPS traffic. Normal traffic included typical web browsing and background system activity, such as telemetry from built-in Windows 10 services. This baseline was critical for evaluating the model's ability to distinguish benign from malicious behavior.

Malicious traffic was generated using six C2 frameworks: Metasploit, Empire, Covenant, Sliver, Merlin, and Posh. These frameworks were selected based on their open-source availability, relative ease of setup, popularity, and diversity in communication style. Some operate on a task-based model, while others support interactive sessions. The goal was to include a variety of C2 behaviors to help the model generalize more effectively across different frameworks.

Each test followed a consistent process: the C2 framework was set up with any required dependencies, a payload was generated and transferred to the target VM, and a Wireshark capture was started. Once the payload executed, the attacker interacted with the target using the C2 framework. Most datasets

captured around 10,000 packets and involved performing similar sets of actions across frameworks to ensure consistency.

The captured traffic was organized into four categories. C2-only datasets consist exclusively of malicious traffic, separated into beaconing and task-related activity. Normal datasets contain benign background traffic from idle Windows services and light web browsing. Mixed datasets combine C2 and normal traffic to emulate more realistic network conditions, with roughly a 10 to 90 class balance between C2 and normal packets. Combined datasets include multiple C2 frameworks along with normal traffic and are intended to evaluate generalization. Each combined dataset builds on the previous one by increasing the diversity or volume of included traffic. This progression helps test how well the model scales and adapts across varying combinations of C2 behaviors. Across all C2 datasets, the attacker performed a consistent set of operations such as shell commands, file transfers, screenshot captures, and polling behaviors. A summary of all datasets is provided in Table 1. For a full list of actions and detailed breakdowns, refer to the datasets.md file in the project GitHub repository.

Table 1: Summary of Datasets Collected

Dataset	**Type**	**Total Packets**
Metasploit	C2-only	9123
Covenant	C2-only	12276
Empire	C2-only	10570
Sliver	C2-only	10151
Merlin	C2-only	8559
Posh	C2-only	10242
Normal Traffic	Normal	249057
Metasploit Mixed	Mixed	15216
Covenant Mixed	Mixed	12084
Empire Mixed	Mixed	13973
Merlin Mixed	Mixed	59593
Posh Mixed	Mixed	10724
Sliver Mixed	Mixed	20013
Combined 1	Combined	270456
Combined 2	Combined	281026
Combined 3	Combined	291177
Combined 4	Combined	299736
Combined 5	Combined	309978
Combined 6	Combined	361975

3.2 Feature Extraction

Packet-level features were extracted from each PCAP file using a custom python script that interfaces with Tshark. Tshark is a command line tool from the Wireshark suite that allows users to interact with packet captures. While the original plan was to use flow-based features similar to those seen in the related works, this approach quickly proved difficult to scale. Wireshark treats each conversation between the C2 server and the target as a single flow, which significantly limited the number of data points available. Given the number of C2 frameworks involved and the additional work required to generate usable flow data and integrate it into a complete training and evaluation pipeline, this approach was not feasible within the scope and timeline of the project. As a result, I opted to explore a packet-based approach instead. It also presented an opportunity to evaluate whether low-level structural and timing characteristics could be effective in detecting C2 activity,

particularly in encrypted environments where payload contents are not available.

The script processes IP-layer packets only, filtering out unneeded packets such as ARPs and layer 2 broadcasts early in the pipeline. For each packet, it extracts core features such as source and destination IP addresses, transport-layer protocol, frame length, and TCP flags. C2 packets are labeled dynamically using a hardcoded C2 server IP address, and the script assigns a binary `c2_label` based on whether the source or destination matches this value. Similarly, the script assigns a binary internal/external flag for both the source and destination IPs. Traffic from `INTERNAL` network shown in the topology is considered internal, and all other traffic is considered external. This setup allowed traffic from both the lab environment and the internet to be grouped under the same external classification, which simplified testing and made the system more general.

These two abstractions, the labeling of C2 traffic and internal vs external IP addresses, were added specifically to address one of the limitations of signature-based systems discussed in the related works. Public rulesets often fail to generalize because they rely on hard-coded metadata such as known malicious IP addresses or domains. Abstracting these values into behavioral features helped make the system more IP-agnostic and less dependent on environment-specific details.

Additionally, TCP flags (SYN, ACK, FIN, and RST) are extracted from the hexadecimal flag value and transformed into separate binary features. This was done in an effort to allow the model pick up on subtle connection behaviors that may differ between C2 traffic and normal traffic.

To capture timing behavior, the script calculates the time since the last packet (`time_since_last`) and the ratio between consecutive inter-arrival times (`delta_t_ratio`). To provide the model with context over short-term trends, rolling averages and standard deviations of `time_since_last` are computed over three window sizes (3, 5, and 10 packets). These rolling features help encode local patterns in packet timing that may vary between C2 frameworks or operational modes. Rolling features were a major focus during development, since it was difficult to find packet-level features that did not rely on known indicators of compromise. With payloads encrypted, the only options available were structural and behavioral.

Once feature extraction is complete, redundant fields such as raw IP addresses and raw TCP flag values are dropped. The final dataset is written to CSV for downstream training and evaluation.

This packet-centric approach differs from traditional flow-based systems like those described by the related works, which rely on session-level aggregation. By working at the packet level, this project focuses on detecting C2 behaviors based on timing and structural characteristics, without relying on decrypted payloads or full connection metadata.

3.3 Model Training

Initial experiments used logistic regression as a baseline due to its simplicity and ease of interpretation. However, the model performed poorly in early tests, especially in identifying C2 traffic. Based on those results and guidance from related works, the approach was shifted to a Random Forest classifier, which provided more consistent and improved performance across all datasets.

Model development followed an iterative approach. After collecting traffic from each C2 framework, I appended the data to the growing training set and retrained the model. This allowed me to gauge how adding more C2 diversity affected generalization, and it served as a sanity check to ensure the model was learning something meaningful.

As the training dataset evolved, I decided to incorporate the mixed datasets into the final training set. This was done to better reflect realistic network conditions, where malicious traffic is often embedded within

otherwise normal traffic. However, since each mixed dataset included a large proportion of normal packets (typically around 90 percent), this skewed the overall class distribution heavily toward normal traffic.

To address this, a 20:80 C2-to-normal class ratio was enforced when building the final training set based on Combined 6. This dataset included all previously captured traffic along with the full set of mixed datasets. After filtering, shuffling, and re-balancing the data, the features were standardized using a **StandardScaler** to normalize feature ranges. The final dataset was then split into training and test sets using a stratified 60/40 split to preserve class balance across both sets. The trained Random Forest model was saved and reused for all downstream evaluation.

3.4 Evaluation Setup

To evaluate how well the model generalized across different conditions, a mix of standard validation techniques and targeted inference tests was used. These evaluations were designed to measure performance not just on held-out data, but also in more realistic scenarios and against C2 traffic the model had not seen before.

The first set of tests focused on general validation strategies. The five main methods are as follows:

- Hold-Out Validation: The dataset was split into fixed training and testing sets. This is the most basic form of validation and was used to establish a quick performance baseline [9].
- Stratified K-Fold: The data was divided into five folds while preserving class ratios. Each fold was used as a test set once, which helped reduce the impact of randomness in any single split [10].
- Repeated Stratified K-Fold: This extends stratified k-fold by repeating the process multiple times with different splits. It provided a more stable estimate of model performance by averaging out variability across runs [9].
- Stratified Shuffle Split: Similar to hold-out, but repeated across several random splits while maintaining the same class distribution. This was helpful to verify consistency across different random samples.
- Leave-One-Group-Out (LOGO): Each data source (e.g., a capture session) was treated as a group, and the model was trained on all groups except one. This helped test the model's ability to generalize across different data sources and session-level variability [9].

After that, inference tests were ran on each of the individual mixed datasets. These tests simulated more realistic conditions where C2 traffic was embedded within mostly normal activity, with malicious packets making up only a small portion of the total traffic.

Full-dataset inference occurred on both Combined 5 and Combined 6. The goal here was to see how adding the mixed datasets to the training set (in Combined 6) affected overall model performance. This was important because the mixed datasets introduced a large amount of additional normal traffic.

Finally came leave-one-framework-out testing. In each run, the model was trained on five C2 frameworks plus normal traffic and tested on the sixth. This setup evaluated how well the model could detect C2 behaviors it had never seen during training. Although this resembles a group-based strategy, it differs from standard Leave-One-Group-Out (LOGO) validation, where each "group" typically corresponds to a capture session or file. In this case, the held-out unit was not a single session, but all data from an entire C2 framework. This distinction matters because it tests whether the model can generalize to completely new attacker behavior, rather than simply handling variation across data sources.

As a final exploratory step, I also ran a few variations of the leave-one-framework-out tests: one using XGBoost instead of Random Forest, and another with the frame length feature removed. These tests showed minimal change in performance and are not discussed further.

To compare machine learning-based detection with traditional rule-based approaches, each PCAP (normal, C2-only, and mixed) was ran through Suricata using the Emerging Threats Open ruleset.

4 Results

4.1 Validation Results

Table 2 summarizes model performance across multiple validation methods using both the Combined 5 and Combined 6 datasets. Metrics focus on detection of class 1 (C2) traffic.

Validation Method	**Dataset**	**F1 (C2)**	**Recall (C2)**	**AUC**
Hold-Out	Combined 5	0.944	0.937	0.995
Stratified K-Fold	Combined 5	0.946	0.941	0.996
Repeated Stratified K-Fold	Combined 5	0.948	0.942	0.996
Stratified Shuffle Split	Combined 5	0.943	0.936	0.995
Hold-Out	Combined 6	0.943	0.936	0.995
Stratified K-Fold	Combined 6	0.945	0.938	0.995
Stratified Shuffle Split	Combined 6	0.942	0.935	0.995

Table 2: Summary of validation performance across different methods. F1 and Recall scores reflect class 1 (C2) detection performance.

Table 3 presents results from Leave-One-Group-Out (LOGO) validation. In this setup, each group represents an individual capture session. The model was trained on all but one session and tested on the held-out session to evaluate generalization across recording conditions. Results are reported for models trained on Combined 5 and Combined 6.

Framework	**Combined 5**			**Combined 6**		
	F1 (C2)	Recall (C2)	AUC	F1 (C2)	Recall (C2)	AUC
Metasploit	0.619	0.461	0.977	0.731	0.593	0.974
Covenant	0.821	0.716	0.971	0.885	0.818	0.973
Empire	0.914	0.866	0.994	0.905	0.853	0.994
Sliver	0.316	0.193	0.902	0.349	0.218	0.911
Merlin	0.508	0.351	0.894	0.570	0.411	0.904
Posh	0.905	0.851	0.996	0.925	0.887	0.996

Table 3: Per-framework performance using Leave-One-Group-Out (LOGO) validation. Combined 6 includes additional mixed datasets.

Figure 2 shows the most important features from the hold-out validation, highlighting that timing and structural attributes that had the greatest influence on the model's predictions.

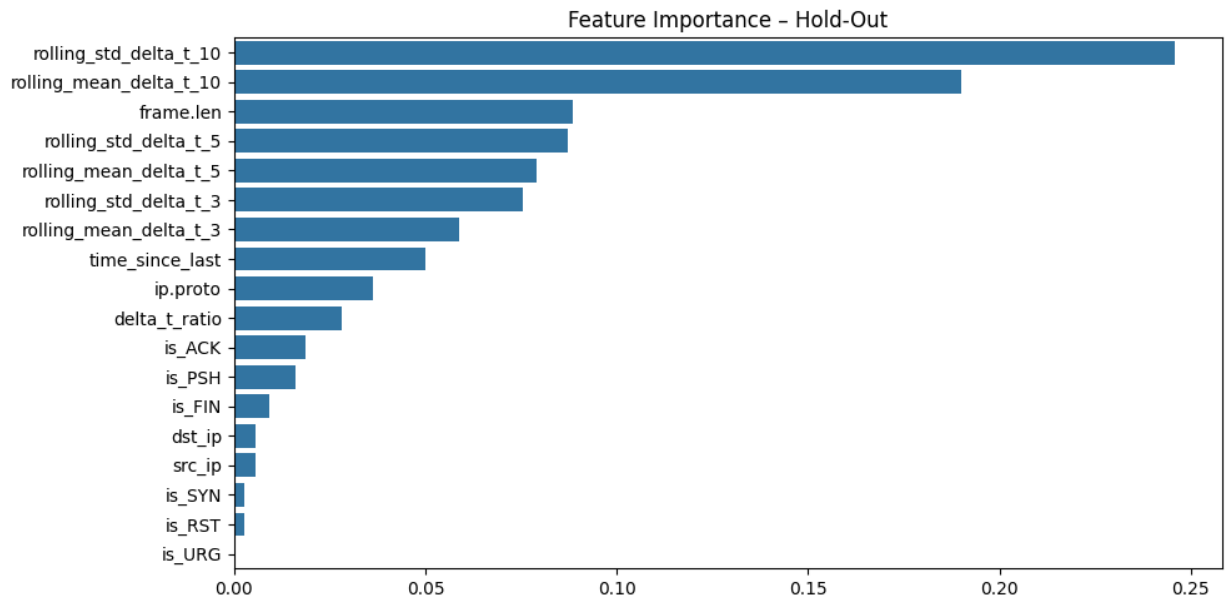

Figure 2: Feature Importance - Hold Out

4.2 Inference on Mixed Traffic

Table 4 shows the results of inference runs on mixed datasets, which contain both normal and malicious packets. These evaluations simulate realistic deployment conditions by testing whether the model can identify C2 activity embedded within largely benign traffic. Models trained on Combined 5 and Combined 6 were tested on each framework's corresponding mixed dataset.

Table 4: Inference results on mixed datasets using a 0.5 threshold

Framework	**F1 (C2) C5**	**Recall C2 C5**	**AUC C5**	**F1 (C2) C6**	**Recall C2 C6**	**AUC C6**
Metasploit	0.6881	0.655	0.9516	0.8833	0.919	0.992
Empire	0.8137	0.7936	0.9874	0.9563	0.9506	0.9987
Covenant	0.6053	0.6401	0.9307	0.8377	0.8848	0.9873
Sliver (session mode)	0.6795	0.5618	0.9543	0.6667	0.5452	0.9555
Sliver (beacon mode)	0.4682	0.4646	0.906	0.5175	0.5015	0.9137
Merlin	0.5977	0.6474	0.971	0.587	0.5539	0.9299
Posh	0.862	0.822	0.9914	0.962	0.9702	0.9988

4.3 Leave-One-Framework-Out

Table 5 displays results from the Leave-One-Framework-Out (LOFO) evaluation. For each run, the model was trained on five C2 frameworks and tested on the sixth, which was excluded entirely during training. The goal is to measure how well the model generalizes to previously unseen C2 traffic.

Table 5: Leave-One-Framework-Out Results for Each C2 Framework

Framework	Precision (C2)	Recall (C2)	F1-Score (C2)
Covenant	0.96	0.72	0.82
Empire	0.97	0.87	0.91
Merlin	0.92	0.35	0.51
Metasploit	0.94	0.46	0.62
PoshC2	0.97	0.85	0.90
Sliver	0.87	0.19	0.32

4.4 Suricata - ET Open

Suricata, configured with the ET Open ruleset, did not generate any alerts when processing the C2 traffic PCAPs.

5 Discussion & Future Works

5.1 General Discussion

As shown in Table 2, the model demonstrated consistently strong performance across multiple validation methods, including hold-out, K-Fold, and shuffle-based splits. F1 scores and recall for class 1 (C2) remained in the 0.94–0.95 range, with AUC values close to 0.995. These results suggest that the model was able to reliably distinguish between normal and C2 traffic across different evaluation strategies. The consistency also indicates that the model did not overfit to specific packet samples. One way to check for overfitting is to look for performance drops on unseen data. Since performance stayed consistent across different splits, results suggest the model was picking up general patterns instead of just memorizing the training data.

LOGO validation results (Table 3) tell a different story. Performance dropped, sometimes significantly, when the model was tested on a capture session it had not seen during training. Frameworks like Empire and PoshC2 still performed well (F1 scores above 0.9), indicating that their traffic had consistent and learnable patterns. In contrast, Sliver and Merlin performed much worse, with F1 scores of 0.32 and 0.51 respectively. These results highlight the model's difficulty in generalizing across session-level differences, especially for frameworks with more variable or evasive traffic. Adding mixed traffic to the training set (as in Combined 6) helped improve performance slightly, but not enough to meaningfully improve generalization for the frameworks that struggled most.

Inference on mixed datasets (Table 4) was meant to simulate a more realistic deployment scenario, where C2 traffic is embedded within a larger stream of normal traffic. The addition of mixed traffic to the training set significantly improved detection performance across several frameworks. Metasploit's F1 score increased from 0.69 to 0.88, Covenant from 0.61 to 0.84, and Empire from 0.81 to 0.96. Sliver and Merlin still performed poorly, suggesting either that their traffic is inherently harder to detect or that the collected datasets for those frameworks lacked quality or diversity. The trend is consistent: Sliver and Merlin underperform compared to other frameworks. These results reinforce the importance of training on large and diverse datasets that include a range of normal and malicious behaviors.

Leave-One-Framework-Out results (Table 5) provide another view into how well the model can handle unseen threats. The model generalized well to Empire, PoshC2, and Covenant, each scoring above 0.82 on

F1. Performance dropped sharply for Metasploit and Merlin, and was especially poor for Sliver, which had a recall of just 0.19. These results show that different frameworks leave different network footprints. Some are relatively distinct and easier to identify, while others closely resemble benign traffic or introduce enough variation to slip past the model.

The feature importance results for the model (Figure 2) help explain why the model made the decisions it did. The most important features were structural and timing-based, with `rolling_std_delta_t_10`, `rolling_mean_delta_t_10`, and `frame.len` at the top. On the other hand, abstracted metadata like TCP flags and internal versus external IP indicators contributed very little. This makes sense given the focus of the project. The goal was to avoid relying on known indicators of compromise and instead detect behavioral patterns. Since all traffic was encrypted and used benign-looking infrastructure, the model had to rely on timing and structural cues. The fact that it performed as well as it did without access to payloads or IP-level information supports the approach of using behavioral features for C2 detection.

5.2 Comparison to Suricata and ET Open

Both the model and Suricata were evaluated on the same PCAPs: mixed traffic datasets with an approximate 10:90 C2-to-normal ratio. Suricata, using the ET Open ruleset, generated zero alerts across all PCAPs, including both C2-only and mixed captures. While surprising at first, these results make sense upon closer inspection. Suricata operates by ingesting large, highly specific rulesets. In the case of ET Open, these rules rely on known indicators of compromise such as malicious IP addresses, domain names, or TLS certificate fingerprints. In this controlled lab environment, none of those were present. All C2 communication used self-signed or framework-generated certificates, and the traffic did not contain any previously observed domains or certs. As a result, the signature-based rules were ineffective. These findings align with Macedo's evaluation of Suricata and the broader limitations of signature-based detection [8], reinforcing a known gap in how well these systems detect C2 traffic under novel conditions.

With that gap clearly identified, the question becomes: how can a model like this one help fill it? The model was intentionally designed not to rely on traditional indicators of compromise. Instead, it aimed to detect malicious traffic based on behavior, not prior knowledge. IP addresses and flags were abstracted into internal versus external classes to avoid encoding brittle rules or introducing bias tied to specific infrastructure. Detection focused on timing patterns, connection behaviors, and structural characteristics, features that are difficult to spoof and often overlooked by rule-based systems. This gave the model the opportunity to recognize patterns that may appear benign to a signature-based IDS.

So what does this comparison tell us? ET Open is highly precise when a matching signature exists, but it is brittle when dealing with new or unknown traffic. Without recognizable indicators, its detection capability is basically shut off, assuming no other systems are in place to catch what it misses. This highlights a core limitation of rule-based systems: they work well for known threats, but offer little protection against unfamiliar or lightly modified attacks.

In contrast, the model was able to produce detections for every C2 framework tested. Performance was better when the framework was included in training, but even when generalizing to unseen frameworks, the model still caught some C2 traffic. This includes Sliver and Merlin, which had the weakest results overall, but still led to at least some level of detection. While these numbers were far from ideal, they stand in contrast to Suricata, which did not generate any alerts at all. This points to a potential advantage of behavior-based approaches in cases where traditional signatures fall short. That said, neither method is perfect. Each has its strengths, and they are not mutually exclusive. Putting these two solutions together could be a practical

way to cover each other's weaknesses. The model can help expose traffic that looks suspicious based on behavior, while Suricata can provide more confident alerts when known indicators are present.

5.3 Limitations and Areas for Improvement

While the model was effective at detecting C2 traffic it had seen during training, its ability to generalize varied considerably. Several areas stand out for improvement. One of the most immediate limitations is the use of packet-level features. The current feature set focuses on individual packets and short rolling windows. Although the original plan was to incorporate flow-based features to provide session-level context, developing datasets large enough to support that approach fell outside the scope of this project. This limitation raises important questions. For example, would underperforming frameworks like Sliver or Merlin still evade detection if the model had access to higher-level features like session duration, total bytes, or packet counts? Or would additional context help surface behaviors that are hard to detect at the packet level?

Model architecture is another area worth exploring. A Random Forest classifier was used here because it was simple to implement, fast to train, and performed reasonably well given the limited feature set. It was a practical choice for the scope of this project, but future work could investigate more advanced architectures such as neural networks. Model tuning was also minimal. A more thorough optimization process could reveal performance gains that were not explored here.

As with most machine learning tasks, the effectiveness of the solution comes down to the quality of the data. Building a virtual environment, deploying six different frameworks, troubleshooting setup issues, and accounting for the relatively low packet output of some frameworks all made it difficult to scale. Only a subset of mixed traffic was included in training. Expanding both the volume and diversity of the dataset, especially by including additional environments or network conditions, could improve the model's ability to generalize, particularly for frameworks that performed poorly in LOGO and LOFO testing.

The Suricata evaluation could also be expanded. While the lack of alerts from ET Open illustrated the limitations of signature-based detection, this result was expected given the controlled lab conditions. Future experiments could inject known indicators of compromise to verify rule behavior or evaluate more comprehensive rulesets like ET Pro. Developing custom rules that focus on behavioral indicators rather than static IOCs may also improve detection.

Finally, this project points to the value of hybrid detection systems. Machine learning and signature-based approaches each have strengths and limitations. Combining the two could help offset their individual weaknesses. A model like this could act as a behavioral screen to surface suspicious traffic, while Suricata could provide precise alerts when known indicators are present. Together, they may offer broader and more reliable coverage than either system on its own.

6 Conclusion

This project explored how a machine learning–based approach to C2 detection compares to a traditional rule-based IDS. By building and evaluating a Random Forest model, the project aimed to address some of the limitations of signature-based tools like Suricata, particularly in detecting encrypted or novel C2 traffic. While the results show promise, especially in controlled conditions, the project also highlighted several challenges, most notably around data collection, generalization, and feature design. There's still plenty of

room to improve, especially by expanding dataset diversity, incorporating flow-level context, and exploring hybrid detection strategies that combine the strengths of both ML and rules-based systems.

References

[1] IBM, "What is an intrusion detection system (ids)?" Apr 2023, [Online]. Accessed: May 21, 2025. [Online]. Available: https://www.ibm.com/think/topics/intrusion-detection-system

[2] R. Grimmick, "What is c2? command and control infrastructure explained," Aug 2022, [Online]. Accessed: May 21, 2025. [Online]. Available: https://www.varonis.com/blog/what-is-c2

[3] S. Networks, "Suricata rules," [Online]. Accessed: May 21, 2025. [Online]. Available: https://www.stamus-networks.com/suricata-rules

[4] E. E. Abdallah, W. Eleisah, and A. F. Otoom, "Intrusion detection systems using supervised machine learning techniques: A survey," *Procedia Computer Science*, vol. 201, pp. 205–212, 2022, the 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922004422

[5] M. A. Talukder, M. M. Islam, M. A. Uddin, K. F. Hasan, S. Sharmin, S. A. Alyami, and M. A. Moni, "Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction," *Journal of big data*, vol. 11, no. 1, p. 33, 2024.

[6] S. Achleitner and A. Neupane, "Using ai to detect malicious c2 traffic," May 2021. [Online]. Available: https://unit42.paloaltonetworks.com/c2-traffic/

[7] N. Känzig, R. Meier, L. Gambazzi, V. Lenders, and L. Vanbever, "Machine learniǹg-based detection of c&c channels with a focus on the locked shields cyber defense exercise," in *2019 11th International Conference on Cyber Conflict (CyCon)*, vol. 900. IEEE, 2019, pp. 1–19.

[8] E. Macedo, "Signature-based ids for encrypted c2 traffic detection," Master's thesis, Universidade do Porto (Portugal), 2022.

[9] S. Fathima, "Validating machine learning models: A detailed overview," Jan 2024, [Online]. Accessed: May 21, 2025. [Online]. Available: https://www.markovml.com/blog/ml-model-validationtypes-of-model-validation

[10] GeeksforGeeks, "Cross validation in machine learning," May 2025, [Online]. Accessed: May 21, 2025. [Online]. Available: https://www.geeksforgeeks.org/cross-validation-machine-learning/