



Enhancing Cybersecurity with Machine Learning: Beaconing Detection in PCAP Data

Bachelor Thesis

Department of Computer Science

OST – Eastern Switzerland University of Applied Sciences

Campus Rapperswil-Jona

Spring Term 2024

Author	Anastasiia Graftceva
Advisor	Nikolaus Heners
External Expert	Ludovico Bessi
Internal Expert	Stefan Kapferer

Acknowledgements

I would like to extend my gratitude to the following individuals for their contributions to this study:

Firstly, I would like to thank Nikolaus Heners for allowing me to undertake this project, his unwavering support throughout the term, and his spot-on ideas. His belief in my abilities and selection of me for this topic, despite the competition, has been a significant motivating factor throughout this journey.

Secondly, my sincere thanks to Ludovico Bessi and Stefan Kapferer for their insightful questions, valuable inputs, and expert advice. Their critical feedback has been instrumental in refining my research and improving the quality of this paper.

Lastly, I am deeply grateful to my husband Marc for his constant support, cheer-ups, and motivation. His understanding and encouragement have provided me with the emotional strength and balance necessary to complete this work.

Abstract

This study explores the enhancement of cybersecurity through the application of machine learning techniques, specifically focusing on the detection of *beaconing* activity in network traffic (PCAP) data. PCAP, or packet capture, refers to the process of intercepting and logging traffic that passes over a computer network.

Beaconing, a communication technique and a common indicator of malicious activity requires complex multilevel detection methods due to its discreet and repetitive nature. My approach involves the development of a dual-model framework with a combination of a Histogram Gradient Booster Classifier (HGBC) and a Long Short-Term Memory (LSTM) neural network. The HGBC classifies the initial features extracted from the PCAP data, while the LSTM model further refines the detection by capturing temporal dependencies between consecutive packet flows.

The combined model achieves an accuracy rate of 99.37%, demonstrating its effectiveness in identifying beaconing patterns. This high level of accuracy illustrates the potential of a combination of machine learning and deep learning algorithms in advancing cybersecurity measures for unmasking threats in network traffic analysis.

Management Summary

Overview

This study explores the application of advanced Machine Learning (ML) and Deep Learning (DL) methods for detecting malicious patterns in network captures (PCAP), focusing specifically on beaconing. Beaconing is a communication technique where malware intermittently sends signals to an external server, known as Command and Control (C2), often to receive instructions or exfiltrate data. Detecting beaconing is challenging due to its low-frequency, regular communication patterns that blend in with legitimate traffic. This discreet behavior makes it a significant threat, as the malware can remain undetected for long periods, facilitating extensive data breaches.

Objective of the Study

The goal of this project is to develop a dual-model framework that integrates a Histogram Gradient Boosting Classifier (HGBC) and a Long Short-Term Memory (LSTM) neural network. The HGBC, an ML algorithm based on decision trees, is designed to identify presence of malicious periodic signals based on PCAP data features analysis, while the LSTM neural network detects temporal dependencies in sequential data for more accurate results.

Key Findings

- **Model Accuracy:** The dual-model framework achieves an accuracy rate of 99.37% on the tailor-made dataset
- **Performance:** The model correctly identifies beaconing patterns in files containing malicious content and accurately recognises benign files.
- **ML Classifier Efficiency:** The HGBC uses histograms to speed up training by considering unique values when looking for the best split, where the algorithm determines the best point to divide the data into subsets that are more homogeneous. By reducing the number of potential split points, the HGBC shortens the time needed for training, allowing for rapid model development even on large datasets.
- **NN Efficiency:** The LSTM neural network effectively captures temporal dependencies in the data, enhancing the detection of sequential patterns associated with beaconing. By leveraging its memory cell architecture, LSTM efficiently processes sequences, making it highly suitable for time-series analysis.
- **Data Preparation:** The process of extracting individual flows and computing relevant metrics, such as flow intervals, packet sizes, and communication frequencies, is central for beaconing detection and has a direct impact on the model's high performance. By accurately capturing and analysing these metrics, the model can effectively differentiate between normal network traffic and the subtle, periodic signals characteristic of beaconing, thereby enhancing its detection capabilities.

Approach

Data Preparation involves converting raw data into individual packet flows based on source and destination IP, port number, and protocol. Relevant metrics such as payload entropy and the mean and standard deviation of intervals between consecutive packets within a flow are computed. For Classification, the computed flow data is processed by the HGBC, which provides an initial assessment of potential malicious patterns. During temporal analysis, the output of the HGBC is used to create sequences for the LSTM neural network, which detects temporal dependencies to refine detection accuracy. Finally, a detailed detection report is generated, indicating the amount and percentage of malicious sequences or their absence in the original input data.

Implications and Recommendations

The proposed dual-model framework could be further integrated into Network Detection and Response (NDR), Security Orchestration, Automation, and Response (SOAR) systems, and other cybersecurity tools to improve pattern recognition and anomaly detection, particularly for identifying beaconing activities. For future work consider the following:

- Framework Adjustment: Tailoring the framework to specific network requirements, considering data volume, unique network and communication patterns, and particular threat detection needs.
- Data engineering: Enlargement of the training dataset that would accommodate the needs of the large and complex networks also in terms of diversity of the protocols and length of individual flows.

TABLE OF CONTENT

1. Introduction	8
1.1 What is beaconing?	8
1.2 Harm caused	9
1.3 Defense vector	10
2. Methods	13
2.1 Building a Dataset	13
2.1.1 Malicious part	15
2.1.2 Benign part	15
2.1.3 Features Computation	16
2.2 Building a classifier	18
2.3 Adding a neural network	19
2.3.1 NN-Components	20
2.4 Building a Pipeline	22
3. Results	25
3.1 Preparatory information	25
3.2 Training	26
3.2.1 Intermediate results: HGBC	26
3.2.2 Final results: HGBC-LSTM	26
3.3 Simulating beaconing	28
3.4 Wireshark Analysis	30
3.5 Inference	32
3.5.1 HGBC	32
3.5.2 HGBC-LSTM	33
4. Conclusion	35

4.1 Model Complexity	35
4.2 Limitations of the study	35
4.3 Application	36
4.4 Requirements	38
4.5 Future Steps and Recommendations	38
Academic Vocabulary	40

1. Introduction

1.1 What is beaconing?

In cybersecurity, a type of attack that aims to stay hidden while keeping operational abilities is often referred to as an "Advanced Persistent Threat"¹ (APT). These threats are a source of substantial concern due to their ability to evade detection over an extended period of time, potentially causing harm without being noticed.

A prevalent method employed by malware on compromised systems to establish communication with an external command and control (C2) server — under the possession of the adversaries² — is known as *beaconing*. This technique is characterized by its ability to execute communications that are either periodic, randomised, or triggered by specific events, thereby complicating the task of consistent detection and therefore is one of the important elements of the APT lifecycle.

Beaconing performs the transmission of signals or data packets from the infected host to the C2 server at predetermined intervals, which may vary from seconds to hours, according to the configuration set by the attackers. Following the installation of the malware on a host system, the process of beaconing becomes automated. The primary objective of this process is to maintain communication channels between the malware and the attackers, enabling the C2 server to verify the operational status of the malware. Moreover, through these periodic communications, the C2 server is able to receive data harvested by the malware from the compromised system(s), as well as dispatch new instructions or updates back to the malware, thereby directing it to execute specific tasks, download additional components, or propagate to other systems within a network (so-called "lateral movement").

Beaconing poses a significant hurdle for cybersecurity professionals for a variety of reasons, but mostly due to its secretive and consistent nature. This mechanism facilitates malware to remain in a hibernating state while simultaneously maintaining a connection with the attacker's C2 servers. The indistinct nature of this communication makes its detection problematic, as it can seamlessly integrate with legitimate network activities. This capability enables attackers to establish and prolong their presence within a compromised network for months or even years.

¹ What Is an Advanced Persistent Threat (APT)? <https://www.kaspersky.com/resource-center/definitions/advanced-persistent-threats>, accessed in April 2024

² Someone or a group that intends to perform malicious actions against other cyber resources (<https://www.sciencedirect.com/topics/computer-science/cyber-adversary#:~:text=Defining Adversary,actions against other cyber resources.>), accessed in April 2024

1.2 Harm caused

I would like to provide a few examples of APT attacks to illustrate their significance and underscore beaconing as a vital component of their life span.

The first attack that comes to mind is cyber espionage on RUAG, a Swiss company specialising in aerospace engineering and defense. In this incident, attackers, believed to be part of an APT group, penetrated RUAG's network and maintained their presence undetected for an extended period. Malware installed on the network "sent HTTP requests to transfer the data to the outside, where several layers of Command-and-Control (C&C) servers were located"³, receiving instructions and exfiltrating data in a controlled manner. This attack, known as "APT Case RUAG"⁴ was discovered in early 2016. However first indications of compromise (IOCs) were already present in logs almost 18 months prior, in September 2014, with no earlier logs available for inspection. According to the "APT Case RUAG. Technical Report", it is still unknown at which point in time adversaries penetrated the RUAG network and how much damage exactly the company sustained.

Another example involving beaconing is the "SolarWinds Orion Platform Attack" which occurred in 2020. The attack stands for one of the most intricate and widespread cyber espionage efforts yet identified, mostly due to the amount of hosts involved. Attackers infiltrated the software build system of the SolarWinds Orion Platform, the "infrastructure monitoring and management platform"⁵ of one of the largest B2B Software Developer in the US, inserting malicious code into software updates distributed to thousands of customers. This inserted backdoor, referred to as "SUNBURST," empowered the attackers to penetrate, surveil, and potentially disrupt operations across over 18,000 organisations⁶ globally, encompassing government entities and corporations. Remarkably, the malware evaded detection for over a year. By "mimicking legitimate network traffic, the attackers were able to circumvent threat detection techniques employed by SolarWinds, other private companies, and the federal government"⁷.

³ Summary: Technical Report about the Espionage Case at RUAG, https://www.ncsc.admin.ch/ncsc/en/home/dokumentation/berichte/fachberichte/technical-report_apr_case_ruag.html, accessed in March 2024

⁴ Technical Report about the Espionage Case at RUAG: https://www.ncsc.admin.ch/ncsc/en/home/dokumentation/berichte/fachberichte/technical-report_apr_case_ruag.html, accessed in March 2024

⁵ <https://www.solarwinds.com/orion-platform>, accessed in March 2024

⁶ A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack <https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack>, APRIL 16, 2021 10:05 AM ET, accessed in March 2024

⁷ New Findings From Our Investigation of SUNBURST <https://orangematter.solarwinds.com/2021/01/11/new-findings-from-our-investigation-of-sunburst/>, accessed in March 2024

These both cases illustrate how the unnoticed infiltration of an APT may cause extraordinary damage, often making it difficult to fully assess its extent.

While APTs are known for their sophisticated use of beacons, this technique is broadly applicable in the malware community due to its effectiveness in maintaining communication with infected hosts.

Adversaries are increasingly adopting beacons as a low-detection communication technique in their ever-evolving malware arsenals. This subtle method of communication is particularly effective in evading traditional security measures that rely on identifying large or anomalous data transfers. Moreover, adversaries are continuously improving their skills, leveraging advanced algorithms, machine learning, and deep learning included, to enhance the secrecy and efficiency of their malware. These sophisticated techniques, once the domain of APT groups, are now becoming more accessible to a broader range of cyber criminals. As a result, even less sophisticated attackers can deploy highly effective beaconing strategies, increasing the prevalence and threat posed by such malware in the cybersecurity landscape.

Another significant threat arises from the integration of IoT devices into modern networks, as it greatly expands the attack surface for cyber hazards. This is largely due to the inherent security weaknesses often found in these devices, including weak default credentials, lack of regular updates, inadequate encryption if applied at all, and the frequent absence of built-in security features. These vulnerabilities make IoT devices particularly susceptible to malware infections. Once compromised, these devices can serve as hidden entry points within a network, enabling attackers to maintain persistent access and control without drawing attention. This persistent access poses a substantial risk, as it can be leveraged to launch further attacks, exfiltrate data, or disrupt network operations sometimes causing astronomical costs due to damages.

1.3 Defense vector

Over the past decade, it has become obvious, that APT attacks are occurring more frequently. According to the publicly available data from the Statista portal⁸, the revenue from the APT *protection* market has been steadily increasing globally by an average of 0.8 billion USD year to year. From 2015 to 2023, the APT protection market growth represents a more than fourfold increase, indicating a strong demand for cybersecurity solutions capable of defending against advanced threats. As cyber attacks continue to evolve, the need for advanced, comprehensive protection measures becomes increasingly decisive, driving sustained investment and innovation in the cybersecurity sector. The market's projected expansion in

⁸ Revenue from advanced persistent threat (APT) protection market worldwide from 2015 to 2027 (in billion U.S. dollars), <https://www.statista.com/statistics/497945/advanced-persistent-threat-market-worldwide/>, accessed in May 2024

upcoming years underscores the importance of continued alertness and development in cybersecurity to safeguard against ever-more sophisticated threats.

As detection techniques have been in constant improvement, including more complex network monitoring tools and anomaly detection systems, the tactics used by malware, including beaconing patterns, have also evolved. Modern beaconing may vary the timing of communications to avoid creating predictable patterns or use different protocols and ports to avoid detection. Some malware even uses social media platforms, decentralised networks, or legitimate web services for beaconing to further obscure their traffic.

While beaconing presents significant risks, its very nature as an automated activity inherently involves repetitive patterns, which can be both a challenge and an advantage in cybersecurity efforts. Regular signaling, even jittered or randomised in any way, creates detectable patterns. It is possible to analyse traffic logs for such patterns, identifying anomalies that recur at unusual intervals which may not align with normal user or system activities. Advanced security tools and network monitoring solutions employ various forms of traffic analysis to detect unusual patterns. This could be achieved through for instance examining data transfers at consistent intervals which could indicate automated communication. Spikes or dips in traffic that occur with unusual regularity can be a sign of beaconing activities.

Modern cybersecurity defense strategies often incorporate ML algorithms that can learn from network behavior to detect deviations from the norm. These algorithms can be particularly effective in environments with high volumes of data, recognizing even slight deviations. Detecting beaconing however, still remains a significant challenge despite advanced strategies. As mentioned above, one key difficulty is that malware can vary its beaconing intervals or randomise packet sizes to evade pattern-based detection. This intentional randomness makes it hard to identify consistent communication patterns over longer periods due to the extensive data volumes that need to be processed and analysed. Moreover, when beaconing data is encrypted, the challenge intensifies. Encrypted communication masks the content of the data packets, making it difficult to look into without complicated, and sometimes invasive, techniques. Decryption requires resources in terms of manpower, know-how, and time, and can pose privacy concerns, that further impede the detection process⁹. These evasion strategies highlight the need for advanced detection methods that can identify subtle anomalies in network traffic, even when typical patterns are obfuscated or encrypted.

Large networks with hundreds of thousands of endpoints highlight additional challenges. Beaconing often involves low-frequency, subtle signals that can be easily overlooked among high-volume, legitimate network traffic. Network environments typically generate large volumes of traffic, making it difficult to isolate beaconing signals without tricky data

⁹ Shining a Light on Malware Beaconing, <https://blogs.blackberry.com/en/2023/03/shining-a-light-on-malware-beaconing>, accessed in April 2024

processing and filtering techniques. Malware dynamic behavior requires detection systems to be highly adaptable and continuously updated. Beaconing traffic can closely resemble legitimate network traffic, such as regular software updates or automated backups. This similarity makes it challenging to distinguish between benign and malicious activity without high accuracy. The high potential for false positives, where benign activity is misclassified as malicious, can overwhelm security analysts and reduce the effectiveness of detection systems.

As detecting beaconing usually requires temporal analysis for higher accuracy, monitoring traffic over time to identify periodic patterns becomes a necessity. On large networks, however, this task is particularly challenging due to the extensive data volumes involved and the need for long-term storage. Handling massive amounts of data necessitates data collection mechanisms and substantial storage capacity, which can be both technically demanding and financially burdensome.

A network with a dozen of thousands of servers and working stations generate terabytes of data every hour. Processing these giant data sets requires high computational power, as the system must sift through enormous amounts of network traffic to detect the subtle, low-frequency signals indicative of beaconing. This processing must be both fast and accurate to be effective, which often demands cutting-edge hardware and optimised software solutions. The financial implications of deploying such resources can be significant, potentially rendering the process cost-ineffective for many organisations.

Moreover, the sheer scale of data can lead to delays in analysis, reducing the timeliness of threat detection and response. Therefore, balancing the need for thorough temporal analysis with the practical limitations of computational resources and budget constraints is a key challenge in the effective detection of beaconing activity on large networks.

2. Methods

2.1 Building a Dataset

To detect malicious software, particularly the type that maintains regular communication with C2 servers through keep-alive and beaconing activities, it is required to perform a detailed examination of network traffic first. It is important, however, to distinguish between traffic data captured in automated, IoT-enhanced, or non-automated environments, to predefine *normality* for each environmental type as their characteristics may drastically differ.

Automated setups often show predictable traffic patterns due to automated tasks. Environments with IoT devices for instance may demonstrate a combination of high-volume and continuous or near-continuous communication with smaller packet sizes, which carry sensor readings or status updates. In contrast, non-automated setups rely on human-initiated communication, resulting in less regular and more varied traffic patterns, as they primarily involve texts, images, and multimedia content.

On the other hand, the duration of traffic monitoring significantly impacts data analysis. Longer monitoring periods yield more data, enhancing the ability to identify patterns indicative of abnormal activity. Typically, it is recommended to monitor traffic for at least several days to capture a broad spectrum of activities and likely potential irregularities. Additionally, measuring network traffic volume, such as packets per minute, is another important point. Large volumes of data transfer could indicate routine operations, which might be considered normal in certain contexts or networks, yet it could also potentially be indicative of malicious behavior. Conversely, low volumes with intermittent spikes might suggest beaconing behavior as malware would communicate at predefined intervals. It is important to note, however, that 'predefined' does not necessarily mean 'every X minutes', as intervals could appear more random, especially if jitter or any other form of randomisation is used.

Analysis of PCAP files involves identifying patterns and anomalies that deviate from the *typical* behavior of network traffic. Protocols like DNS (Domain Name System) and HTTP (Hypertext Transfer Protocol) hold significant relevance in the detection of beaconing behavior due to their employment in both legitimate and malicious network communications.

Cyber adversaries frequently exploit DNS to secretly establish communications with C2 servers, thereby avoiding detection. This exploitation is facilitated by the use of DNS queries and responses to create a camouflaged bidirectional communication channel, enabling data exfiltration and the reception of commands from C2 servers. This method, often referred to as DNS tunneling, exploits the fact that DNS traffic often bypasses firewalls and can carry small data payloads. It poses detection challenges, as these packets blend seamlessly with legitimate

DNS traffic, thus requiring advanced analysis techniques to differentiate between harmless and malicious activities.

It is worth mentioning that certain malware variants may employ so-called Domain Generation Algorithms¹⁰ (DGAs) to dynamically generate a large number of domain names for communication with C2. These domain names often seem random or nonsensical, making it difficult for security systems to predict or block them. By generating domain names in real time, malware can establish communication channels with C2 servers without using static, easily identifiable domain names. Monitoring of DNS request patterns for irregularities may help in the identification of such behavior. Notably, malicious DNS queries may not depend on high traffic volumes to achieve their objectives, rendering volume-based detection strategies insufficient. Instead, the periodic, regular nature of these queries —characteristic of beaconing — emerges as a distinct indicator when observed over a duration, accentuating the need for temporal analysis to detect and mitigate such threats effectively.

The payload carried by DNS queries is characteristically unencrypted, presenting an opportunity for malicious actors to embed malicious code within these payloads. Despite the potential for encryption to conceal such malicious content, its implementation can inadvertently trigger suspicion from Intrusion Detection Systems (IDS) due to the distinctive even distribution of bytes in the payload, a common trait of encrypted data. Consequently, this aspect of DNS traffic becomes a critical vector for analysis, particularly the examination of payload randomness.

Analysing the entropy, or a measure for randomness, of the DNS payload provides valuable insights into the nature of the traffic. If high entropy is often indicative of encrypted data, reflecting a random distribution of bytes, conversely, low entropy suggests a more predictable and potentially structured payload, which could be indicative of regular, unencrypted data or, in the context of cybersecurity threats, a sign of malicious activity.

Given the unencrypted state of typical DNS traffic, the detection of payloads with unusually low entropy becomes a potential indicator of malicious intent. Malicious actors aiming to avoid detection might opt for encoding mechanisms that do not exhibit the tell-tale signs of encryption, such as high entropy, to blend in with legitimate DNS queries. Therefore, the analysis of payload entropy in DNS traffic emerges as a strategic method for identifying anomalies that deviate from expected patterns, providing a basis for identifying and investigating potential security breaches. This approach underscores the nuanced balance between the tactics employed by adversaries and the sophisticated detection methodologies required to identify and counteract malicious activities within network communications.

In the course of this research, I collected my own data alongside two open-source datasets to build a model for proofing the concept. Due to limited resources, I opted for a

¹⁰ Dynamic Resolution: Domain Generation Algorithms, <https://attack.mitre.org/techniques/T1568/002/>, accessed in April 2024

small dataset that is good enough to show the framework capabilities. Obviously, it does not reflect the real-world scenario, as the dataset of such manageable size is a fraction of a production environment.

2.1.1 Malicious part

The first open-source dataset, TII-SSRC-23 Dataset¹¹, contains both raw (PCAP) and processed (CSV) data, systematically divided into malicious and benign categories. The malicious segment of the dataset is organised into folders named after the specific types of attacks they represent, such as Mirai¹², Denial of Service (DoS), and Brute Force, among others. Within the scope of malicious traffic analysis, my specific attention is directed toward the Mirai-botnet DNS and HTTP PCAP files, given their relevance to the research objectives. What makes Mirai (and botnets in general) relevant to my interest in C2 communications is that the compromised devices are controlled through a C2 server. The botnet receives instructions from attackers via these C2 servers, coordinating the bots' actions, including targeting and executing DoS attacks. This is a clear example of malware that relies on C2 communications for its functioning.

2.1.2 Benign part

The second open-source dataset, MQTTset¹³, is composed of data from eight IoT sensors operating on the MQTT¹⁴ (Message Queuing Telemetry Transport) protocol. These sensors collect data across a variety of parameters, including temperature, light, humidity, CO-Gas, motion, smoke, door status, and fan operation, each with distinct communication intervals reflective of the varied behavior patterns among the sensors. The inclusion of MQTTset complements the TII-SSRC-23 by offering insights into periodic communication behavior typical of genuine IoT devices. I also partially include a benign portion of TII-SSRC-23. It is organised by the nature of the traffic, such as audio, text, and background traffic, thereby providing a diversified range of normal network activities for comparison and analysis.

During the collection of benign data, I noted several key observations. Benign network flows tend to have a longer duration, particularly in highly automated environments equipped with IoT devices. Estimating the packets-to-flow ratio accurately is challenging, as this metric varies depending on numerous factors, such as network configuration, the types of devices

¹¹ <https://www.kaggle.com/datasets/daniaherzalla/tii-ssrc-23>, accessed in March 2024

¹² What is Mirai? <https://www.cloudflare.com/en-gb/learning/ddos/glossary/mirai-botnet/>, accessed in March 2024

¹³ <https://www.kaggle.com/datasets/cnrieit/mqttset?select=requirements.txt>, accessed in March 2024

¹⁴ <https://mqtt.org/>, accessed in March 2024

employed, and traffic volume. Consequently, gathering a substantial volume of benign flows to balance effectively against the malicious data proved to be challenging.

However, it is important to underline that, despite some types of automated traffic having extremely long flows (e.g., a few hundred flows per several million packets), this should not be taken as an indicator that raw files with an extremely low flow-to-packets ratio are necessarily benign. The complexity and variability of network traffic patterns mean that both benign and malicious files can exhibit a wide range of flow characteristics. Thus, comprehensive analysis and context-specific understanding are central for accurate classification.

To obtain the missing portion of benign data, mostly to balance my malicious part, I conducted a packet capture on my own home network, which includes a few IoT devices such as a CCTV camera, a robot vacuum cleaner, a few garden sensors, and wifi-enabled pet accessories. The capture was performed continuously over an 8-hour period, encompassing a variety of protocols and capturing the typical traffic generated during a daily routine. This capture aimed to reflect a realistic and diverse set of benign network flows, enhancing the dataset's representativeness and balance against the malicious data.

2.1.3 Features Computation

After having collected the required PCAP files, I processed them as follows:

1. Flow Extraction

To analyse the network traffic, I extracted individual flows using the 5-tuple identifiers: source IP, destination IP, source port, destination port, and protocol. Here's a detailed explanation of each component and the extraction process:

Source IP is the IP address of the device that initiated the flow. It helps in identifying the origin of the traffic within the network. Destination IP is the IP address of the device that is the target of the flow. It indicates where the traffic is directed. Source port is a number assigned to the session by the originating device. It helps in differentiating multiple flows coming from the same source IP. Destination Port is a number assigned to the session by the receiving device. It helps in directing the incoming traffic to the appropriate service or application on the target device. Protocol refers to the network protocol used for the communication (e.g., TCP, UDP, ICMP, etc). It provides an understanding of the nature of the traffic and how the data is being transmitted.

2. Feature Calculation

Features in a dataset are individual measurable characteristics or properties, which are used as input variables by models to make classifications or predictions. For each flow, I computed the following features:

- *mean_interval*: Specifies the average time interval between packets in a flow. This metric provides insights into the regularity and rhythm of communication patterns. A non-zero mean might indicate periodic communication.
- *std_dev_interval*: Specifies the standard deviation of the time interval between packets, indicating variability in the intervals. A very low or close-to-zero standard deviation suggests an automated process.
- *mean_payload_entropy*: Specifies the entropy of the payload data, measuring the randomness or unpredictability of the data content in packets. High entropy often indicates encrypted or compressed data, while low entropy (1-4 bits) could suggest structured data or potentially malicious payloads that avoid encryption to remain undetected.
- *mean_packet_size*: Specifies the average size of packets within the flow.
- *std_dev_packet_size*: Specifies the standard deviation of packet sizes, showing how much packet size varies within the flow. Low values of standard deviation indicate that the packets are of similar size, which could be a sign of automated check-ins.
- *label*: Serves as a classification marker for the flow, where 1 indicates malicious and 0 indicates benign.
- *mean_frequencies*: Specifies the average frequency observed in the flow. Beacons might show a strong dominant frequency (high mean value) that corresponds to the periodicity of the beacon signals.
- *std_dev_frequencies*: Specifies the standard deviation of the frequencies within a flow. I would expect less variability in the frequencies, reflecting the regularity of signal intervals. A lower standard deviation suggests that most of the packet intervals are around a few dominant frequencies, reinforcing the possibility of beaconing.

Frequencies were calculated using the Fourier Transform, a mathematical technique that transforms a signal from its original domain (often time or space) into a representation in the frequency domain. The original signal consists of the time intervals between consecutive packets. These intervals represent how often packets are sent over a network. After applying FFT (Fast Fourier Transform), the signal is represented as a series of frequencies that show how often certain patterns of packet timing repeat. Each value in the result of the FFT corresponds to a specific frequency component of the packet timing. These components indicate the presence of periodic patterns within the packet intervals. Each frequency value corresponds to a different rate at which these patterns repeat, providing insight into the regularity and potential periodicity of the network traffic.

Low frequencies often represent slower, more dominant patterns, possibly indicating regular or periodic traffic patterns, while high frequencies can indicate more rapid changes in packet intervals, potentially pointing to bursts of traffic or irregular activities.

2.2 Building a classifier

HGB is a machine learning method based on the gradient boosting algorithm. This algorithm builds decision trees sequentially, one at a time, where each new tree is built to predict residuals (errors) of previously built trees combined. The key components that define how the algorithm optimises, learns, and integrates results from previous predictions are Loss Function, Weak Lerner, and Additive Model.

Loss Function quantifies the difference between the predicted values and the actual values. It essentially measures the error of a model on a dataset. After each round of boosting, the algorithm evaluates the loss and uses its gradient (i.e., the first derivative of the loss function) to determine the direction in which to update the model predictions.

The Loss function used in my model is binary cross-entropy loss (BCE). For a single instance (data point or sequence) loss the formula would be the following:

$$L(y, \hat{y}) = - [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})],$$

where y is the true label (0 or 1) and \hat{y} is the predicted probability of the class with label 1 (malicious).

There are many loss functions that can be used for training a classifier. BCE in my case is well-suited for binary classification tasks, enabling the model to quantify the difference between predicted probabilities and actual binary labels effectively. Since HGB builds decision trees sequentially, during training it minimises the loss function over each individual prediction.

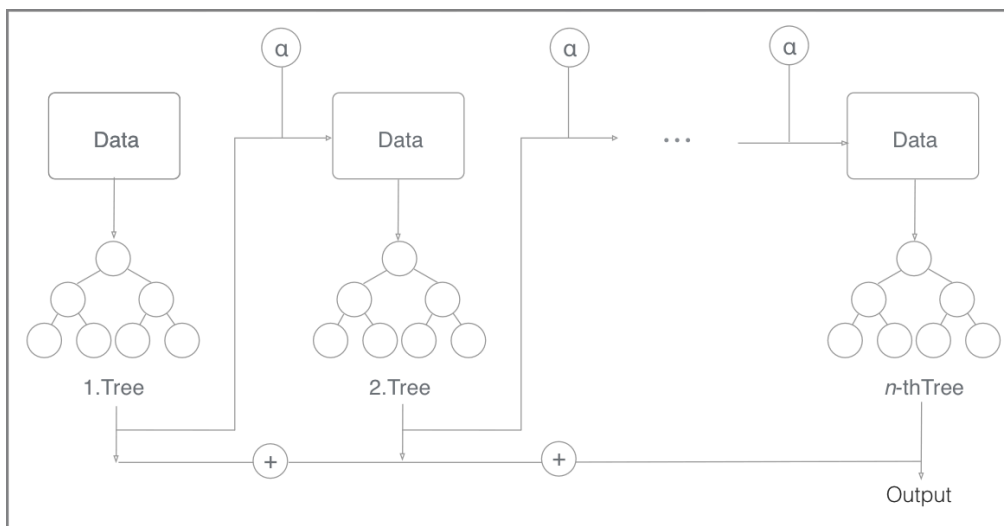


Fig.1 Gradient Booster diagram

Weak Learner is a simple model that does slightly better than random guessing but is generally not very accurate by itself. In gradient boosting, decision trees are commonly used as weak learners. The fundamental idea behind boosting is to combine multiple weak learners

to create a strong learner. Each weak learner is trained to correct the errors of the preceding ones. In each iteration, a new weak learner focuses on the instances that had the most error for the previous models. Weak learners are typically shallow trees, sometimes only one level deep. They have high bias but low variance. By combining multiple weak learners, boosting aims to maintain low variance while reducing bias, which helps achieve a balance between accuracy and generalisation (so-called 'bias-variance tradeoff'¹⁵).

Additive model in boosting refers to the way the final strong learner is constructed, which occurs by sequentially adding weak learners to one another, rather than modifying existing ones. Each new weak learner in the sequence is fitted to the residual errors made by the previous model in the sequence. In mathematical terms, if $F_m(x)$ is the model obtained after the m -th step, then the model at step $m + 1$ is $F_{m+1}(x) = F_m(x) + \alpha \cdot h(x)$, where $h(x)$ is the new tree and α is the learning rate, a hyper-parameter that scales the contribution of each weak learner. It controls how 'fast' the model learns, with smaller values generally leading to more robust models at the cost of requiring more trees.

HGB efficiency is built on the fact, that it incorporates histograms¹⁶ to speed up training by evaluating unique values for optimal splits, thereby minimising split points and accelerating training while conserving memory. Moreover, HGB operates effectively without data normalisation and exhibits resilience to outliers in the dataset.

2.3 Adding a neural network

As already stated, beaconing activities involve regular and predictable communication patterns. Detecting this involves identifying communications patterns amidst other network traffic, which can vary greatly in volume *and timing*.

By focusing on the timing aspect of network communications, it is possible to more accurately distinguish between legitimate automated traffic (like regular updates or backups) and malicious beaconing activity. This specificity helps in reducing false positives, where benign activities are incorrectly flagged as malicious.

Timing data becomes even more powerful when correlated with other metrics like payload entropy, packet size, and traffic frequency. This multi-faceted approach amplifies the accuracy of detecting beaconing activity by confirming suspicions that arise from timing analysis with other indicators of compromise.

The second stage of the research augments the non-temporal feature set, specifically focusing on time series analysis of the communication patterns. This involves adjusting the

¹⁵ Understanding the Bias-Variance Tradeoff, <https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229>, accessed in June 2024

¹⁶ Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," NeurIPS, 2017.

existing pipeline to accommodate the additional complexity of temporal data, which requires sequence modeling to effectively capture and utilise the dynamic nature of the data.

By incorporating both non-temporal and temporal feature analysis, I aim to achieve a more nuanced understanding and detection capability for identifying malicious network activities, thereby enhancing the overall effectiveness of the cybersecurity measures deployed to hold out against cyber espionage.

2.3.1 NN-Components

Before explaining the architecture of LSTM I would like to give a short overview of the most important components used in a neural network.

Layers are the fundamental structural units in a NN architecture that process input data and transform it through various computations. Each layer typically consists of nodes (or neurons), which are connected by weights and biases that are adjusted during training.

Input layer is the entry point for data in the neural network. Each input layer node corresponds to a feature in the input data. There are no computations performed in this layer; it is used to pass the input data to the next layer.

Hidden layers perform the majority of the computations required by the network. There are several types of hidden layers, but in my research, I focus on the combination of LSTM and Dense Layer:

Long Short-Term Memory, or LSTM, is a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in capturing long-term dependencies in sequential data. LSTMs are composed of repeating modules of neural network layers. Each module has three main components:

Forget Gate: This gate determines what information from the previous time step should be discarded or "forgotten". It takes as input the previous hidden state $h_{(t-1)}$ and the current input X_t , passes them through a *sigmoid* activation function, and outputs a vector of numbers between 0 and 1. A value of 0 means "completely forget this information", while a value of 1 means "remember this information".

Input Gate: This gate decides what new information should be stored in the cell state. It consists of two parts: a *sigmoid* layer that decides which values to update, and a *tanh* layer that creates a vector of new candidate values that could be added to the state. The *sigmoid* layer outputs numbers between 0 and 1, determining which values to update, and the *tanh* layer outputs numbers between -1 and 1, representing new candidate values.

Output Gate: This gate controls what information from the current cell state should be output to the next hidden state. It takes as input the previous hidden state $h_{(t-1)}$ and the current input X_t and passes them through a *sigmoid* activation function. The cell state C_{t-1} is then passed through a *tanh* function to squish the values between -1 and 1. The output of the

sigmoid gate is multiplied element-wise with the output of the *tanh* gate, resulting in the final output h_t for the current time step.

The cell state represents the long-term memory of the network. It runs straight down the entire chain of LSTM units through time, with only minor linear interactions. The cell state can be seen as the conveyor belt that carries information across different time steps. It is updated and modified by the forget gate, input gate, and output gate, allowing the LSTM to selectively add or remove information from the cell state at each time step.

The hidden state is a filtered version of the cell state that only contains relevant information for the current prediction task. It can be thought of as the "short-term memory" of the network. The hidden state is computed based on the cell state but is passed through the output gate, which selectively exposes certain parts of the cell state while suppressing others. The hidden state carries information that the network has deemed relevant for the current prediction or classification task.

Dense Layer is a type of neural network layer where every input node is connected to every output node. The "dense" in its name refers to the complete interconnection of nodes. In simpler terms, in a Dense layer, the outputs from the previous layer are connected to every neuron in the current layer.

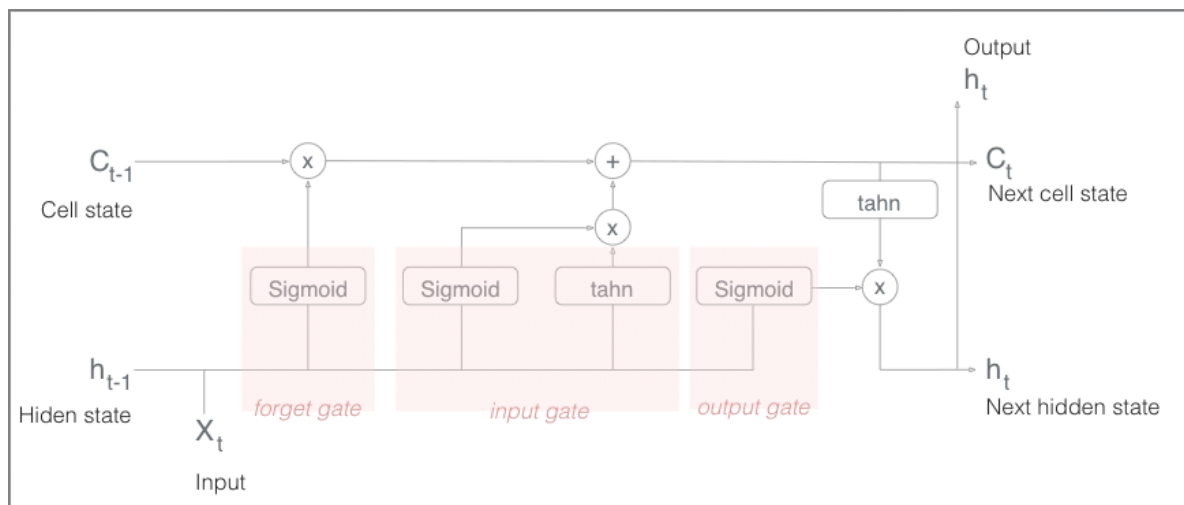


Fig.2 LSTM diagram

Each neuron in a Dense layer performs a linear transformation on the inputs it receives, which can be expressed as a dot product of the inputs and the weights plus a bias term. After the linear transformation, an activation function is applied to each output. This activation function can be nonlinear, such as *sigmoid*, *tanh*, or *ReLU* (Rectified Linear Unit). The activation function is a mathematical operation applied to the output of a neuron in a neural network, determining whether it should be activated or not, influencing the information flow through the network. It introduces nonlinearity, enabling the network to learn complex patterns and relationships in data. There are many activation functions available, I will focus on the two, most relevant for my model:

1. *Sigmoid* Activation Function: The sigmoid function, also known as the logistic function, is a smooth, S-shaped curve that squashes input values between 0 and 1. It's given

by the formula:
$$f(x) = \frac{1}{1 + e^{-x}}$$

It's commonly used in the output layer of binary classification models to produce probabilities.

2. *Tanh* is another S-shaped curve that outputs values between -1 and 1. It's given by

the formula:
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

LSTM networks use both *sigmoid* and *tanh* activation functions in their layers to manage and regulate the flow of information. The *sigmoid* function is used for gating mechanisms to control the flow of information, while the *tanh* function is used to ensure that the values added to the cell state and output are properly scaled and can represent both positive and negative information. This combination has its complementary purpose that allows LSTMs to effectively capture long-term dependencies and manage information flow through the NN.

2.4 Building a Pipeline

To build a pipeline I first convert the raw data of traffic captured in a PCAP file into a CSV data frame which I then feed into my classifier. Raw data does not tell me much about the probable presence of beaconing activity, it is just a collection of sequential packets within a network recorded over a certain period of time. Therefore the first step in the preparation of data is the computation of the metrics I discussed in *Chapter 2.1.3 Features Computation*.

After computing all necessary properties and saving them in a CSV file, I import the pre-calculated CSV data into a Pandas DataFrame. I then preprocess the data to remove all NaN values, which result from the computation of means for single-packet flows. Although some machine learning classifiers can effectively handle NaN values, their presence might pose challenges when used in NN.

The total number of NaNs eliminated from my dataset was just under 10%, which is unlikely to significantly affect the dataset's integrity. Nevertheless, to ensure this step does not damage the classifier's accuracy, I retained both versions of the dataset for further experimentation with the model. Subsequent experiments showed that eliminating single-packet flows from the dataset did not result in any drop in accuracy.

Next, I divide the dataset for training, testing, and validation purposes. Since I have a two-stage approach where each model (HGBC and LSTM) should be trained separately, it is necessary to avoid any so-called data leakage, which happens when the data used to train the classifier overlaps with the data used to validate or test either the classifier or the LSTM.

I use the following strategy: first, I divide the entire dataset into a Development Set (DS) and a Final Test Set (FTS). The DS is used to train the classifier and contains 60% of

the data. The remaining 40% is set aside for the final testing of the complete pipeline. Next, the DS is further split into Training (70%), Test (15%), and Validation (15%) subsets. This split ensures that I can train, tune, and test the classifier without any data overlap with the LSTM or the final test phase. The Training Set is used to train the classifier, the Validation Set provides insights for tuning hyper-parameters to improve model performance, and the Test Set is used to evaluate the classifier independently from the LSTM.

After training the HGBC and validating its performance, I use its output as input for the LSTM model. I define two variables: `y_pred`, which contains the predicted labels for the final test dataset `X_final_test`, and `y_proba`, which extracts the probability estimates for the malicious class from the test predictions. This is useful for threshold-based classification and further analysis.

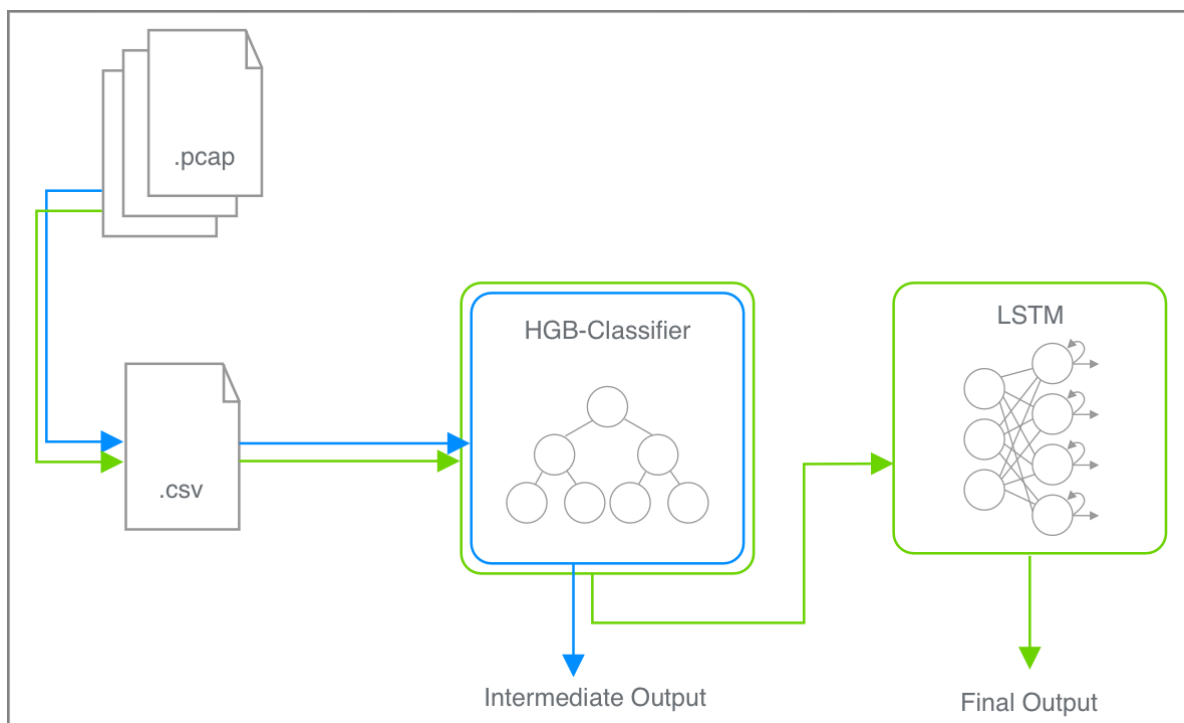


Fig.3 HGBC-LSTM Pipeline

Detecting beaconing activity using `y_pred` (class labels) might be more straightforward. However, using `y_proba` (probability outputs) provides a richer input for the LSTM, as it includes information about the model's confidence in each prediction. This additional information can help the LSTM better capture subtle changes or uncertainties in the communication patterns indicative of beaconing.

Using probabilities offers several additional benefits, particularly in providing a sense of confidence about the predictions. This is especially useful in borderline cases, where the distinction between categories is not clear-cut. Probabilities give a quantitative measure of confidence. For example, if a model predicts an outcome with a probability of 0.95, it

indicates a high level of confidence in that prediction. Conversely, a probability of 0.55 suggests low confidence, indicating that the model finds the decision to be a close call.

By observing how probabilities change over time, the LSTM can better identify patterns suggesting beaconing, such as periodic spikes in the probability of the malicious class.

At the next step, I construct sequences from probability estimates and associate them with the actual labels, which serve as input for my NN Model. A sequence is an ordered set of data points grouped together as a single input feature to the model. These data points are related and arranged in a specific order. In my case, these are values recorded over consecutive time intervals or a series of events that occur one after another.

My given sequence length is set to 30, meaning the LSTM model will look at 30 consecutive flow classifications at a time, moving forward by 1 flow at each iteration, similar to a sliding window. However, a file might have fewer flows than 30. In such cases, the sliding window with a sequence length of 30 would be too large for the file. To handle this, I use padding. This approach allows files with fewer flows to be padded with zeros to match the required sequence length, ensuring that all flows are of uniform length for input into the LSTM model.

The sequences are used as input to the LSTM, which attempts to identify patterns in these sequences. For example, it might learn that certain combinations of benign and malicious flows suggest a larger security threat.

The importance of sequence data lies in the temporal or logical relationship between data points. This relationship can provide context that helps a model make more accurate predictions than if the data points were considered independently. In the context of beaconing detection, a sequence should ideally cover enough packets to potentially include *several cycles* of beaconing activity to capture the periodicity. The length of the sequence is a critical attribute and needs to be determined empirically. There is also a computational constraint, where longer sequences can increase both the model's complexity and the computational power required to train and run the model.

Since the number of flows per PCAP file may vary significantly, the sequence length needs to be flexible enough to handle files on the smaller end without losing too much information on larger files. Therefore I consider the typical dynamics within a flow. If key patterns or malicious behavior manifest over shorter sequences within a flow, a shorter sequence length may capture these effectively. Conversely, if malicious activities are spread out or require a broader context, longer sequences might be necessary.

Once the LSTM is trained with HGBC output, I evaluate its performance on new unseen data. This process helps in understanding the accuracy of the trained LSTM model when applied to data it has not encountered before. This final evaluation tests the entire model pipeline — from initial classification to sequential processing by the LSTM — reflecting application performance similar to a real-world case.

3. Results

3.1 Preparatory information

To sum up, for this study I have developed a two-stage HGBC-LSTM framework to predict malicious flows indicative of beaconing activity based on the features discussed in Chapter 2.1.3 *Features Computation* and time-series data collected from a variety of sources (as discussed earlier in Chapter 2.1 *Building a Dataset*).

The dataset comprises 12'447 instances, split into 60% development, and 40% final test sets, while development is further split into 70% train, 15% validation, and 15% testing segments. This split technique provides 42% of the total samples for training purposes which corresponds to 5'228 samples, and 9% of the total (1'120 samples) for validation and testing each. The final test set (4'979 samples) is used for testing the entire pipeline.

```
Test Set Accuracy: 0.9866190900981266
Test Set Classification Report:
      precision    recall  f1-score   support

     0       0.99      0.98      0.98       471
     1       0.98      0.99      0.99       650

 accuracy                0.99       1121
 macro avg              0.99      0.99      0.99       1121
 weighted avg           0.99      0.99      0.99       1121

Validation Set Accuracy: 0.9910714285714286
Validation Set Classification Report:
      precision    recall  f1-score   support

     0       0.99      0.99      0.99       468
     1       0.99      0.99      0.99       652

 accuracy                0.99       1120
 macro avg              0.99      0.99      0.99       1120
 weighted avg           0.99      0.99      0.99       1120

Test Set Length, X, y: 1121 1121
```

Fig.4 Accuracy report on HGBC

The LSTM model was configured with two hidden layers with 50 units each and trained using the Adam optimiser over 10 epochs. Model performance was primarily evaluated using accuracy and loss metrics, chosen to balance the need for predictive power and interpretability. I ran my model on an Apple M1 Max GPU using Python 3.9, Sci-kit Learn 1.3.1, and Keras 2.14.0. My results should be reproducible within similar environments.

3.2 Training

3.2.1 Intermediate results: HGBC

After training the HGB-classifier I have obtained the following results: *Fig. 4* shows a standard accuracy report with some important metrics, where *precision* shows the accuracy of positive predictions; *recall* shows how well the model identifies actual positives; and *f1-score* shows a ratio of precision and recall combined. Support shows a number of flows identified per each class.

These results suggest that the model is performing very well and generalising effectively across unseen data. Both the test (98.66%) and validation (99.11%) accuracies are excellent, indicating that the classifier is effective at distinguishing between the two classes.

The precision and recall values are nearly perfect, which means that the classifier not only labels a high proportion of positive samples correctly (precision) but also labels the most positive samples correctly identified by the classifier (recall).

The f1-scores, which balance precision and recall, are close to 1, suggesting a balanced classifier that doesn't overly favour precision at the expense of recall or vice versa.

The training speed of the classifier on a subset of 5,228 samples is 0.654 seconds. Several factors contribute to this quick training time. The dataset is relatively small for modern machine learning tasks, which inherently requires less computation. The features used are numeric and relatively straightforward, which simplifies the processing. The HGBC is designed for speed and is particularly suitable for larger datasets, making it efficient even with smaller datasets. Additionally, the data is clean and the features are preprocessed efficiently, which reduces overhead during training. The hardware used also plays its role in speeding up the training process.

3.2.2 Final results: HGBC-LSTM

The HGBC-LSTM model demonstrated a slightly better accuracy of 99.35% and a loss of 2.8% over 10 epochs if compared with the HGBC on a standalone basis.

Initially, the loss starts at 0.5479 and rapidly drops to 0.2196 by the second epoch, indicating significant learning between the first and second epochs. Afterward, the loss continues to decrease at a diminishing rate until it stabilises around 0.03. This pattern suggests that most learning occurs in the early epochs, with the model quickly converging to an optimal solution.

The accuracy, starting from the initial epoch, quickly reaches approximately 99.39%, stabilising around 99.35% to 99.39% for the subsequent epochs. This high level of accuracy

indicates that the model is highly effective in correctly classifying the given sequences according to the dataset.

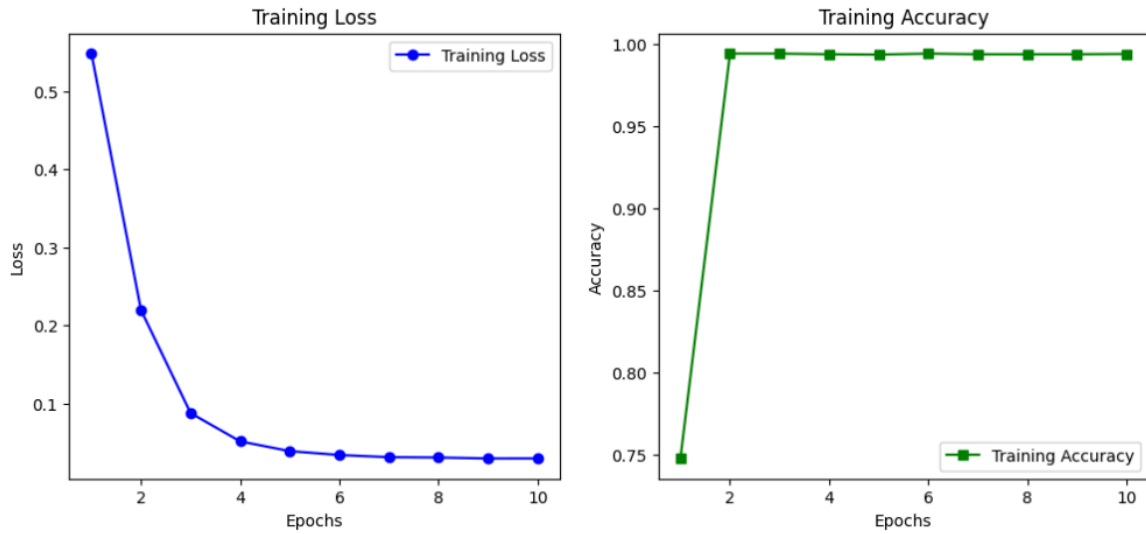


Fig. 5 Training Loss and Accuracy Plots

The model reaches low loss and high accuracy relatively quickly, which is a positive sign of its effectiveness. However, this could also suggest that the task may be somewhat straightforward for the model given its architecture, or that the model is extensively benefiting from easily learnable patterns in the data.

```

Training Summary:
Total epochs: 10

Epoch-wise Loss and Accuracy:
Epoch   Loss           Accuracy
1        0.5479         0.7475
2        0.2196         0.9939
3        0.0879         0.9939
4        0.0517         0.9935
5        0.0392         0.9933
6        0.0343         0.9939
7        0.0315         0.9935
8        0.0311         0.9935
9        0.0298         0.9935
10       0.0299         0.9937

Final Metrics:
Loss: 0.0299
Accuracy: 0.9937
    
```

Fig.6 HGBC-LSTM training results

The stabilisation of loss and accuracy in later epochs indicates that continuing training beyond 10 epochs might not yield significant improvements under the current configuration

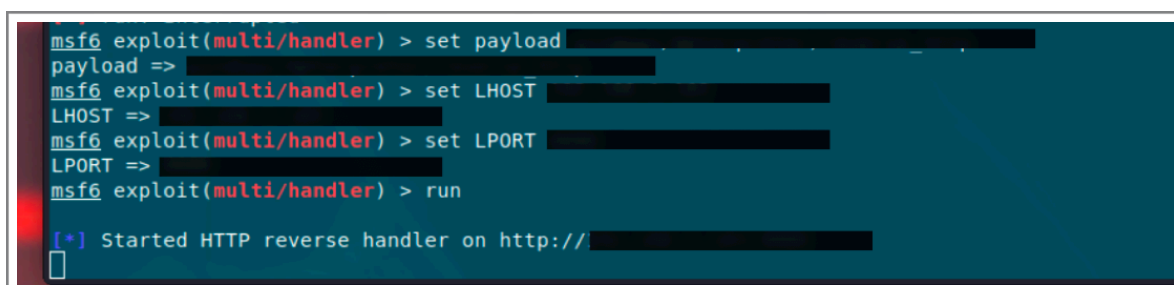
and data. This suggests that the model's capacity and the complexity of the task are well-matched. The model's accuracy on the final test set is very close to the accuracy observed during training, suggesting that the model generalises well to new, unseen data. This also indicates that the model is not significantly overfitting.

As described above, both the HGBC and HGBC-LSTM models demonstrate excellent performance. The HGBC-LSTM combination shows just a slight improvement over the HGBC alone. This slight improvement highlights the effectiveness of integrating the two models' architecture, as the HGBC provides solid initial predictions, and the LSTM leverages temporal patterns and probability estimates for further refinement. The combined approach allows the system to capture both static and dynamic features of the data, resulting in a more nuanced and accurate classification. This synergy between machine learning and neural networks underscores the potential benefits of hybrid models in complex pattern recognition tasks.

3.3 Simulating beaconing

To test my model on previously unseen data, I need to generate beaconing traffic, inject it into my network, and capture the resulting data for the model evaluation.

This process involves several steps and can be accomplished using various tools and techniques. For my setup, I use Kookarai¹⁷, a penetration testing Linux virtual machine, which I have already preinstalled on one of my laptops.



```
msf6 exploit(multi/handler) > set payload [REDACTED]
payload => [REDACTED]
msf6 exploit(multi/handler) > set LHOST [REDACTED]
LHOST => [REDACTED]
msf6 exploit(multi/handler) > set LPORT [REDACTED]
LPORT => [REDACTED]
msf6 exploit(multi/handler) > run

[*] Started HTTP reverse handler on http://[REDACTED]
```

Fig.7 Simulating C2 server with Metasploit

Since the Kookarai Virtual Machine (VM) comes equipped with several tools for cybersecurity professionals, I make use of the Metasploit¹⁸ penetration testing framework for beaconing generation. The Metasploit framework is a powerful tool that can cause harm if not used diligently; for ethical reasons, I do not disclose the exact commands in this paper.

The idea behind the beaconing simulation is similar to what I discussed in section *Introduction 1.1 What is Beaconing?* To simulate a C2 server I first configure a Metasploit handler on my VM. I use a reverse HTTP payload, which creates a reverse connection back to

¹⁷ Kookarai <https://kookarai.idocker.hacking-lab.com/>, accessed in May, 2024

¹⁸ <https://www.metasploit.com/>, accessed in May, 2024

the handler. Here LHOST is set to my VM IP address, and LPORT is the connection port that the handler will be listening on. The handler's role is to manage the connection initiated by the payload, which in a real-world attack would be the compromised system calling back to the attacker's server.

In a real-world scenario, a malicious actor would then create a payload designed to be installed on the victim's system using various techniques, such as ARP spoofing¹⁹, phishing, or exploiting software vulnerabilities. Once installed, this payload would establish a connection to the C2 server, allowing the attacker to control the victim's system remotely. However, for testing purposes, I take a simpler approach to avoid the complexity and ethical concerns of using real malware. I write a Python script that simulates the beaconing process. This script sends periodic HTTP GET requests to my VM host every 30 seconds. It is also designed to print the status code of the response to confirm that the beacon check-in has been sent, where a status code of 200 would indicate a successful transmission. By running this script on a separate laptop, I can simulate the behavior of an infected machine communicating with a C2 server.

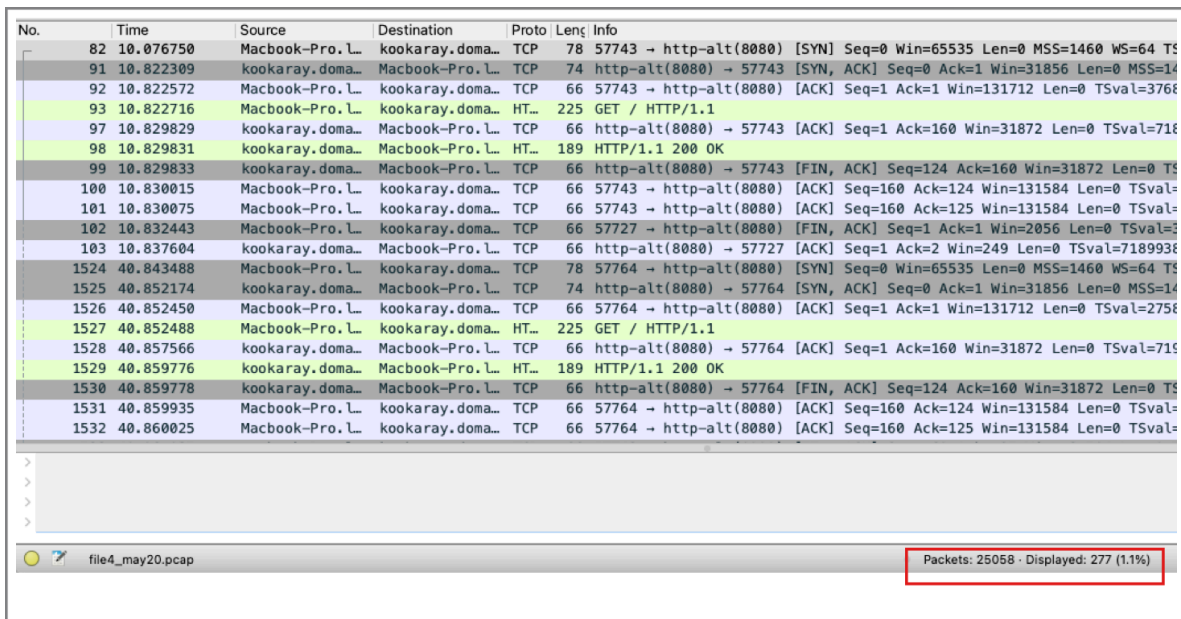


Fig.8 Filtered network capture

While my script is running, I capture traffic on my "victim" laptop using Wireshark. Due to time constraints, I do not plan to capture traffic over several days, as my script is made to send a signal every 30 seconds. I only need to generate enough data to run my model (despite that the model's design allows a low number of flows per file I would not try to do it with less than 30 flows due to the necessity to recognize patterns). After some time, I obtain a PCAP file containing a total of 25,058 packets. To ensure that my network traffic capture includes

¹⁹ "A hacker sends fake ARP packets that link an attacker's MAC address with an IP of a computer already on the LAN.", <https://www.okta.com/identity-101/arp-poisoning/>, accessed in June, 2024

sufficient relevant data, I filter out only packets containing the malicious payload, resulting in 277 packets. Within these filtered packets, I observe communication with the VM and status code 200 for the GET requests, confirming that the beaoning generation was successfully captured as expected.

It is important to note that the beaoning simulation used in this study is intentionally simple and direct, designed purely for testing purposes. In a real-world scenario, such communication would be much harder to detect by merely inspecting the filtered PCAP file. The straightforward nature of this simulation does not capture the complexity employed in actual cyberattacks. Real-world beaoning often uses various obfuscation techniques, such as encrypted traffic, randomised intervals, and blending in with legitimate traffic, substantially complicating detection efforts.

Moreover, real-world penetration testing should incorporate a more sophisticated beaoning simulation. Such simulations should closely mimic actual attack scenarios, including the use of advanced evasion techniques, to thoroughly test and validate the model's effectiveness. By simulating attacks that are as realistic as possible, researchers can better assess the model's capabilities and ensure it is solid enough to handle the nuanced and intricate nature of genuine cyber threats.

3.4 Wireshark Analysis

An important preliminary step before performing inference testing on unseen data is an analysis of PCAP files. This analysis may be conducted in many ways, using tools like Wireshark or tcpdump, or writing scripts with help of libraries²⁰ for traffic capture analysis. This initial step helps to understand the data and provides an overview of the patterns of network traffic, which has its critical importance before inputting this data into an HGBC-LSTM model for further analysis.

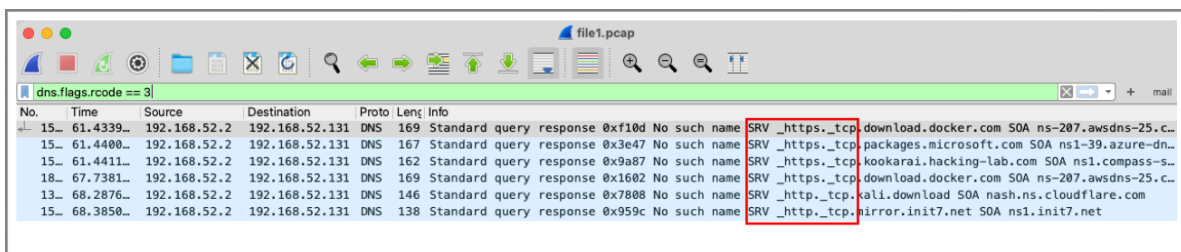


Fig.9 Example of filtering NXDOMAIN responses on malicious traffic capture

For the inference phase, I have a collection of 4 PCAP files (file1.pcap, file2.pcap, file3.pcap, file4.pcap). The first three files each contain 10'000 packets and were provided for blind testing with the acknowledgement that two of them intentionally include beaoning

²⁰ For instance, Scapy, "a powerful Python-based interactive packet manipulation program and library." <https://pypi.org/project/scapy/>, accessed in May, 2024

injections, but without specifying which ones exactly. The fourth file was obtained during the beaoning simulation experiment discussed earlier in Chapter 3.3 *Simulating beaoning*.

Further analysis and inference will be performed on these four network capture files to evaluate the model's ability to detect beaoning activity. The goal is to determine which of the provided files contain the beaoning injections by leveraging the patterns and insights derived from the training data.

As mentioned in the Chapter 1 *Introduction*, malware often employs DNS queries for beaoning purposes. Therefore to determine if my files 1 to 3 exhibit any signs of malicious activity, the first step I take is to filter the traffic capture for NXDOMAIN DNS responses, that occur when a DNS query is made for a domain that does not exist. Malware, particularly those utilising beaoning techniques, frequently generates random domain names for DNS queries in an attempt to locate and communicate with C2 servers. By identifying and analysing these NXDOMAIN responses, I can obtain input on potential malicious activity within the provided network traffic.

For instance, looking at the filtered file1.csv I see a few NXDOMAIN responses in the traffic capture, with two of them being repeated. Repeated queries often indicate an automated process, but based on this screenshot alone, I cannot definitively conclude that these connections are malicious. There are several potential reasons for these repeated queries. Firstly, the application or service attempting to resolve the domain might be configured to automatically retry upon failure. If the domain continuously resolves to NXDOMAIN, the retry could be a built-in response to attempt reconnection or re-resolution. Secondly, a benign automated process might be running on the system, attempting to access a download repository, but for some reason is incorrectly formatted or using a broken URL.

2..	248..	192.168.52.2	192.168.52.131	DNS	169	Standard query response	0xf79a	AAAA	auth.docker.io	AAAA	2600:1f18:2148:bc02:445d:9ace:d20b:c303	AAAA				
2..	249..	192.168.52.2	192.168.52.131	DNS	175	Standard query response	0xb3f4	AAAA	registry-1.docker.io	AAAA	2600:1f18:2148:bc00:8d61:9b62:40aa:8bb1	AAAA				
2..	249..	192.168.52.2	192.168.52.131	DNS	139	Standard query response	0xc937	A	registry-1.docker.io	A	54.236.113.205	A	54.198.86.24	A	54.227.20.25	
2..	249..	192.168.52.2	192.168.52.131	DNS	169	Standard query response	0xe87a	AAAA	auth.docker.io	AAAA	2600:1f18:2148:bc00:8d61:9b62:40aa:8bb8	AAAA				
2..	249..	192.168.52.2	192.168.52.131	DNS	133	Standard query response	0x200f	A	auth.docker.io	A	54.227.20.253	A	54.198.86.24	A	54.236.113.205	OPT
2..	249..	192.168.52.2	192.168.52.131	DNS	169	Standard query response	0x681a	AAAA	auth.docker.io	AAAA	2600:1f18:2148:bc01:571f:e759:a87a:2961	AAAA				
2..	249..	192.168.52.2	192.168.52.131	DNS	133	Standard query response	0x55d8	A	auth.docker.io	A	54.198.86.24	A	54.236.113.205	A	54.227.20.253	OPT
2..	250..	192.168.52.2	192.168.52.131	DNS	243	Standard query response	0xeb77	AAAA	production.cloudflare.docker.com	AAAA	2606:4700:1:6810:67cf	AAAA				

Fig.10 Example of legitimate DNS queries

Two domains (<https://tcp.download.docker.com>, <https://tcp.packages.microsoft.com>) appear to be legitimate services related to Docker and Microsoft. However, the use of service records (SRV `_https._tcp`) in these queries is rather atypical for normal operations of fetching packages or updates. This unusual behavior could be a result of misconfiguration, on the other hand, it could also be an attempt to mimic legitimate traffic for evil-minded purposes. Typically, when systems fetch updates or download software packages, they use standard DNS queries that resolve to A or AAAA records as pictured in *Fig.10*. The presence of SRV queries in this context raises a red flag, suggesting that further investigation is

necessary to determine the intent behind these queries and ensure they are not part of any malicious activity.

Out of the three files I have checked, only file1.pcap and file2.pcap contain NXDOMAIN responses, which might indicate some suspicious connections. Therefore, my particular attention will be concentrated on these first two files to further investigate and analyse the potentially malicious activity indicated by these responses.

3.5 Inference

3.5.1 HGBC

First, I present intermediate results from the ML classifier. These results provide valuable insights and can be used independently or in conjunction with the LSTM model. While they may not represent the final outcome, they already indicate the direction and effectiveness of the initial classification process.

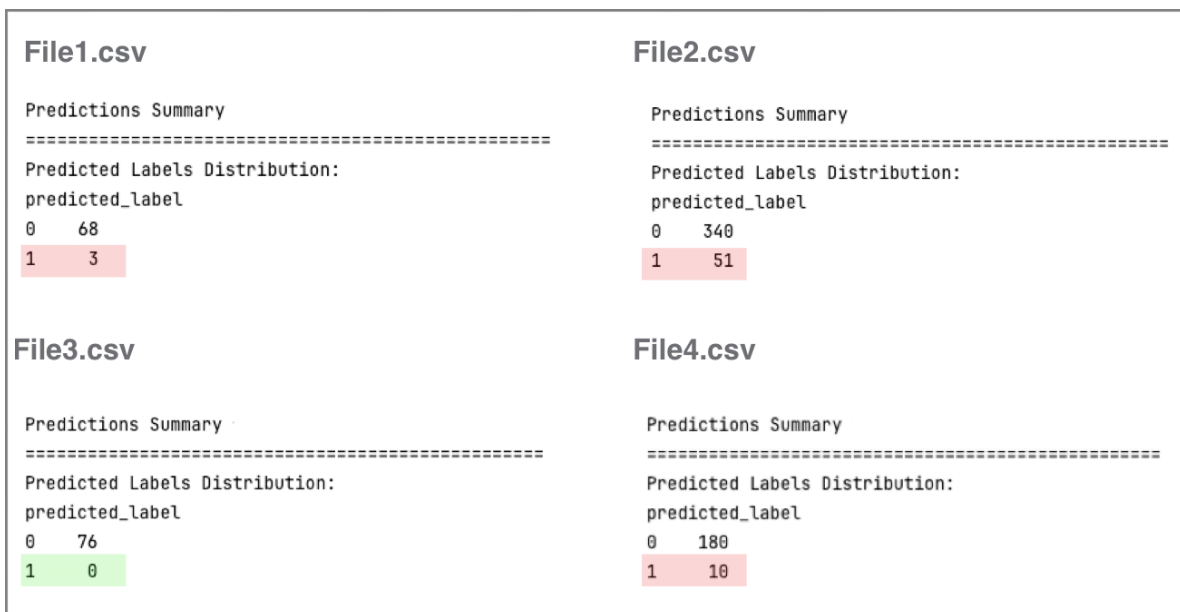


Fig.11 HGBC accuracy report

For file1.csv, the model identified 68 benign flows and 3 malicious flows. While it is challenging to match these detections directly to the PCAP data, this rough estimate indicates that the classifier was able to detect the beaconing activity.

In contrast, file2.csv, which contains three times more flows than file1.csv, showed 340 benign flows and 51 malicious flows, representing approximately 13% of the total flows. This higher proportion of detected malicious flows further demonstrates the classifier's effectiveness in identifying beaconing activity amidst regular network traffic.

As expected, file4.pcap contained the beaconing activity generated during the simulation experiment. The classifier successfully identified the periodic requests and the corresponding responses, validating its accuracy against the known data.

Additionally, a strong sign of validation can be observed in the classifier's performance on file3.csv, which contains only benign traffic. The classifier detected zero malicious flows in file3.csv, indicating its accuracy and reliability in distinguishing between benign and malicious communication patterns. This result underscores the model's capability to accurately identify beaconing activity without generating false positives in a benign dataset.

```
file3.csv

Statistics:
Total sequences: 47
Malicious sequences: 0
Percentage of malicious sequences: 0.00%
```

Fig.12 HGBC-LSTM results on unseen benign file

These intermediate results suggest that the HGBC on a standalone basis is effective in distinguishing between benign and malicious traffic. This indicates that the HGBC can be reliably used on files with a similar volume of flows, providing accurate detection of beaconing activity without falsely flagging benign traffic. This capability is important for ensuring the reliability of the classifier in practical network applications.

3.5.2 HGBC-LSTM

The full model successfully detected malicious sequences in all three files with beaconing injections: file1.pcap, file2.pcap, and file4.pcap from the simulation. Specifically, in file1.csv, the model identified 2 malicious sequences out of 42, resulting in a malicious rate of 4.76% flows in the file. In file2.csv, it detected 48 malicious sequences out of 362, with a malicious rate of 13.26%. In file4.csv, the model found 9 malicious sequences out of 161, yielding the malicious rate of 5.59%.

As for the file3.pcap, the only benign file available for the testing, the model did not find any suspicious sequences which verifies the intermediate output results.

The HGBC-LSTM results demonstrate marginal deviation from the results when HGBSC is used on a standalone basis, just adding some additional accuracy to the prediction. The sequences analysed, each with a length of 30 flows, ensure that temporal dependencies and repetitive patterns can be effectively captured.

file1.csv	file2.csv
Statistics: Total sequences: 42 Malicious sequences: 2 Percentage of malicious sequences: 4.76%	Statistics: Total sequences: 362 Malicious sequences: 48 Percentage of malicious sequences: 13.26%
Sequence 34: Benign Sequence 35: Malicious Sequence 36: Benign Sequence 37: Benign Sequence 38: Malicious Sequence 39: Benign Sequence 40: Benign Sequence 41: Benign Sequence 42: Benign	Sequence 249: Benign Sequence 250: Malicious Sequence 251: Malicious Sequence 252: Benign Sequence 253: Benign Sequence 254: Benign Sequence 255: Benign Sequence 256: Benign Sequence 257: Benign Sequence 258: Malicious Sequence 259: Malicious Sequence 260: Malicious Sequence 261: Malicious Sequence 262: Malicious Sequence 263: Malicious Sequence 264: Malicious Sequence 265: Malicious Sequence 266: Malicious Sequence 267: Malicious Sequence 268: Malicious Sequence 269: Malicious Sequence 270: Malicious Sequence 271: Benign Sequence 272: Benign Sequence 273: Benign
file4.csv Statistics: Total sequences: 161 Malicious sequences: 9 Percentage of malicious sequences: 5.59%	
Sequence 75: Benign Sequence 76: Malicious Sequence 77: Benign Sequence 78: Benign Sequence 79: Benign Sequence 80: Malicious Sequence 81: Malicious Sequence 82: Benign	

Fig.13 HGBC-LSTM results on unseen malicious files

These findings illustrate the high productivity of the framework structure suggested for this study. The initial results from the classifier were already promising, showing the model's capability to identify malicious activity. However, incorporating the LSTM further enhances this accuracy, leveraging its strength in capturing temporal relationships and improving the model's overall performance. The combination of the Histogram Gradient Booster Classifier and the LSTM neural network ensures that the model not only achieves high accuracy in controlled environments but also maintains its effectiveness in more complex and varied real-world scenarios. This improved accuracy underscores the potential of the model to be integrated into Network Intrusion Detection and Response systems or other security toolsets, significantly advancing cybersecurity measures.

4. Conclusion

4.1 Model Complexity

Complex models tend to be more prone to overfitting, especially when trained on relatively small datasets. Overfitting occurs when a model learns the noise in the training data instead of generalising from the patterns. This can lead to excellent performance on training data but poor performance on unseen data.

However, the use of a more complex HGBC-LSTM model may still be justified under certain conditions. The combination of HGBC (Hierarchical Gradient Boosting Classification) and LSTM (Long Short-Term Memory) can model complex non-linear relationships and temporal dynamics in the data more effectively than either approach alone. If the underlying data structure is complex or if there are interactions across time steps that are critical for prediction, the additional complexity of the LSTM may provide substantial benefits.

While the dataset currently used for training consists of just over 12,000 samples, this may not sufficiently challenge the model's capacity to generalize. In scenarios where the model is expected to handle more varied and larger datasets, the robustness provided by the HGBC-LSTM combination could prevent performance degradation. The LSTM component excels at capturing temporal dependencies, making it suitable for modeling the sequential nature of network traffic, while HGBC can effectively handle non-linear relationships and feature interactions.

Moreover, in cybersecurity, where the patterns of malicious activity can be subtle and distributed over time, the ability to model these temporal dynamics is central. The HGBC-LSTM model's capacity to learn from sequential data and adapt to complex patterns enhances its potential for accurate beaconing detection. Ensuring the model's strength and performance on larger, more diverse datasets will be essential for its successful deployment in real-world environments. This approach aims to strike a balance between model complexity and generalization, leveraging the strengths of both HGBC and LSTM to build a powerful and reliable detection system.

4.2 Limitations of the study

The insufficiency of real-world data in cybersecurity is primarily due to concerns about privacy and security. Organisations are often reluctant to share data because it can contain sensitive information about their networks, operations, and vulnerabilities. Sharing such data could potentially expose them to further attacks or legal liabilities. Additionally, there are often regulatory and compliance constraints that limit the extent to which data can be shared, particularly in sectors like finance and healthcare where data protection laws are strict.

Another reason is the inherent difficulty in capturing and labeling cybersecurity data. Cyber attacks are complex and vary in nature, making it challenging to create comprehensive datasets that accurately reflect the diverse threat landscape. Furthermore, much of the valuable data resides within private networks and is not accessible to the broader research community.

These factors contribute to a lack of publicly available, high-quality datasets, which hinders the development and testing of advanced cybersecurity models. Researchers and practitioners must often rely on simulated data or small, anonymised datasets, which may not fully capture the complexities of real-world cyber threats.

The significant testing and evaluation of my model were limited due to several reasons. Firstly, there were time constraints that restricted the duration available for comprehensive testing. This limited timeframe meant that not all possible data flows and scenarios could be thoroughly evaluated. Additionally, there were resource limitations, both in terms of computational power and manpower, which further constrained the ability to conduct extensive trainings and tests. These limitations obstructed the ability to evaluate the model on varied data comprehensively, potentially affecting the durability and generalisability of the model's performance across different scenarios.

4.3 Application

Combining the HGBC-LSTM model with traditional anomaly detection methods such as statistical analysis, rule-based systems, or clustering algorithms, creates a hybrid detection system that leverages the strengths of both approaches. This hybrid system can improve overall detection capability and reduce false positives by using the HGBC-LSTM model's advanced learning algorithms to identify complex patterns in network traffic indicative of beaconing activity, while traditional methods provide a foundational layer of anomaly detection. For example, in a corporate network, this system could more accurately detect subtle, sophisticated beaconing attempts by APTs that might otherwise be missed or flagged incorrectly.

One of the possible scenarios for integration of the HGBC-LSTM model in a real-time detection and response system might look as follows: first, network traffic is continuously monitored, and packets are captured at regular random intervals to ensure a comprehensive representation of the traffic over time. These captured packets are saved in PCAP files. These batches are then fed into the HGBC-LSTM model for analysis. The model evaluates the processed data against predefined thresholds, and if the output exceeds the set threshold, it indicates potential malicious activity or an anomaly. Upon detecting an anomaly, the system triggers an alert. This alert can notify security personnel, activate automated response mechanisms, or initiate further investigative procedures. The entire process, from packet

capture to alert triggering, could be automated using for instance tools as Cron, a Unix-based scheduling utility that automates the execution of scripts or commands at specified intervals. Cron scheduling may very well employed in combination with a Python script and libraries suitable for network traffic analysis (scapy, tshark, tcpdump, etc) or with any of the available network automation frameworks.

One of the key advantages of the HGBC-LSTM model is its modularity. The intermediate results from the HGBC can be utilised independently, providing valuable insights into the classification process before the LSTM component is applied. This modularity allows for flexible integration into different stages of the detection pipeline, enabling security teams to leverage the classifier's output for preliminary assessments or as part of a more extensive analysis framework. Additionally, the HGBC-LSTM model boasts a fast training speed, significantly reducing the time required to deploy updates and retrain the system in response to emerging threats. This rapid training capability ensures that the detection system remains up-to-date and effective against the latest beaconing techniques, providing a powerful and adaptive defense mechanism for real-time security applications.

This model framework is not limited to security applications; it also holds potential for tasks involving pattern recognition and pattern matching due to the high adaptability of its architecture. To apply the framework to new tasks, one needs to adjust the preprocessing steps where the raw data is converted into a data frame format specific to the new domain.

For example, in network performance monitoring, the model could be used to detect and predict network congestion by identifying patterns indicative of congestion versus normal traffic flow. In Quality of Service (QoS) management, it could analyse traffic patterns to classify traffic as either meeting QoS requirements or not, thereby ensuring optimal bandwidth allocation and minimising latency.

In network anomaly detection (which also covers misconfigurations, and might not be malicious per se), the framework can identify unusual traffic patterns by classifying them as either normal or anomalous, helping to spot configuration issues. Additionally, in network usage analytics, the model can be used to classify user behavior patterns as either typical or atypical, assisting in optimizing network resources.

Lastly, for network traffic classification, the model can distinguish between different types of traffic based on predefined categories, classifying them as either belonging to the expected category or not. This helps ensure the efficient handling of various data streams.

Overall, the flexibility and adaptability of this binary classifier model framework make it a valuable tool for enhancing network management and optimisation across a variety of applications within computer network traffic.

4.4 Requirements

The HGBC-LSTM model meets a variety of functional and non-functional requirements, which are central for its effectiveness and reliability in real-world applications. Among functional requirements, I would like to name a few:

1. Real-time Detection, where the model can analyse network traffic data in real-time to identify beaconing activities swiftly.
2. Integration with Existing Systems, specifically integration with existing security infrastructures such as SIEM, IDS, SOAR, and firewalls.
3. Adaptability as it is able to adapt to new types of beaconing activities and update its detection capabilities accordingly.

The model also successfully satisfies some non-functional requirements such as:

1. Performance (as in speed), as the model processes data and generates results quickly, with minimal latency to ensure real-time detection and response.
2. Scalability, as it can handle large volumes of network traffic and scale according to the organisation's size and network complexity.
3. Accuracy, as it drastically minimises false positives and false negatives²¹, ensures that genuine threats are detected while benign activities are not incorrectly flagged.
4. Maintainability: The model is easy to update and maintain, with straightforward procedures for retraining and integrating new threat intelligence.
5. Efficiency: The model is resource-efficient due to utilising the Histogram gradient booster algorithm for intermediary results, it uses computational resources effectively to avoid overloading the network infrastructure.

Overall, the HGBC-LSTM model demonstrates its ability to effectively enhance an organisation's cybersecurity, providing effective and adaptive defense mechanisms for the detection and mitigation of beaconing activities across various sectors, thereby bolstering the overall security disposition of organisations.

4.5 Future Steps and Recommendations

To better leverage the complexity of the HGBC-LSTM model, I recommend augmenting the dataset with more diverse and extensive data, including encrypted traffic, jittered signals, and other obfuscated communication types.

Adversaries often employ various techniques to conduct beaconing discreetly. The most relevant techniques include but are not limited to:

²¹ according to the accuracy reports, discussed in the Chapters 3.2 *Training* and 3.5 *Inference*

- DGAs, that generate numerous domain names for C2 servers, complicating the blocking of malicious traffic.
- Fast Flux, a technique that rapidly changes the IP addresses associated with domain names to evade detection and takedown efforts.
- DNS Tunnelling, which helps encode data in DNS queries and responses to create a hidden communication channel.
- HTTP/HTTPS Tunnelling, that allows hiding malicious communication within standard web traffic.
- Steganography, when data is embedded within legitimate files or images to conceal beaconing signals.
- Packet Fragmentation, that breaks up the payload into smaller packets to avoid detection by IDS/IPS systems.

Ideally, the training dataset should have a volume of over 1 million flows. This would significantly help in evaluating the true benefit of the complex model architecture under more challenging conditions, ensuring its steady performance and generalisability.

I suggest the unseen data be obtained over a longer period of time to capture a comprehensive view of network activity. It should ideally include traffic data from different times of the day continuously recorded over longer periods of time, during various days of the week, and even account for seasonal- or festive-related traffic changes if necessary. This approach ensures that the dataset reflects the specific patterns and anomalies of the network, providing a more accurate representation of its *normal* and *abnormal* behavior. Collecting data over an extended period will help identify sequencing and any cyclical changes that might occur. Additionally, it is important to consider special events, holidays, and other factors that may impact network traffic. By encompassing these variables, the dataset will be more durable and effective for training and testing the model, ultimately leading to better performance and reliability in real-world applications.

I would also recommend developing a framework for incremental learning where the model can continuously learn from new data without needing complete retraining. This can help in maintaining the model's accuracy and relevance over time as network patterns evolve.

Academic Vocabulary

Beacon - (in cybersecurity) a signal sent by malware or a compromised device to an external server.

BCE - or Binary Cross-Entropy, a loss function used for binary classification tasks, measures the difference between predicted outcome and actual label.

Bias - (in machine learning) is a type of error where certain aspects of a dataset are given more emphasis or representation than others.

Cell State - a component of Long Short-Term Memory neural network, that carries and updates information over time, allowing the network to maintain long-term dependencies.

Entropy - a measure for randomness, unpredictability (here: payload entropy shows how uneven byte distribution is)

Flow - sequence of data packets transmitted across a network from a source to a destination, involving various stages such as packet creation, routing, forwarding, and reception.

Gradient Boosting - a machine learning algorithm that constructs sequential trees and aggregates their predictions

Hidden State - (in neural networks) a set of values that capture and maintain information about past inputs to inform current and future predictions.

Long Short-Term Memory - is a recurrent neural network (RNN) architecture commonly used in deep learning for its ability to capture long-term dependencies, making it particularly well-suited for sequence prediction tasks.

Network protocols - a set of rules that define how connected devices communicate across a network to exchange information efficiently and securely.

Outliers - data points that significantly differ from others

Packet - a smallest data unit that traverse through a network

Variance - (in machine learning) variability in model predictions, indicating how much the ML function's output can change based on different datasets. High variance typically arises in highly complex models with a large number of features.

Disclaimer

This report has been proofread with the assistance of [ChatGPT](#).