UNIVERSITY OF CAGLIARI

MASTER DEGREE IN COMPUTER ENGINEERING, CYBERSECURITY AND ARTIFICIAL INTELLIGENCE

Real-Time DoS Detection: a ML-based system to combat DoS attacks.

COURSE
Network Security

STUDENT
Christian Scano (70/90/00346)
Marco Ledda (70/90/00471)
Luca Minnei (70/90/00347)

A.A 2023/2024

Contents

Abstract

With the increasing reliance on digital infrastructure, Denial of Service (DoS) attacks have become a prevalent threat to the availability of online services. These attacks, often orchestrated by botnets, can affect negatively networks, causing economic and reputational damage. This project proposes a novel approach to bot detection, aiming to improve network security by accurately identifying and mitigating DoS attacks. In this project we propose a prototype of a Machine Learning model, involves analyzing network logs patterns to distinguish between legitimate user activities and anomalous activities that could be related to bot activities. Theoretically, the project in the real world, should be designed to work in real-time environment performing data processing and using adaptive learning techniques, in order to improve its detection accuracy driven by the data. The aim of the project is to develop a light and efficient framework that can be integrated in already existent network infrastructures without affecting the performances of the network, providing at the same time robust defense against DoS attacks. What we expect is to significant reduce the incidence of successful DoS Attacks, by finding several thresholds for different time slices that distinguish the normal traffic from the anomalous one. Through rigorous testing and validation, this project aims to offer a valuable tool for network administrators and cybersecurity professionals in the fight against cyber threats.

Chapter 1

Introduction

Since our project aims to identify Denial of Service (DoS) attacks that usually are made by bot network, is necessary to briefly introduce what we mean with bot detection.

1.1 Bot detection

The world of software robots, or "bots," spans a broad spectrum of sophistication and purposes. However, one trait unifies them: their capability to automate tasks and processes. Automation is, therefore, the primary goal of bots, which becomes apparent through their interaction patterns within network traffic. The general rule is that the more sophisticated a bot (i.e., its ability to blend with or mimic normal traffic), the higher the level of abstraction required to detect such patterns, necessitating increasingly sophisticated detection mechanisms. This scenario sets the stage for a continuous arms race: as bots become more sophisticated, also the mechanisms should become more sophisticated to identify and distinguish them from legitimate traffic sources. This introduction outlines three principal levels of bot sophistication—low, medium, and high—and outlines detection measures for

each. To circumvent detection at a given level of sophistication, adversaries must intensify their efforts. Attacks designed to evade detection at a specific level may be revealed through detection methods at a higher level of sophistication, thus escalating the effort required for detection. For every level of sophistication, Machine Learning can be exploited to automatically learn from both existing ground truth (known exploits, bots, attack techniques) and traffic observed in real-world web services, enabling the implementation of anomaly and outlier detection mechanisms.

1.1.1 Low Sophistication Bots

This category includes programs that clearly identify themselves by submitting specific inputs to a web server or application, such as a distinct User-Agent or request URIs/parameters. These bots do not attempt to conceal their nature, either because their intent is benign (e.g., Google Bot) or because they are deployed by adversaries with minimal effort or knowledge (e.g., script kiddies using publicly available exploits with default configurations).

1.1.2 Medium Sophistication Bots

This category includes programs that employ some form of hiding or evasive technique to mask their nature, such as using a fake User-Agent, leveraging public cloud providers or anonymization networks, and sending custom requests to web servers and applications. These bots are operated by adversaries who invest some effort and knowledge to customize their behavior (e.g., as part of a botnet or coordinated activity) and thus cannot be reliably detected by methods designed for low sophistication bots.

1.1.3 High Sophistication Bots

This group consists of programs that utilize advanced hiding or evasive techniques to completely emulate the behavior of a real user or normal traffic, such as through browser automation frameworks or exploiting browser vulnerabilities across a multitude of real users. Their source may also be indistinguishable from that of legitimate users or traffic. These bots are deployed by adversaries investing significant effort and knowledge to customize their behavior (e.g., as part of targeted activities against web services) and thus cannot be reliably identified by detection methods suited for low/medium sophistication bots.

Chapter 2

Scenario

In an e-commerce landscape, bots have become increasingly sophisticated, engaging in unauthorized data scraping, inventory hoarding, and launching denial-of-service attacks. These bots can mimic human shopping behavior, making it challenging to distinguish between legitimate customers and malicious bots. As a result, an e-commerce platform experiences frequent out-of-stock situations for high-demand items, customer dissatisfaction, and compromised website performance. To combat these challenges, the e-commerce platform employs advanced bot detection techniques, leveraging machine learning algorithms and behavioral analysis. By analyzing user interactions and traffic patterns in real-time, the platform can identify and differentiate between human and bot activities with high accuracy. Anomaly detection techniques further enable the identification of unusual traffic spikes or patterns that deviate from known user behavior, flagging them for further investigation or immediate blocking. This approach allows the e-commerce platform to protect its inventory from being hoarded by bots, ensure fair access to products for genuine customers, and maintain optimal website performance. Through continuous adaptation and learning, the platform stays ahead of bot developers, safeguarding its operations against emerging threats.

Chapter 3

Implentation

3.1 Case of Study

Our case study focuses on the logs from an Iranian e-commerce website. We selected this log file as our dataset because it presents a real-world scenario similar to the one we aim to analyze. The dataset consists of various columns: we only consider those related to the IP address associated with the request and the timestamp. The reason why we chose only these columns is because our model tries to use as little information as possible so as to make it more complicated for the attacker to modify them. Each entry represents a single request from a unique source. The dataset contains approximately ten million samples collected over four days, which, for our purposes, is an adequate sample size for testing. In a practical scenario where we would apply this model, opting for a dataset that spans a longer period (such as a month) might be preferable. Nevertheless, this could lead to an increased time requirement for model fitting.

Our model's job is to spot anomalous high numbers of requests during specific time periods. The basic concept is pretty straightforward: we're going to count how many requests are made in a set period of time - which we'll refer to as a

"window" - and then use a math formula to figure out the number of requests (the threshold) that marks unusual behavior from a single source during that window.

3.2 Offline implementation details

In the field of network security, the identification of anomalous behavior within data streams is paramount to ensuring the integrity and reliability of information systems. This chapter dig up into a sophisticated approach for anomaly detection, specifically through the implementation of the Cumulative Probability Distribution for Numerical Values (CVN) model. This model is designed to efficiently identify and alert on anomalous events within a data set, leveraging statistical methods to distinguish between normal and aberrant behavior.

3.2.1 Global Anomaly Probability Threshold

At the core of the CVN model lies the global anomaly probability threshold, denoted as p_{max} This threshold represents the maximum probability for an event to be considered anomalous. The flexibility of p_{max} allows it to be tailored to each model instance, enabling a customizable sensitivity level for anomaly detection. By setting this parameter, organizations can control the volume of anomaly alerts generated, ensuring that the detection system is aligned with their operational requirements and risk tolerance.

3.2.2 Cumulative Probability Distribution

The CVN model operates on the principle of cumulative probability distribution. It calculates the probability $p(n \leq N_i)$ for a numerical variable n, which represents an observable metric within the data (e.g., the number of requests from an IP address within a certain timeframe). The variable n is considered as a random

variable, with the heuristic understanding that higher values of n are indicative of more anomalous behavior.

To compute this distribution, the model evaluates the ratio of sources that have generated a value of n less than or equal to N_i, over the total number of sources, where N_i are the observed values sorted in ascending order (Formula 3.1). This approach allows for the identification of significant deviations from the expected behavior, pinpointing anomalies effectively.

$$p(n \leq N_i) = \frac{\text{Number of sources that have generated } n \leq N_i}{\text{Number of total sources}} \tag{3.1}$$

3.2.3 Alarm Threshold

The alarm threshold N_t, is a critical parameter within the CVN model. It is set through a process similar to single-linkage clustering, aiming to identify a point in the distribution where the slope is minimal. This point, or plateau, signifies that the addition of further data points does not significantly alter the distribution, suggesting that the majority of the observed values are legitimate. The expression related to the computation of N_t is:

$$N_t = \arg\min \frac{p(n \leq N_{i+1}) - p(n \leq N_i)}{N_{i+1} - N_i} \tag{3.2}$$

An alarm is triggered when the probability:

$$p(n > N_t) = 1 - p(n \leq N_t) \tag{3.3}$$

falls below or equals p_{max}, indicating an anomalous event.

3.2.4 Anomalous requests detection per source and time window

The CVN model is particularly capable of monitoring and analyzing data traffic from various sources, such as IP addresses, over distinct time windows. These windows range from 10 seconds to approximately 16 minutes encompassing three distinct temporal intervals. The reason why we have chosen just these three temporal intervals is because in a real scenario, after 16 minutes there is a high probability that the infrastructure, that should be defended, has gone down.

3.2.5 Training Algorithm

The training algorithm for the CVN model involves the collection and analysis of request data over specified time windows. A dictionary called count structure is employed to store the number of requests per time window and IP address. For each request made by a unique source in given a time window (w), the algorithm calculates the cumulative probability $p(n \leq N_i|w)$. All the cumulative probabilities in the window are used for calculating The alarm threshold $N_t(w)$ for each window, serving as a crucial parameter for real-time anomaly detection.

This model emphasizes the importance of adaptability and precision in the detection of network anomalies. By carefully setting the parameters such as p_{max} and N_t, and by analyzing the cumulative probability distributions, the CVN model provides a robust framework for identifying potentially malicious or anomalous behavior within network traffic.

Algorithm 1 Training Algorithm for Anomaly Detection

Initialize a dictionary `count` to store the number of requests for each time window and IP address.
for each request **do**
    Obtain the timestamp and source IP address.
    for each time window w **do**
        Calculate `window` by right-shifting the timestamp by w.
        `count`[`window`][`IP`]$+ = 1$.
    end for
end for
for each time window w **do**
    Calculate the cumulative probability $p(n \leq N_i|w)$ where N_i is the number of requests generated by each source within window w.
    Find N_t, solving the optimization problem (Ref. 3.2).
end for

Chapter 4

Experimental results

4.1 Offline experimental results

The offline experiment uses the implementation of the Machine Learning model explained above with the same setup. The setup consists of the dataset presented before and the Python implementation of the model. The offline experiment consisted of finding the thresholds that will be set in the online experiment (an e-commerce website). In order to find the results, we decided to plot a curve that presents as x-axis the number of requests and as y-axis the cumulative probability associated with that number of requests. Therefore, in order to find the plateau of the curve, which represents the threshold for a specific window, we calculate the derivative for each point of the curve and we consider the minimum one with a value greater than zero. In the figure below, we can see the curves respectively for the window of 10 (blue curve), 100 (orange curve) and 1000 (green curve) seconds (Fig 4.1)).

From the plateau values we have found for each curve, we can see that, as we expected, increasing the window we will get a greater threshold.

It's difficult to say if the threshold found is good enough to identify an anoma-

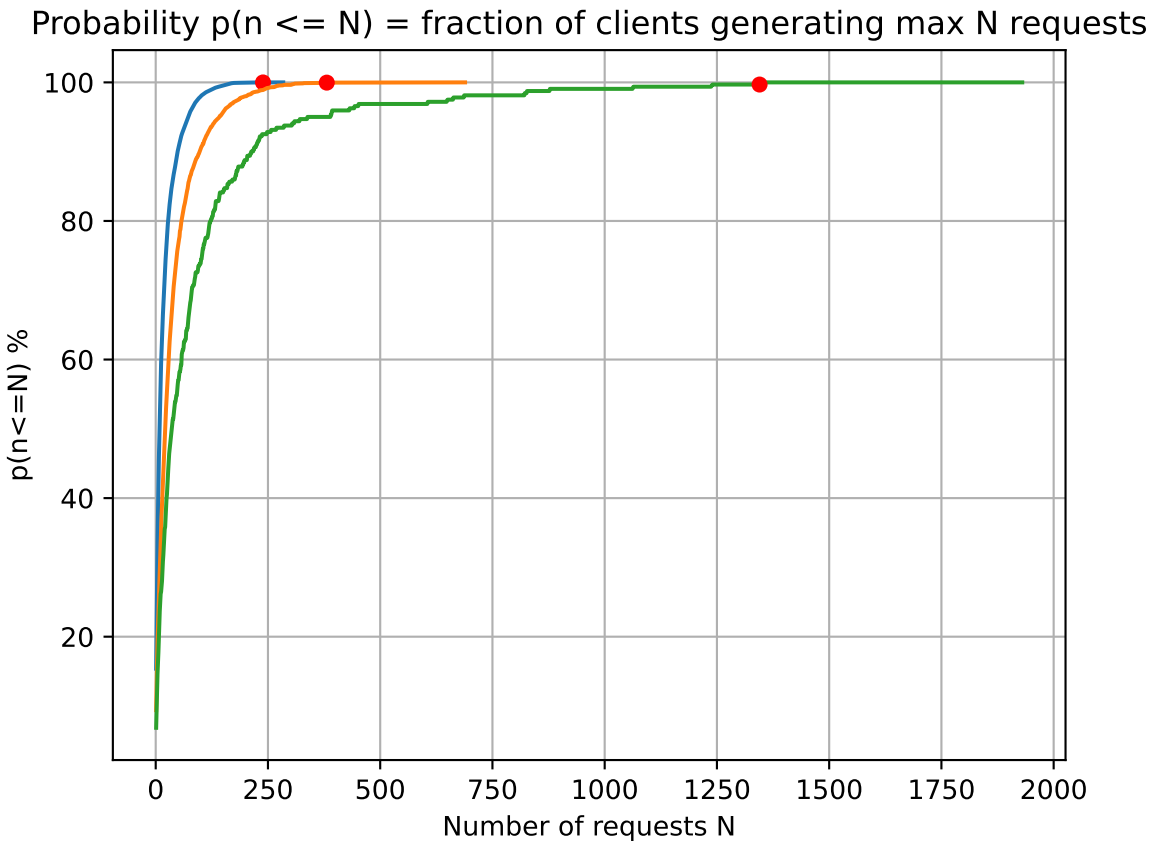

Figure 4.1: Curves representing the cumulative probability related to the number of requests. The red points represent the thresholds N_t found by the training algorithm. The blue curve is related to the window of 10 seconds. The orange curve is related to the window of 100 seconds. The green curve is related to the window of 1000 seconds.

Window Size	**Threshold N_t**
10 seconds	239
100 seconds	381
1000 seconds	1278

Table 4.1: Threshould found for each window.

lous source. However, we are sure that the algorithm prevents users who are consuming too many resources in the web service that could be part of a Denial of Service Attack or Bot attack.

4.2 Online experimental set up

During the testing phase, we constructed a simulated e-commerce platform to serve as our client infrastructure. Additionally, we engineered an active component—a Rust-based reverse proxy—designed to overlay the e-commerce system and provide protection. This proxy makes available an API capable of receiving metrics, or "thresholds," generated by our offline model. These thresholds were utilized to identify anomalous activities and subsequently block clients exceeding predefined thresholds.

4.3 Online experimental results

To test if the Online Part is working we generate a certain number of requests from a specific source, to check if the system blocks the sources that generate several requests above the thresholds. Fortunately, everything works fine, and the online model blocks correctly the sources. Obviously, in a real scenario, the threshold is

gonna be updated, depending on the necessities of the client, in a certain amount of time changing thresholds.

Chapter 5

Conclusions

In conclusion, the development of our network DoS (Denial of Service) detection machine learning model represents cybersecurity measures that several web applications could use without changing anything because our model is domain-specific, which means that depending on the domain gonna use different thresholds that fit a specific domain. Lightly structuring the model allows the model to be easily used for different applications because doesn't need much resources. By analyzing network traffic patterns and anomalies, the model can efficiently detect and respond to malicious activities, thereby bolstering the overall resilience of network infrastructures. Moreover, the adaptive nature of the model enables continuous improvement of the detection capabilities, ensuring adaptability in different periods of the year where the web application can have an average of requests higher or lower than the previous one. A possible improvement could be considering other parameters of each request becoming more complicated but maybe more reliable, or could be interesting to consider not only the single sources but also maybe a group of sources for identifying the DDoS attacks. And in the end, this kind of tool nowadays is becoming crucial in safeguarding network integrity and maintaining operational continuity.