Flow-Based Botnet Detection with AI Models

Bohdan Panchuk [1]

[1]*V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine, Akademika Glushkova Avenue, 40, Kyiv, 03187, Ukraine*

Abstract

This paper outlines the generalized framework for building end-to-end botnet network activity detection systems using artificial intelligence (AI) techniques. Network flows reconstruction was used as a primary feature-extraction method and different AI classifiers were considered for achieving better detection rates. The results of the latest research by other authors in the field are incorporated to implement a more efficient approach for botnet discovery. The implementation of the described intrusion detection approach was tested on a dataset with real botnet activity traces. The performance metrics for different AI classification models were obtained and analyzed in detail. Different data preprocessing techniques were tried and described which helped to improve the results even further. Some options for future enhancement of network feature selection were proposed as well. Finally, the comparison of the obtained performance metrics was drawn against the results provided by other researchers in this field, demonstrating the appreciable improvements of CNN- and LSTM-based classification of network flows data organized as time-series.

Keywords

Information security, intrusion detection, botnet, network flow, artificial intelligence

1. Introduction

In the environment of ubiquitous Internet access and integration, network attacks have become a usual phenomenon. Regular means of protection like thoroughly configured firewall rules and signature-based antivirus programs do not cover the full attack space. Many attacks are conducted exploiting well-known regularly open ports and are masked under the normal network activity. Such activity is impossible to detect using just static security mechanisms like firewalls and signature matching. Intrusion detection systems (IDS) are the dynamic line of defense designed to identify malicious activity at runtime. This paper describes a network-based detection system with AI models used for traffic classification. To better understand the benefits of such an approach, it is worth mentioning the common alternatives.

Hervé Debar in 2009 [1] gave an overview of different IDS types and their taxonomy, dividing detection tactics into knowledge-based and behavior-based.

Knowledge-based systems, which are also known as rule-based and considered to be a classic approach, utilize a set of predefined rules created by a human expert, describing the known attack patterns, and matching those against the captured events (e.g., logs or network packets). An example of such a rule may be “more than N failed connection attempts from source IP address X”. After the rule is triggered, the system produces an alter and potentially performs an automatic response (e.g., blocking the malicious IP address). There are several drawbacks of such an approach. A human expert is required for rules creation and to keep the list up to date later. More importantly, the construction of some more sophisticated rules may be extremely difficult as the attack patterns cannot always be easily described. Attackers often intentionally stretch the intrusion in time by sending malicious

13th International Scientific and Practical Conference from Programming UkrPROGP’2022, October 11-12, 2022, Kyiv, Ukraine
EMAIL: bogdanscloud@gmail.com (A. 1)
ORCID: 0000-0002-5389-359X (A. 1)

packets irregularly and mixing them with benign data. Finally, as the name implies, knowledge-based systems require specific prior knowledge to counter the attack.

Behavior-based systems, on the other hand, are presumably more flexible as designed to "learn" the user behavior and create a normal activity profile. An outlier in the monitored activity can be viewed as an anomaly and classified as an intrusion. Despite the enhanced flexibility, the complexity of that approach lies in the difficulty of defining what is a normal behavior. Usual users can change their behavior at any time, and it is hard to define to which extent the change is acceptable before considering it an anomaly.

AI-driven systems are an alternative that looks the most promising, considering rapid AI development in the last decade. The advantage of AI is that it may encompass the benefits of knowledge- and behavior-based systems with significantly less human input involved. AI models do not need human experts as they learn the rules and profiles from the data. For instance, decision trees and neural networks can derive significantly more complex rulesets in comparison to knowledge-based systems. A supervised learning model studies the normal activity alongside the attack patterns, while unsupervised learning models can learn to distinguish anomalous traffic on their own, which makes them superior to behavior-based systems. There are 2 major drawbacks, however: firstly, is that those learned rules are often not human-interpretable; secondly, a significantly larger training dataset is required – in the case of supervised learning, the dataset must be labeled as well.

It was shown that AI-based IDS performance significantly depends on the feature selection method as well as the choice of underlying classifier architecture. The work edited by Al-Sakib Khan Pathan in 2014 [2] among others briefly describes the benefits and drawbacks of using different AI techniques to perform the detection. Those which are mentioned: Deep Neural Networks (DNN), Self-Organized Maps, Markov Models, Bayesian Systems, and Support Vector Machines (SVM). In more detail, Arnaldo et al. in 2017 [3] described the usage of Random Forests (RF), Feed-forward neural Networks (FFNN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) to analyze log and relational data as time-series. However, in that study, neither RNN nor CNN enhanced the results in comparison to the regular FFNN or RF. In the scope of this paper, it will be shown that LSTM (a variation of RNN) and CNN do improve the results in the case of network flow analysis, given that data preprocessing and aggregation is done correctly.

In order not to dissipate the reader's attention, this paper is focused on botnets traffic detection only. Botnets are among the top of the most destructive cybersecurity threats and their number is actively growing on a yearly basis. Unlike other attacks, botnets are often designed to stay under the radar for a long time period – the longer the infection stays undiscovered, the more harm is caused in the form of data leakage, malicious surveillance, and involuntary participation of the infected machine in other network attacks, so botnet detection is crucial for any information security system.

It should be noted, however, that despite the focus on botnets, the specialization is made only at the stages of network feature selection and the dataset used for AI models training. The general detection framework can be applied for other kinds of network attacks, except the models must be trained specifically against those and the feature set for the effective discovery may be different.

2. Intrusion detection based on network flows

This chapter provides a high-level description of the data-processing pipeline for flow-based network intrusion detection. Each step will be described in more detail in the following chapters. The complete pipeline for AI-driven network-based intrusion detection is shown on the Figure 1.

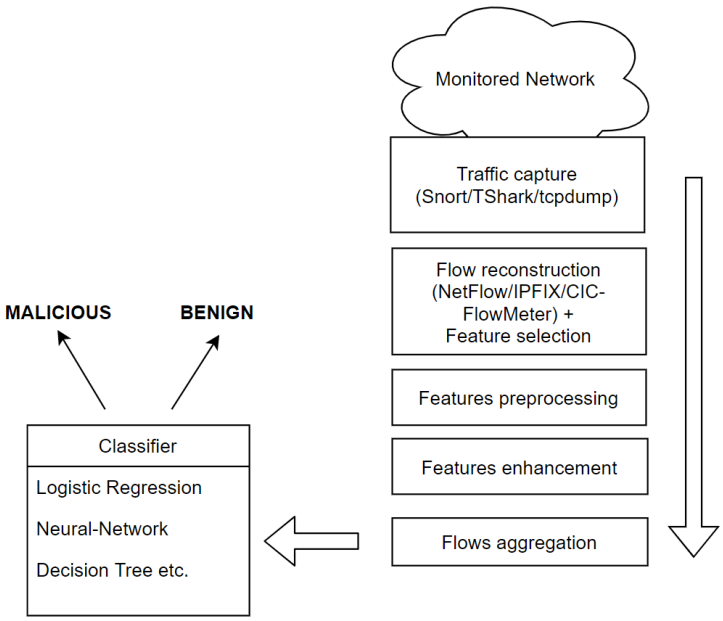

Figure 1: AI-driven network-based intrusion detection pipeline

The initial stage is network monitoring in form of traffic capture. The capture should be performed at the nodes of interest of the monitored network. Packets captured propagated down the detection pipeline for further analysis.

The second stage is network flows reconstruction from the captured packets. Network flow aims to represent the end-to-end conversation between two remote processes (endpoints). The major benefit of such an approach is that it allows isolating packet groups belonging to the same communication process from the rest of unrelated traffic, so no matter how much noise is coming through the channel, a classifier can operate separate conversations.

Another benefit of working with network flows is easy contextual features extraction, which is the third stage of the pipeline. Aggregates across all the array of packets are computed, those may be statistical values describing packet properties like frequency, size, etc. The occurrence of some specific network events in the flow can be considered as well (e.g., presence of DNS lookups). These calculated feature values numerically describe the full context of the conversation. There might be multiple flows over the longer time span between two considered endpoints, so it is important to capture the timestamp of when the current conversation was initiated and when it ended – it can be used at the later stages of the pipeline.

After the numerical representation of each individual flow is obtained, the fourth step is feature preprocessing: cleaning up invalid values, normalization, and scaling. At this point, it is already possible to perform traffic classification based on those features only. However, as will be shown later, the detection quality can be improved by feature enhancement applying techniques like dimensionality reduction and smart feature selection. This stage is optional. The detection rate can be improved even further if instead of considering each flow in isolation, flows are aggregated into ordered groups and treated as time series. The approach was proposed by Arnaldo et al. in 2017 [3], although, they describe it in terms of grouping related events by fixed time intervals and calculating statistical features for each group. Such timeframe-based aggregation is not equivalent to the network flow reconstruction technique. More details are provided in the later chapter.

The final stage is flows classification by AI model based on the features obtained in the previous stages. Depending on the model architecture, the classification can be performed on both individual flows and a series of flows.

Essentially, the network-based detection problem is reduced to aggregating the captured packets into flows represented by vectors of numeric (and categorical) features and then marking those as malicious or benign by an AI-based classification model.

This approach assumes supervised learning, so to train the model there must be a labeled dataset provided. Such a dataset can be generated in laboratory conditions by artificially producing malicious traffic (e.g., infecting machines by a botnet) and capturing their network traces. The labeling is performed by knowing malicious IPs and time intervals – after flows reconstruction, all the flows containing the known IP as their source or destination (and falling under specific time intervals if those are specified) are considered malicious. Other flows are labeled as benign.

The full detection pipeline was implemented in scope of this paper creation: a labeled set of flows was derived from ISCX Botnet dataset 2014 [4] provided by the Canadian Institute for Cybersecurity. Further feature processing and aggregation was performed. Afterwards, different AI classification models were trained based on those features. Finally, the pipeline was tested on previously unseen data and performance metrics were obtained. More details are provided in the following chapters.

3. Network traffic capture

Event capturing is an important part of every intrusion detection system. In the case of the detection pipeline described previously, such events are network packets. Network traffic monitoring is usually performed by "packet sniffing". These techniques allow capture network packets on layers 2-5 of the OSI model passing through device network interfaces in any direction. The usual tools to perform such tasks are:

- Wireshark – the most common packet capture tool with UI and ability to perform various filtering (e.g., based on source/destination IP, port protocol, etc.). It can recognize different protocols on different network layers and reconstruct the packet, segments, requests, etc.
- TShark – console version of Wireshark. Can run as a background process and is most useful in environments with multiple hosts, where automatic provisioning is used.

It should be noted, however, that plain packets analysis is quite expensive and sometimes not accessible process. First, because of the huge amount of data being generated: essentially capturing a packet means duplicating it on some host machine, before passing further to the analyzer. This at least doubles the amount of data being transferred and stored. Furthermore, it is not always possible to use the regular capturing tools on some network devices (e.g., some switches and routers), without degrading its throughput performance. Techniques like "port mirroring" double the amount of traffic and may not be supported on all the devices (especially on the cheap ones). Moreover, capturing packets on a single node is not sufficient: different packets with the same destination may take different routes while traveling over the network, so to capture all possible packets, all intermediate nodes must be monitored. This, however, may lead to one packet captured on multiple intermediate nodes it came through, which causes even higher data duplication. All these large volumes of data must be stored and analyzed, often in real-time, which sometimes is too computationally expensive. Finally, data duplication and raw packets analysis is always a security risk, as most likely it contains sensitive or confidential data. Also, in terms of IDS, while the packet body may contain the crucial information to identify an attack (e.g., botnet IRC communication commands), deep packet analysis becomes impossible, when traffic is encrypted (which is often the case).

4. Network flows reconstruction

The complementary approach to network traffic analysis, which addresses some of the issues mentioned above, is processing network flows, instead of separate packets. To achieve this, flows need to be reconstructed from the groups of individual packets and only then passed further to the analyzer.

There are various ways for doing network flow reconstruction, with different complexity and result quality. In theory, a network flow should unite all packets transferred between two processes (endpoints) sharing the same context and time span to represent the complete communication session between those. An example would be the sequence of packets exchanged in the scope of a single TCP connection. Flows can be unidirectional or bidirectional, depending on reconstruction procedure and requirements. As the name implies, unidirectional represents the set of associated packages coming

from one endpoint to another in a single direction, while bidirectional captures packets coming in both ways. A traffic flow is identified by the following tuple:

<Source IP address, Destination IP address,
Source Port, Destination Port,
(Transport) Protocol, Start Time (Finish Time)>

So, given a set of network packets, the flow can be considered as an aggregate of packets grouped by those attributes. Aggregation is performed by computing a set of values of interest (further called features), which represent the generalized description of the properties of all packets associated with the flow. Those features can be statistical (e.g., avg/min/max packet size), categorical (e.g., protocol type) or simple counters (the number of specific TCP flags, total packets count, etc.). All these values are crucial and used later by the detection classifier to make the judgment of whether the flow/a series of flow is malicious or not.

The flow reconstruction algorithm can be generalized to 3 main processes: 1) registration of new active flows 2) association of a packet with one of the currently active flows 3) marking flows, which have hit one of the termination conditions, as finished and removing those from active flows list. The algorithm constantly tracks the list of already discovered active flows; for each captured network packet check if there is an active flow the packet can be associated with. If a packet matches one of the active flows, then feature values of that flow are adjusted accordingly. In case if there are no active flows the packet can be associated with, a new flow is created and added to the active flows list. The mentioned tuple of packet attributes is used to uniquely identify the flow. As this is the first packet in the flow, its timestamp is put into flow`s "Start Time" field. This process is repeated for each analyzed packet. In addition to that, flows are occasionally terminated and removed from the active flows list. The list of active flows must be checked periodically whether there are any expired flows, either by "flow idle timeout" or by "flow activity timeout" – the predefined external parameters controlling the lifetime of a flow:

- **flow idle timeout** - maximum timespan allowed between 2 subsequent packets in a single flow: if the idle timeout has passed, but no new packet registered in the flow, it is considered idle and deleted from the pool of active flows. Figure 2 demonstrates the termination by flow idle timeout.
- **flow activity timeout** – maximum time interval during which a flow is considered to be active. If the currently constructed flow lifetime exceeds this value, it is marked as inactive, and a new flow is created. Flow termination by the activity timeout is shown in the Figure 3.

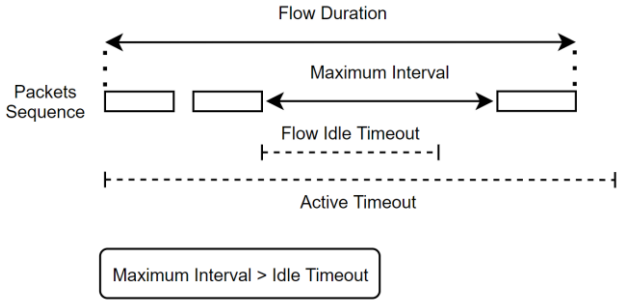

Figure 2: Flow termination by its idle timeout

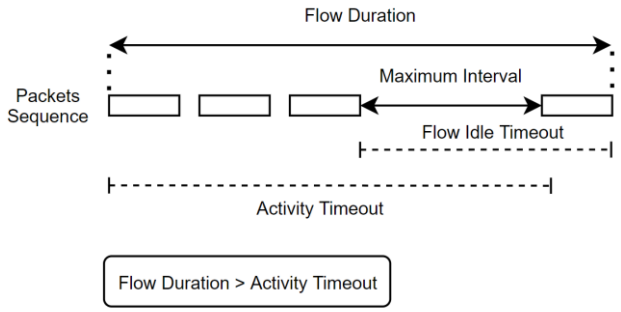

Figure 3: Flow termination by its activity timeout

Besides the timeout parameters, TCP connections have some additional termination conditions (see Figure 4):

- The most frequent is the "graceful disconnect" flags sequence: **FIN-ACK-FIN-ACK**. If the first FIN flag is encountered, the algorithm searches subsequent ACK-FIN-ACK flags and then marks flow as finished.
- Another case of TCP flow termination is when **RST** flag is encountered, which means connection was abruptly reset by one of the peers.

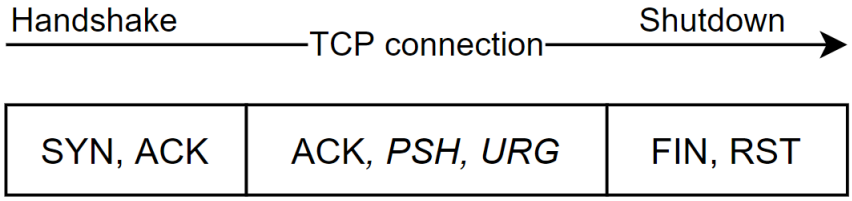

Figure 4: TCP connection lifecycle and the corresponding flags

For UDP and ICMP flows idle and activity timeouts are the only termination conditions as they have no other mechanism for conversation termination.

5. Flow representation formats

In practice, to construct a network flows intrusion detection pipeline it is essential to select a descriptive format for flow`s features representation and pick a corresponding tool for extracting those features.

Some fundamental work of flows format analysis and feature selection was done by Mark Graham in 2018 [5]. The work outlines two major flows representation, which are **NetFlow** (different versions) and **IPFIX (IP Flow Information eXport)**.

NetFlow is a technology developed and patented by Cisco in 1996 and built-in most of their routers. The first implementation (V1) was restricted to IPv4 traffic and contained very few fields associated with a flow. Later, after a few internal iterations, version V5 was released which is the most used nowadays along with V9. V5 has 18 static fields including header ones characterizing the flow. Unfortunately, only 10 are useful for intrusion detection. The final version of NetFlow is V9, which extended the static fields count to 79 (104 for Cisco devices) and added support of the templates, so collected flow info format could be customized according to the needs: some fields could be added, and others could be excluded.

There are two major problems with NetFlow in the context of IDS: 1) it is proprietary 2) it was not originally designed as a traffic monitoring tool for intrusion detection, but rather for network resource consumption and performance analysis or simply network management.

IPFIX (aka **NetFlow V10**, even though has nothing to do with the original Cisco NetFlow) is a standardized technology for capturing network flows. It has 433 information elements, 79 of which match NetFlow version 9 for the sake of compatibility. Like NetFlow V9 it is template-based, so the user can choose a specific set of fields to capture and report. Just as V9 it supports IPv6, multi-cast, and MPLS. IPFIX is vendor-neutral and standardized in RFC 7011. One major difference of IPFIX from NetFlow is that it supports application-layer (L7) protocol analysis as well (e.g., HTTP and IRC), which is crucial for threat detection. According to Mark Graham, 2018 [5] IPFIX is the preferable format for intrusion detection.

Despite NetFlow and IPFIX can provide reach and extensible flow representation, they turned out to be hard to obtain from the existing datasets. To process the ISCX Botnet dataset 2014 [4], which is provided as PCAP files (Wireshark-compatible packet captures), a tool was needed to convert packets from those files into flows (either in IPFIX format or any other). Although IPFIX is an open technology, most of the tools capable of producing IPFIX-formatted flows from packet captures are proprietary (e.g., Nprobe). So, for the task of flow reconstruction and primary feature extraction CICFlowMeter V4.0 [6] was chosen, which is an open-source tool developed by the Canadian Institute of Cybersecurity. It can construct flows from packets both offline (from PCAP files) and in

real-time (by network sniffing). The output flows are presented in CSV format. Each constructed flow contains 83 information fields.

6. Flow features extraction

Out of 83 flow fields obtained by CICFlowMeter 6 fields (namely, Source/Destination IP, Source/Destination Ports, Protocol, Start Timestamp) were used for a flow identification. Another 72 feature-field were utilized for classification, namely: *Flow Duration, Fwd/Bwd Header Length, (Fwd/Bwd) Packet Length Min/Max/Mean/Std/Total, Total Fwd/Bwd Packets, (Fwd/Bwd) Inter-Arrival Time Min/Max/Mean/Std/Total, (Fwd/Bwd) SYN/FIN/ACK/RST/CWR/PSH/URG/ECE flags count, Packets/second, Bytes/second, Flow Active Duration Min/Max/Mean/Std, Subflow (Fwd/Bwd) Packets/Bytes, Up/Down Ratio*.

The remaining fields have been left unused due to occasional faults in their calculation by the tool. Note: Since the flows were considered bidirectional, Fwd references packets sent forward with the relation to the first packet initiating the flow, Bwd - packets sent backward accordingly.

7. Features preprocessing

At this point, it is already possible to train the flows classification model based on the feature fields obtained from the previous stage. However, to obtain reasonable results, additional preprocessing must be done.

Firstly, it is important to clean up the data. In practice, packets timestamps do not always correspond to the order packets arrive and being processed, which may cause feature calculation anomalies (if such scenario is not foreseen by the flow reconstruction tool). For example, after processing the ISCX Botnet dataset 2014 with CICFlowMeter, some fields like *Flow Duration* or *Inter-Arrival Time* had negative values. Such flows were discarded as not valid.

Secondly, great attention must be paid to data normalization and standardization. Due to the nature of the selected features, their values may vary immensely – both, comparing to themselves and comparing to each other. For instance, *Flow Duration* may vary from several milliseconds to several hours. Such heterogeneity negatively affects the model's ability to train. To compensate for that, values must be normalized. In this paper, min-max scaling (1) was used for each feature individually, so its final value is in the range between 0 and 1.

$$x_{std} = (x - x_{min})/(x_{max} - x_{min}), \quad (1)$$

In addition, counters fields, like TCP flags count, are several orders of magnitude smaller than, other fields and may be overwhelmed by the fields of much greater value, like *Packet Length* or *Flow Duration.* Features standardization addresses this issue, by subtracting the mean value and scaling to the standard deviation of each feature. In the case of logistic regression classifier, such manipulation has helped improve detection AUROC metrics from 0.778 to 0.7997.

8. Possible improvements in feature selection

It is worth mentioning, that despite those statical features characterizing generic network flows yield quite good detection rates (*AUROC 0.918* on unseen data), crafting other features specific for botnet detection may significantly improve the results.

Employing cluster and correlation analysis on real-world botnet data, Mark Graham, 2018 [5] concluded that the features listed in Table 1 contribute the most to successful botnet discovery. Even though some fields are purely utilitarian and obvious (e.g., IP addresses), others, like L7 flags (DNS, HTTP, SMTP, IRC), are closely related to the ways botnets often communicate. For instance, known techniques involving DNS lookups, which are used by infected machines to discover Command and Control (C&C) botmaster, namely "Fast flux" [11] and "Domain Generation Algorithm" (DGA) [12]. Studying the specifics of botnet functioning and their intrinsic network pattern and heuristics may help select feature sets for more effective botnet discovery. The same statement is true for the discovery of other network attacks.

Table 1
IPFIX features for botnet detection template [5]

Name	Description
srcIPv4	Source IPv4
dstIPv4	Destination IPv4
srcPort	Source Port
dstPort	Destination Port
packetTotal	Total Number of packets transmitted
flowEndMS	Timestamp of when the flow ended
flowStartMS	Timestamp of when the flow started
protocol	OSI protocol used (TCP, UDP, ICMP etc.)
initTCPFlag	Flags exchanged during while establishing TCP connection
tcpSeqNos	TCP sequence numbers
collectorIPv4	IPv4 address of flows collector
flowKeyHash	Hash of flow identification tuple + time. Used as Flow ID
ircTextMessage	The text and command sequences in case of IRC communication
httpGet	Whether the flow contains HTTP GET request
httpResponse	HTTP response code
dnsARecord	Flow contains domain resolution of A type record
dnsSOARecord	Flow contains domain resolution of SOA type record
smtpHello	Flow contains SMTP ELHO or HELO commands
sslName	Details of SSL certification authority

9. Network flow classification techniques

There are several different ways to perform the classification of the captured network traces. The most basic classification unit is an isolated network flow itself. Since each flow is already labeled as malicious or benign, flow feature vectors can be fed into the classifier one by one to learn whether it is a part of a botnet communication based on the data encoded into its feature values alone. If each flow vector contains ***F features***, the dimensionality of the classifier input and the dimensionality of the full input matrix during training is N×F, where N is the number of flows in the training dataset. This method is used as the baseline to compare against. The best performance was shown by Deep Neural Network (DNN) with 3 hidden layers 16x16x16 (AUROC - 0.866). Random Forest classifier took second place (AUROC - 0.848) and Logistic Regression took third (AUROC - 0.778).

The second more sophisticated approach for classification is centered around the idea of treating the flows set as time series and analyzing each flow in the context of the prior flows encountered. To achieve this, each flow`s timeframe is considered – all flows are sorted based on the flow start timestamps forming time series of flows. After that, different techniques can be used for those series processing. Flows can be split into groups by similar time intervals or into chunks with a fixed flows number. Later, those chunks are joined into one vector and passed into the classifier. Grouping arbitrary flows, however, makes the prediction results quite vague, so besides grouping based on time sequencing only, it is reasonable to aggregate flows based on their intrinsic data, like source/destination IP couple, ports, and protocols. Since an attacker can change communication port

and protocol regularly, the grouping network flows by source and destination IP addresses may capture the whole bot-to-C&C communication occurring during a timespan. Further, classification is done on each flows group rather than on an isolated flow, while still treating a group as a time series of flows (for reference see Figure 5).

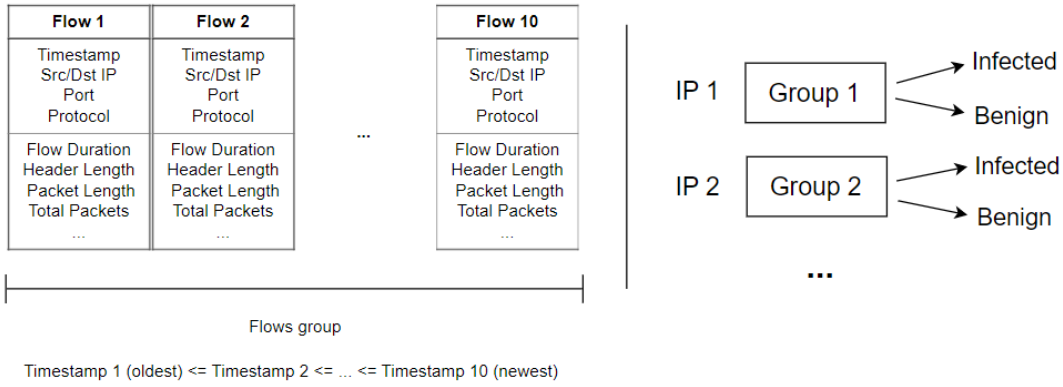

Figure 5: Groups of ordered flows

By combining these two approaches, it should be possible to perform a reasonable judgment on the communication nature as it regards the context of each flow in both time and event spaces. In such case, the classifier takes an input as a M×F matrix, where F is the number of features and ***M - flows group size***. Input training tensor is K×M×F, where K is the number of groups. For the classification of such time-series data Convolutional Neural Networks are used with temporal convolutions. Such an approach was taken by Arnaldo et al. in 2017 [3], where they have originally described its application for log-based (connection-based) data, but later applied to network flows data, obtained from ISCX 2014 Botnet dataset. In their case, unfortunately, that did not yield any positive results: the classification performance was worse than in the case of individual flow analysis. They did, however, manage to improve results slightly (by 5.6%), by extending the original feature set with generated features.

In this paper, we used a similar approach, but with some enhancements and modifications. Firstly, the base feature set for each flow was extended from 23 to 72, which makes the input data much more representative. Secondly, the grouping was done based on a Source IP address only. The motivation for that we want to identify the infected host itself, which may be sending C&C discovery requests to different remote IP addresses before finding an available one. An attempt to establish C&C connection is known to be inherent for all botnets. Finally, it seems when considering groups with more than one flow Arnaldo et al. [3] discarded groups containing fewer than N flows. So different classifiers were trained and used for different cases (N = 7, 14, 28). In contrast, in the scope of this paper all series were split by static groups of size 10 and for all the groups containing fewer samples, the input tensor was simply padded with zeros, which is a common technique applied when using convolutions on data of varying size. Together with the features preprocessing and filtering techniques described in the previous chapters, using a CNN classifier on the groups of time-series yielded significantly better results in comparison to the baseline single-flow DNN classification. AUROC metrics increased by 6% (0.918), which indicates the effectiveness of considering multiple related flows in the context of the order of their occurrence. CNN model contained 2 convolutions layers and two 1-dimensional pooling layers. The first convolution layer had its kernel size of 3 with 32 filters, the second one had the kernel size of 2 with 16 filters. Simplified version of the model is shown on Figure 6.

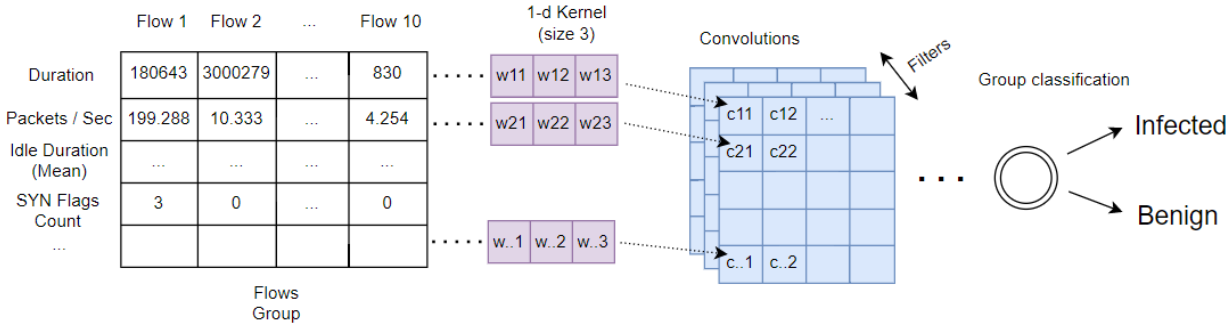

Figure 6: Network flows group classification with CNN

This approach, however, has a major drawback – it is not practical to be used for real-time (semi real-time) classification, as it essentially needs the whole group of flows to be analyzed altogether.

Another commonly known approach towards time-series data processing is based on ingesting events in order of their appearance one by one, but at each step taking into account the previous events. Firstly, a vector of "historical features" is extracted by applying the model to the events on the past timesteps (in our case, applying to the previous flows in the group). This vector represents historical context for the currently analyzed flow. Then the historical features are combined with the current features sample and fed into the classifier together, which makes the judgment based on both. The classifier input F_i can be described as (2).

$$F_i = F_{historical} \cup F_{current} = (x^h_1, \dots, x^h_{|F_{historical}|}, x^c_1, \dots, x^c_{|F_{current}|}) \quad (2)$$

where $F_{current}$ - feature vector of the current sample, and $F_{historical} = E(\cup^k_{current-1} F_k)$ – historical feature vector, which, for the sake of reducing the number of calculations, can be presented as (3).

$$F_{historical} = E(\cup^k_{current-1} F_k) = E(F_{historical-1} \cup F_{current-1}) \quad (3)$$

Such an approach can be implemented with Recurrent Neural Network (RNN) with the benefit that a series of feature vectors can be processed element by element at each step producing some new result. The flows groups are processed sequentially. It keeps track of the dependencies between a series of events and recurrently passes the values obtained on the previous step (historical features) to the input of the current step. In other words, such classifiers can "memorize" the prior states and take those experiences into account while doing future classifications. In this paper, long short-term memory (LSTM) was used as RNN architecture, which is more sustainable to vanishing gradients problem in comparison to vanilla RNN. AUROC metrics reached the value of 0.907, which is 2.37% better than plain single-flow analysis, but still worse than CNN.

10. Classification models

After analyzing the obtained experimental results of different AI model performances for botnet detection it is worth outlining the summary for each model – the reason why they succeed or fail and their possible usage cases.

Logistic regression can be used as a baseline classifier to compare against. Due to its simplicity, it is relatively quick to train and has the best real-time performance. Besides, the trained weights can be further interpreted to get the understanding of which features contribute the most to the decision process. Logistic regression classifier gives the probability as an output, so simple threshold tuning can be used for reducing the number of false positives, if needed, which is valuable for intrusion detection systems. Logistic regression classifiers can be used for real-time decentralized IDS, with limited resources capacity on each node. During the experiments, the logistic regression classifier showed the base performance of AUROC 0.778 with the accuracy of 0.78. Applying normalization and standard scaling (described in chapter 7) substantially improved the results: AUROC rose to 0.799 and accuracy increased to 0.722. Dimensionality reduction of the input data with Principal Component Analysis (PCA) helped to improve the results even further (see the numbers in Table 2).

In theory, decision trees are among the best candidates for the most sophisticated classification of network activity. The intuition is that every feature of the analyzed network activity has some

interpretable semantic meaning and is usually included based on the prior knowledge of the common attack patterns. For botnet detection based on network flows several examples of such features are: “was IRC protocol used?”, “were type-A DNS lookups made and how many?”, “what was the number of TCP connection denials (no proper response to SYN flag) and resets (RST flag)?”, “how much data was transmitted before the reset?”, etc. A human expert would use all this information to denote a flow or a host as “suspicious” depending on the answers to those questions. Decision trees are capable of automatically constructing the “reasoning chains” based on the learned abstract conditions, which may remotely reflect the human reasoning sequence (e.g., *if X and Y then if Z then … else …*). This classifier type has performed well in a number of works in this field [7][8][9]. However, decision trees suffer from overfitting the training set, which is a big problem for intrusion detection, as training sets are often small, artificially generated, and include a limited number of botnet/attack traffic samples. This was demonstrated in the experimental part of this work, where AUROC value for decision trees classifier was only 0.477 (unless Principal Components Analysis was preliminarily applied).

To address that issue, random forests can be used instead. By sacrificing accuracy and bias, random forests have much lower variance. Citing Hastie et al. [10], "because it is invariant under scaling and various other transformations of feature values, is robust to the inclusion of irrelevant features and produces inspectable models. However, they are seldom accurate". Random forests are based on the decision trees classification technique, but instead of using a single tree prone to training noise and overfitting, they use an ensemble of trees. During training, a random uniform sample with the replacement of data features is used for each tree. In the case of solving the regression problem, prediction averaging is performed among each decision tree to learn the result. For a classification problem, “voting” (selecting the class of the majority) is used. This gives the random forest classifier a significant advantage over the decision tree (AUROC 0.848 and AUROC 0.91 if combined with PCA). This is especially significant for network attacks detection problems, as there are not many publicly available labeled datasets and those which are available are often very moderate. This makes the overfitting problem even more significant for traffic classification.

A single flow is often not sufficient to reconstitute the full picture of the attack. So single-flow classification techniques may fail to detect intrusions stretched in time. Unlike the previously described models, Convolutional Neural Networks and Recurrent Neural Networks can take contextual information into account.

CNN performs sampling (convolution) of different features/events into a higher-level entity. Such sampling can be done over the same feature values changing in time. Such 1-dimensional sampling is called temporal convolution. This allows capturing the temporal nature of a feature across multiple associated network flows, which results in better detection performance. The drawback of CNN is that it operates on statically sized groups of events. So, to obtain the final prediction result, the full chain of flows must be captured, which makes them inapplicable for immediate intrusion detection, but very useful for analyzing historical data.

LSTM models can accept sequences of arbitrary length, producing updated classification result for each new flow ingested. The detection accuracy may be low initially but gradually raises as the models get more and more historical context.

11. Related works and models performance comparison

Table 2 demonstrates the performance of different network activity classification techniques obtained during the work on this paper. For reference, the table also provides similar metrics obtained by a few other researchers who used ISCX Botnet 2014 dataset [4] and the same feature extraction technique based on network flows. The results for similar AI models to the ones applied in the scope of this paper were selected, for the sake of making a direct comparison.

Table 2
AI models on ISCX Botnet 2014 dataset performance comparison

Paper	Feature Selection and Dimensionality reduction	Classifier	Performance on unseen data

This paper.	(1 flow)	Logistic Regression	AUROC 0.778 (0.799), Accuracy 0.648 (0.722)
	(1 flow)	Decision Tree (d9)	AUROC 0.477, Accuracy 0.65
	(1 flow)	Random Forest (d9)	AUROC 0.848, Accuracy 0.798
	PCA (16) (1 flow)	Logistic Regression	AUROC 0.804, Accuracy 0.78
	PCA (16) (1 flow)	Decision Tree (d9)	AUROC 0.844, Accuracy 0.833
	PCA (16) (1 flow)	Random Forest (d9)	AUROC 0.91, Accuracy 0.87
	(1 flow)	DNN (16x16x16)	AUROC 0.866
	Grouped by Src IP (10 flows, 0-padded)	CNN (2 conv + pool)	AUROC 0.918
	Grouped by Src IP (10 flows)	LSTM (100 cells)	AUROC 0.908
Arnaldo et al. [3]	PCA (1 flow)	Random Forest	AUROC 0.769
	(1 flow)	Random Forest	AUROC 0.768
	PCA (28 flows flattened) + generated features	Random Forest	AUROC 0.811
	(1 flow)	DNN	AUROC 0.724
	Grouped by Src IP and Dst IP (7 flows)	CNN	AUROC 0.644
	Grouped by Src IP and Dst IP (7 flows)	LSTM	AUROC 0.624
Meshal Farhan et al. [7]	MUTUAL* (1 flow)	Random Forest	Accuracy 0.81
	MUTUAL* (1 flow)	Logistic Regression	Accuracy 0.748
	ANOVA** (1 flow)	Decision Tree	Accuracy 0.73
	MUTUAL* (1 flow)	DNN	Accuracy 0.736

** "Mutual information analysis"*
*** "Analysis of variance"*

12. Conclusion

This paper has described a structured end-to-end AI-based solution to the botnet detection problem. The outlined intrusion detection pipeline is not scoped to botnets only and represents a solid foundation for future research on the topic of any network-based intrusions. The performance metrics given in this paper can be used as the baseline for future improvements and conducting research on each separate stage of the pipeline should result in further detection performance increase.

13. References

[1] H. Debar, An Introduction to Intrusion Detection Systems, IBM Research, Zurich Research Laboratory, Zurich, Switzerland, 2009. URL: https://www.researchgate.net/publication/228589845_An_Introduction_to_Intrusion-Detection_Systems.

[2] K.P. Al-Sakib, The State of the Art in Intrusion Prevention and Detection, 2014. O'Reilly Media. URL: https://www.oreilly.com/library/view/the-state-of/9781482203523/.

[3] I. Arnaldo, A. Cuesta-Infante, A. Arun, M. Lam, C. Bassias, and K. Veeramachaneni, Learning representations for log data in cybersecurity. In: Proc. Conference on Cyber Security Cryptography and Machine Learning, 2017, pp. 250–268. doi:10.1007/978-3-319-60080-2_19.

[4] E.B. Beigi, H.H. Jazi, N. Stakhanova, A.A. Ghorbani, Towards effective feature selection in machine learning-based botnet detection approaches. In: 2014 IEEE Conference on Communications and Network Security, 2014, pp. 247–255.
[5] M. Graham, A Botnet Needle in a Virtual Haystack, Anglia Ruskin University, East Anglia, England, 2018. URL: https://arro.anglia.ac.uk/id/eprint/702723/1/Graham_2017.pdf
[6] A. Habibi Lashkari, CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyzer for anomaly detection, 2018, doi:10.13140/RG.2.2.13827.20003. URL: https://github.com/ISCX/CICFlowMeter.
[7] M.F. Al-Anazi, M. G. M. Mostafa, Efficient Botnet Detection using Feature Ranking and Hyperparameter Tuning. In: International Journal of Computer Applications (0975 – 8887) Volume 182 – No. 48, 2019, pp. 55-60. doi:10.5120/ijca2019918739.
[8] S. Almutairi, S. Mahfoudh, S. Almutairi, Jalal S. Alowibdi, Hybrid Botnet Detection Based on Host and Network Analysis. In: Journal of Computer Networks and Communications, 2020, pp. 1-16. doi:10.1155/2020/9024726.
[9] P.A.A. Resende, A.C. Drummond, HTTP and contact-based features for Botnet detection, 2018. URL: https://doi.org/10.1002/spy2.41.
[10] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning (2nd ed.), 2009. Springer. ISBN 0-387-95284-5. URL: https://doi.org/10.1007/978-0-387-84858-7.
[11] C.H. Hsu, C.Y. Huang, K.T. Chen, Fast-Flux Bot Detection in Real Time. In: Proc. Recent Advances in Intrusion Detection 13th International Symposium (RAID 2010), Ottawa, Ontario, Canada, 2010, pp. 464-483. doi:10.1007/978-3-642-15512-3_24.
[12] C. Hwang, H. Kim, H. Lee, T. Lee, Effective DGA-Domain Detection and Classification with TextCNN and Additional Features. Electronics, 9(7), 1070, 2020, doi:10.3390/electronics9071070.