CyberVTI: Cyber Visualization Tool for Intrusion Detection

Pedro Marques[1,2] Luis Dias[1,2] Miguel Correia[2]
[1]*CINAMIL, Academia Militar* – Lisboa, Portugal [2]*INESC-ID, Instituto Superior Técnico* – Lisboa, Portugal
marques.pmb@exercito.pt dias.lfxcm@exercito.pt miguel.p.correia@tecnico.ulisboa.pt

***Abstract*—This paper presents the Cyber Visualization Tool for Intrusion detection (CYBERVTI), a tool that provides intrusion detection algorithms with an effective graphical interface. The goals of the tool are twofold. First, it aims to aid the human analyst on observing ongoing threats as a step towards responding to these attacks. Second, it aims to help the analyst study the performance of the algorithms with different parameters, in order to decide which combination best suits his goals. The tool integrates a set of recently-proposed intrusion detection algorithms based on unsupervised machine learning, which it aims to help configuring. CYBERVTI follows a client-server system architecture and implements a REST API that mediates the communication between the server and the GUI on the client-side. A significant challenge is to define data structures that allow storing and accessing data efficiently in an interactive manner. A clear and minimalist visual aspect was chosen for the GUI, containing functionalities that enable the user to analyze the network data in a simple and clear way. For system evaluation, we used the ISO/IEC 25010:2011 requirements. For validation, we compared the results obtained by our tool with the results described in the original papers that present the algorithms.**

Keywords: Cybersecurity, Machine Learning, Intrusion Detection System, Security Analytics, Visualization Tool

I. INTRODUCTION

The amount of digital data generated in the world continues to grow, even more so at this time when remote work and education have become part of the reality of many people [11]. This growth in the Information Technology (IT) field combined with an environment of constantly changing risks and threats make cyber security essential to us all. Therefore, to protect individuals and organizations more effectively, it is necessary to identify threats using *Intrusion Detection Systems* (IDSs) [2].

With the evolution of the Machine Learning (ML) field, the number of IDSs that use unsupervised ML, e.g., anomaly-based IDSs, has been growing [1], [14], [18], [5], [20], [3], [16], [7], [8], [10]. IDSs that use such algorithms do not need to have signatures of the attacks to detect them, which is an advantage in comparison to knowledge-based (or signature-based) IDSs, as it allows detecting new attacks.

IDSs can be enhanced with *visualization tools* for faster and more immediate verification of the results obtained by the algorithms they use [26], [19], [21], [12], [13], [30]. The visualization tool can become a key component of the IDS, allowing the analyst to visualize graphically the information extracted from the potentially large volume of monitoring data being collected and produced. Therefore, the goal of this work is to design and implement a graphical tool to simplify the activity of those responsible for the security of computer networks.

978-1-6654-9550-9/21/$31.00 ©2021 IEEE

The paper presents the Cyber Visualization Tool for Intrusion detection (CYBERVTI). The goals of the tool are twofold.

First, it aims to aid the human analyst in observing ongoing threats as a step towards responding to these attacks. It aims to be a graphical tool for the automatic detection of threats in cyberspace that is intuitive and easy to use.

Second, it aims to help the analyst study the performance of the detection algorithms with different parameters, in order to decide which combination best suits his goals. To this end, it provides the user with an abstraction of the event analysis processes, allowing the optimization of the utilization of the algorithms applied for intrusion detection.

The tool integrates a set of recently-proposed intrusion detection algorithms based on *unsupervised ML*, specifically on *clustering* techniques [3], which it aims to help configuring for achieving high performance: FLOWHACKER [25], OUTGENE [7], and DYNIDS [8]. The initial version includes these three algorithms, but it is extensible to as many as needed.

CYBERVTI follows a client-server architecture. The server-side / back-end exports a REpresentational State Transfer (REST) Application Programming Interface (API) that allows accessing the data that it stores and processes [24]. The API is invoked by the client / front-end, that provides the visual interface and with which the analyst interacts – the Graphical User Interface (GUI).

An important challenge is to define data structures that allow storing and accessing data efficiently. Specifically, we wanted the analyst to have a comfortable user experience when accessing and requesting the processing of data in an interactive manner, avoiding long waiting times for data to be processed. As far as the GUI is concerned, a clear and minimalist visual aspect was chosen, containing functionalities that allow the user to analyze the network data in a simple and clear way.

For system evaluation, we used the theoretical requirements for system and software evaluation of the ISO/IEC 25010:2011 standard. For validation, we compared the results obtained by our tool with the results described in the original papers that present the algorithms.

II. Background

This section presents the background necessary to understand the intrusion detection algorithms that are used in CyberVTI: clustering and the three algorithms currently integrated (FlowHacker, OutGene, and DynIDS).

A. Clustering

The objective of this class of ML algorithms is to find patterns in multidimensional unlabeled data. These patterns are found after the data is grouped based on a measure of similarity [3]. The clustering process that analyses the input data and leads to its classification, can be divided into four steps [29]: i) *Feature selection and extraction* that chooses and normalises the different features of a candidate set that are important for clustering; ii) *Clustering algorithm selection* that uses a criterion function to group the data points according to the similarity of their characteristics; iii) *Cluster validation*, when standards and evaluation criteria are applied to provide a degree of confidence on the pooling results; and iv) *Interpretation of results* that generates knowledge and relevant information from the initial data.

Clustering algorithms can be categorized by the methods they use, which can be: partition, hierarchical, density, and neural. K-Means, the cluster algorithm currently supported by CyberVTI, is a partition algorithm [28]. K-Means is used to partition a dataset into k groups automatically. It proceeds by selecting k initial cluster centres and then iteratively improving them in two phases: i) each instance d_i is assigned to its closest cluster center; and ii) each cluster center Cj is updated to be the means of its constituent instances. It converges when there is no further change in the assignment of instances to clusters. K-Means has a linear complexity, $O(nkdi)$, where: n is the number of points to consider; k, number of clusters generated; d, data-point dimension; i, the number of iterations until convergence.

B. Algorithms: FlowHacker, OutGene, DynIDS

FlowHacker [25] is a network intrusion detection algorithm that uses anomaly-based techniques. It implements unsupervised ML, specifically clustering (K-Means), to group machines with similar network behaviours. Potential threats are grouped in different clusters from the remaining entities, so hosts or users with anomalous behaviour are in isolated clusters and can be identified. FlowHacker uses 28 features divided into two groups. Half of the features are related to the source computer, and the other half are the same but relative to the destination computer. Eight of the 14 features consider the bidirectional flow (to and from ports) of 4 application-layer protocols, which are: HTTP (port 80), IRC (port 194), SMTP (port 25), SSH (port 22), and IRC (port 6667). The remaining features are: number of connections made, number of ports used by the source, number of ports contacted by the source, the sum of bytes sent by the source, and the sum of packets sent by the source.

OutGene [7] has several similarities to FlowHacker but is introduces two new concepts: i) *genetic zoom*, using a genetic algorithm that identifies the best subset of features leading to the formation of the same clustering output that all features would generate; and ii) *time stretching*, to detect stealth attacks that use a low execution pace, analyze the flow of events in different time windows at different time scales. OutGene removes the two features associated to port 6667 (IRC) as it is less used today.

DynIDS [8] is another network intrusion detection algorithm that uses anomaly-based and clustering methods to group hosts with similar behaviour and detect threats. This behaviour is characterized by features extracted from network flows.

The main innovation of DynIDS [8] is the dynamic (runtime) selection of features based on the observed traffic flow. This algorithm uses 12 static features, half relative to the source and half relative to the destination. These static features are: the number of different IPs contacted by an entity, number of flows where the entity is the source, number of different source ports used, the number of different destination ports contacted, sum of total packet length receiver, and the sum of total packets length sent. However, it also uses port-based features obtained dynamically, according to the data analyzed in each time window. DynIDS uses four features for each port (number of packets sent and received in a port, source and destination host). The dynamically defined features are identified based on the DYN3_X algorithm that filters at the $x/3$ ratio the ports that appear most in the flows, the ports that appear in fewer flows and the ports used by fewer machines. This approach does not limit the system's ability to detect attacks related to specific ports because the features are correlated with the traffic flow in the network.

DynIDS [8] also applies the concepts of genetic zoom and time stretching. Regarding clustering, it applies 3 clustering algorithms: K-Means (partition-based), Agglomerative (hierarchical) and DBSCAN (density-based). In K-Means, the elbow method is used, where several k (number of clusters) are tested until the optimal number of clusters (k_{OPT}) for grouping the data. This way, it is possible to obtain a better performance in identifying outliers, which is done by intercepting the results of the three algorithms.

III. CyberVTI

CyberVTI is a visualization tool for intrusion detection. It is based on a client-server architecture, using a REST API [24]. We chose this approach to remove the data storage and complex processing from the client-side, supporting many users while hiding from them the complexity of the server-side. The algorithms and the database (DB) are on the server-side with more computing power to do the processing and with more storage capacity. We use REST because it is a simple and widely-adopted request/response mechanism. Its calls are message-based and follow the HTTP standard. We integrate the network intrusion detection algorithms in the back-end of our system, and the communication is made through HTTP(S) requests. The results obtained are sent to the client and are displayed in the GUI, which is local.

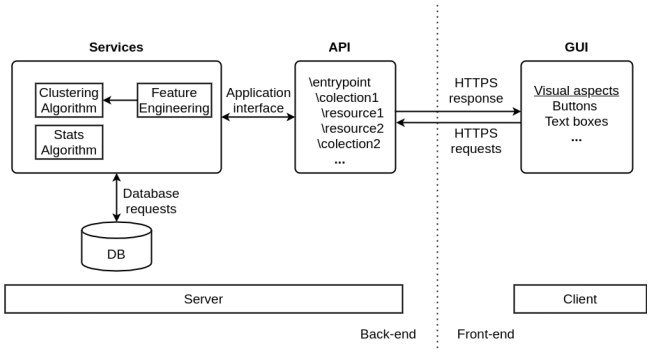

Fig. 1. CYBERVTI architecture

A. Architecture and Data Flow

The architecture of CYBERVTI is presented in Figure 1. The server-side, or back-end, does the processing and storing. It is composed of a database and a data processing module (Services in Figure 1), where the algorithms responsible for feature engineering, clustering, and obtaining statistical data are executed. The API is also implemented on the server-side.

CYBERVTI uses network monitoring data to detect intrusions. This data can be collected by routers, by tools such as *tshark* and *Wireshark*, or taken from datasets that can be used to study the performance of the detection algorithms. The data flow in CYBERVTI (in the server-side) is the following:

1) Data on-boarding: the data is loaded, filtered (to remove unnecessary data), and input files are generated (.csv).
2) Data preparation: data is verified and validated.
3) Feature engineering: the features are selected, extracted, and normalized.
4) Data processing: the statistics acquisition and clustering algorithms are applied.

After steps 2, 3 and 4, the data are stored in the database and in a specific directory for the features on the server-side.

B. Technologies

CYBERVTI is written in Python (version 3.8), and all the tool code (GUI and back-end) is available on the A3CE Project public repository on GitHub[1]. The intrusion detection algorithms that are integrated into the tool are also implemented in Python. They are based mainly on the *Pandas* and *Scikit-learn* libraries.

For implementing the REST API, we used *Flask* [17] [23], which is a web microframework. Its main purpose is to provide a simple and extensible application core. This allows the developer to choose which systems to integrate and how.

The DB was implemented in *SQLite* [22]. This is an open-source embedded relational database designed to provide a convenient way for applications to manage data. *SQLite* has the main goal of being highly portable, easy to use, compact, efficient, and reliable. To manage and interact with DB, we use *SQLAlchemy* as an interface between *SQLite* and Python. *SQLAlchemy* [6] is a Python library that provides a high-level Python-style interface to relational DBs. To save the results of the feature extraction, *Parquet* files were used. *Parquet* [27] is an open-source file format available that offers columnar data storage format different from row-based files such as .csv files. *Parquet* is optimized to work with complex bulk data and has different ways of efficiently encoding data.

In the front-end, *Tkinter* was used for all graphic development of the visualization tool. *Tkinter* [15] is a standard Python interface based on a thin object-oriented layer on top of Tcl/Tk[2] for building a GUI. However, we used *matplotlib*, a Python package that allows constructing graphics efficiently. It is also allowed to interact and change the visualization aspects of the graphs and insert information that leads to a better perception of the displayed data.

C. Graphical User Interface

The GUI offers an abstraction layer that allows any type of user to use the CYBERVTI. The user can visualise and analyse the data more effectively as it is presented graphically. It is clearly simpler than the common approach of executing scripts through the command terminal. Features were added to the system to obtain relevant data for the user and to provide flexibility in data analysis.

The first time the CYBERVTI GUI is opened, the user will not find any files loaded. The GUI will present two buttons, one to upload a file and the other to refresh the view after uploading. When the user has files on the application, a list of filenames is displayed.

When a file is loaded in the application, several default analyses are performed immediately, in order to speed up the requests the analyst may do. At this stage, the user can also define new analyses to be done immediately. Figure 2 shows the view at this point. The view is divided into 5 parts corresponding to its function:

1) Name of the selected file and a graph with the number of events (all the net flows) in the file. The x-axis is the timestamp of the events.
2) Drop-down menu and an input text box to choose the time interval for new file analysis.
3) Input text box allows the user to choose the time window for the analyses (or several, separated by commas), and radio buttons to select the method that will be applied to extract the features. The currently supported methods correspond to the features of each of the three algorithms supported, to the features combination of DYNIDS and OUTGENE, and to a free selection (the user chooses the list of ports to analyse).
4) Back button to go to the initial view.
5) Three buttons that allow: starting the analysis, viewing the event graph in more detail, and viewing the results obtained.

[1]Repository url: https://github.com/a3ceProject/CyberVTI

[2]Open-source widget toolkit that provides a library of basic widgets for GUI. Adapted from: https://www.tcl.tk/

Fig. 2. CYBERVTI analysis view.

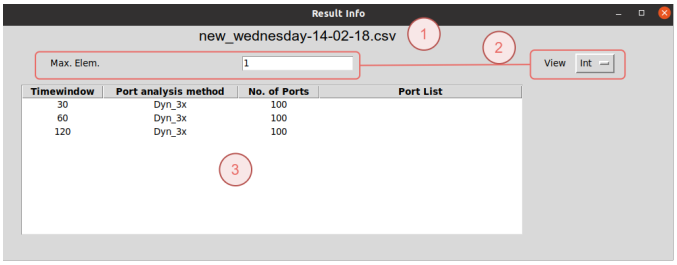

Fig. 3. CYBERVTI results first view.

Regarding the visualisation of the results, the first view the user sees is in Figure 3. The name of the file is identified by the number 1. In number 2 is the input box to choose the maximum number of elements in the same cluster. To find outliers, the default value has to be left as 1. The user can also select the IP view on the network shown in the drop-down menu. This view can be: internal (Int), only IPs belonging to the network; or external (Ext), only IPs outside the network. Number 3 shows a list containing the parameters of the data processing already performed. With a double-click on a line of the list, GUI displays the respective results view.

After choosing the file analysis, the tool presents a list of results obtained: number 4 in Figure 4. This list comprises the cluster timestamp, the cluster number, the number of machines in that cluster, and the IPs of those machines. In case the maximum number of elements per cluster is 1, the IPs shown are those identified as attackers or victims. This is a simple decision criterium used in many works in the area [7], [8]. Number 5 is associated to three functionalities: i) display the heatmap of the data obtained for the chosen timestamp; ii) save in .csv format in a user directory the clusters obtained from the file through the selected analysis; and iii) present the metrics corresponding to the performance of the analysis performed.

The metrics view is presented in Figure 5. In the zone identified by number 1 are the metrics that were obtained through the total values of the following classifications: True Positive (TP), True Negative (TN), False Negative (FN) and False Positive (FP), the formulas for these calculations are

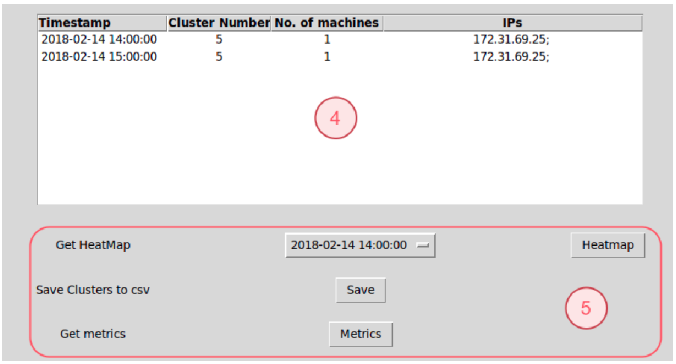

Fig. 4. CYBERVTI analysis second view.

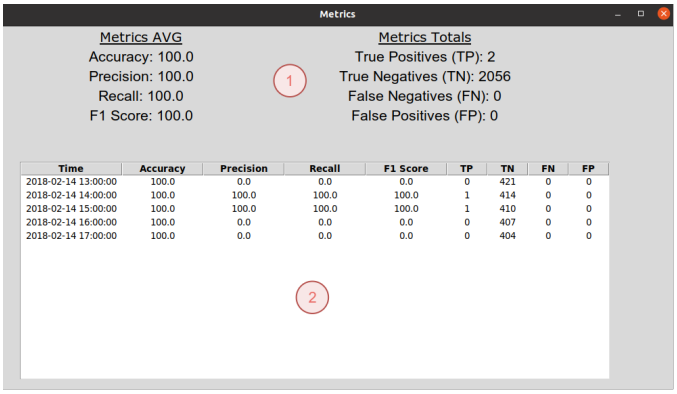

Fig. 5. Algorithm evaluation metrics view.

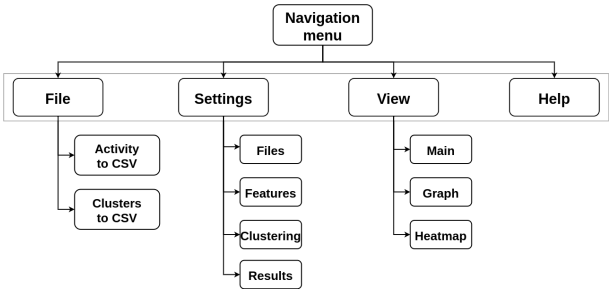

Fig. 6. Navigation menu organisation.

presented in Section IV-B. In number 2, the metrics are shown but divided by their respective analysis intervals.

The GUI of the CYBERVTI (Figure 6) has a navigation menu in the upper part of the window, which is divided into four sub-menus: Files, Settings, View and Help. In the file submenu, it is possible to save the total number of events at certain time intervals and save all clusters of a given analysis into a .csv file on a local path.

The files sub-menu (Figure 7) allows changing the parameters related to the file upload: i) *Frequency*, defines the time intervals in which the number of events in the file are grouped; ii) *Date Format*, timestamp format used in the input file; iii) *Default Time Windows*, list of time windows used for the initial analysis (when the file is loaded); iv) *Default Methods*,

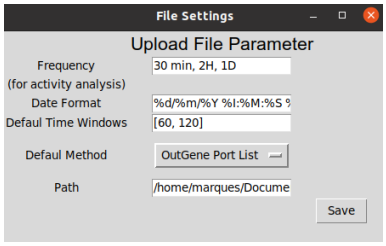

Fig. 7. File settings window.

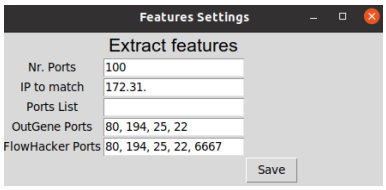

Fig. 8. Features settings window.

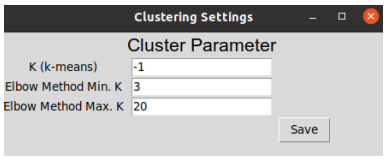

Fig. 9. Clustering settings window.

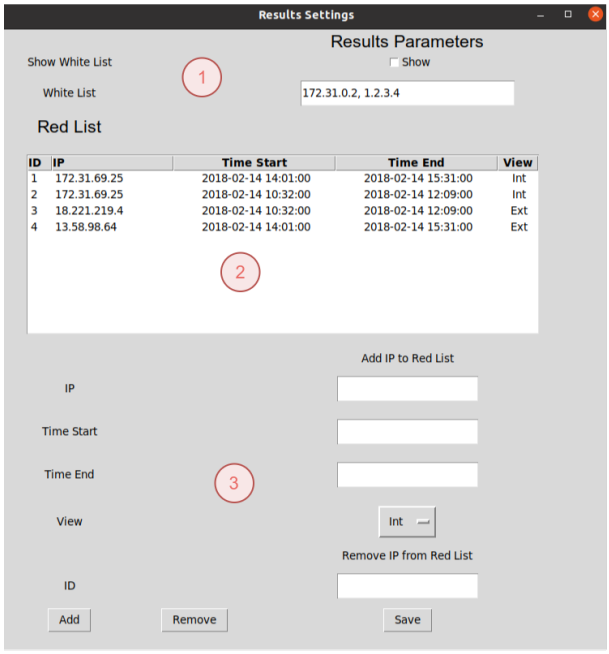

Fig. 10. Results settings window.

the extraction method used to extract features in the initial analysis; and v) *Path*, sets the directory in which the new file upload window will open.

The features sub-menu (Figure 8) allows changing the parameters related to the features extraction: i) *Nr. Ports*, set number of port in a dynamic features analysis (Dyn3_x); ii) *IP to match*, the IP prefix that define external and internal network; iii) *Ports List*, user-defined ports list for a static analysis; and iv) *OutGene Ports* and *FlowHacker Ports*, the default list of ports of the respective intrusion detection algorithms.

The clustering sub-menu (Figure 8) allows to change the parameters related to clustering analysis: i) *K*, number of clusters or -1 to run the elbow method to find the best value for *K*; ii) *Elbow Method Min. K* and *Elbow Method Max. K*, range of values of the number of clusters that the elbow method uses iterations.

In the results sub-menu (Figure 10), the parameters used in the calculation of the performance metrics can be changed. Two lists are used: i) *White List*, IPs addresses of hosts that are not threats and the metric calculation algorithm will not consider this IPs; and ii) *Red List*, IPs addresses of the hosts that the user knows are attackers or victims, and the time intervals in which they are active. Both lists are used to calculate performance metrics. In number 1 are the white list and the option to show its IPs if they are considered threats. In number 2, the red list is shown, the id, the IP address, the start and end time of the attack, and the respective view about the network. In number 3, the user can add or remove an element from the red list.

The view submenu allows configuring parameters related to visual aspects in the main view, the graph of events and the heatmap. In the help sub-menu, the user will have access to the description of the procedures to help him use the graphical interface.

D. Back-end

The resources, services, and functions that allow the tool to be used are implemented on the server-side. An important requirement is to allow efficient access to data. For that reason, it is necessary to have a solid and stable REST API.

For efficient storage management, the first step was to know what data has to be saved and how to do it. The DB implemented in *SQLite* has 4 tables: Users, Files, Features Metadata, Clusters. The Users table is simple and has the user id as Primary Key (PK) and the username that is a unique constraint and cannot be null.

After the upload of the input file in .csv format, the storage of this occurs in two phases: 1) the file is stored in its entirety in a server directory called *Data_files*, where there is a folder relative to the user; and 2) the metadata related to the file is stored in an *SQLite* DB table (Table I).

The *Features_Metadata* table (Table II) stores data about how the features were extracted to avoid repeating the processing already done. Extracted features are stored in a server-side directory in *.parquet* files.

The Clusters table (Table III) is where the clustering results and heatmap data are stored.

The processing module is divided in 3 main components:

TABLE I
INPUT FILE METADATA DB TABLE.

Files		
id	Integer	PK
user_id	Integer	FK
upload_timestamp	Varchar(n)	
file_name	Varchar(n)	
time_view	Blob	
start_time	Varchar(n)	
end_time	Varchar(n)	

TABLE II
INPUT FEATURES METADATA DB TABLE.

Features_Metadata		
id	Integer	PK
file_id	Integer	FK
time_window	Integer	
method	Integer	
nr_ports	Integer	
ip_to_match	Varchar(n)	
port_day_list	Blob	
outgene_ports	Blob	
flowhacker_ports	Blob	
standard_ports	Binary	

TABLE III
CLUSTERS DB TABLE.

Clusters		
id	Integer	PK
file_id	Integer	FK
config_id	Integer	FK
view	Varchar(n)	
ip	Varchar(n)	
c_timestamp	Varchar(n)	
cluster_number	Integer	
data_to_heatmaps	Blob	

- Activity analysis: analyzes and groups the data that the input file contains (e.g., netflow events, with a frequency set by user).
- Feature extraction: this module extracts features from the input file. The first step is to check if features have already been extracted for the input parameters. After that, features are extracted according to the selected method and parameters and are saved.
- Clustering: this is the main module for producing results, where the various IPs are grouped using a clustering algorithm (K-Means). The outliers will correspond to the clusters that have only one IP address in them.

CYBERVTI has the REST API, which is also implemented on the server-side, to communicate between the back-end and the front-end. The endpoints that the API has been divided into *Users*, *Files*, *Features*, *Clusters* and *Metrics*.

IV. EVALUATION AND RESULTS

The first part of the evaluation is based on the metrics for system and software evaluation presented in ISO/IEC 25010:2011 Systems and software Quality Requirements and Evaluation (SQuaRE) [9]. The second part compares the results obtained by CYBERVTI with those described in the original articles of the implemented algorithms.

A. ISO/IEC 25010:2011 Evaluation

The evaluation of CYBERVTI is based on the theoretical concepts of systems and software evaluation presented in ISO/IEC 25010:2011 Systems and software Quality Requirements and Evaluation (SQuaRE) [9]. It considers the GUI and the back-end of the system. The evaluation parameters were divided into two groups: Key Evaluation Parameters and Complementary Evaluation Parameters. The Key Evaluation Parameters and their evaluation are:

- Functional Suitability: This aspect concerns the achievement of the objectives by the functions of the tool. It evaluates if the results presented are in accordance with what was obtained by the various algorithms. CYBERVTI meets its main objective to detect anomalous behaviour and classify machines as potential threats or victims. CYBERVTI allows the user to choose several different analysis methods and offers the ability to analyse network activity over several time periods.
- Compatibility: This refers to the tool's ability to function correctly regardless of the hardware or software used. This aspect also considers the interoperability and the degree to which two or more systems, products, or components can exchange information and use the information that has been exchanged. The CYBERVTI GUI is compatible with both Windows and Linux environments. A Docker container image is provided with the back-end code, packages, and dependencies so that the tool works regardless of the operating system used. Regarding the input data format, they are limited to the .csv format and the structure of the columns with fixed headers. However, this format is simple and easy for the user to input data.
- Usability: Degree of effectiveness, efficiency, and satisfaction obtained by users when using the tool. The difficulty of use, ability to configure and operate, protection against user errors, and whether the outputs are shown are important and properly organized should be considered. The CYBERVTI GUI is made in a simple way, containing only the necessary functions for efficient data analysis. The results obtained are organized and presented in tables and graphs. The GUI has a help button that allows the user to clarify doubts about the use of the tool, protecting the system against user errors.
- Security: How the data (used and stored) and the whole system are protected and the system's reliability. CYBERVTI is ready to use HTTPS requests to obtain more security during communication. The security of stored data is directly related to the server's security where the back-end is running. Regarding the system's reliability, in the graphical interface folder, there is a configuration file that must not be corrupted. Otherwise, the GUI may stop working.

The Complementary Evaluation Parameters and their evaluation are:

- Performance Efficiency: This point considers the efficiency of algorithm execution in terms of processing time and how the memory in the system is used. The algorithms integrated into the CYBERVTI used multiprocessing. However, the processing done by the server can still be improved with the use of distributed systems and load balancers.
- Maintainability: How effectively a system can be modified, the modules should follow the open-close principle, and if system maintenance is done simply. CYBERVTI requires maintenance to manage the files stored in folders on the server for efficient use of storage space, i.e., files that are not used for a long time should be deleted.
- Portability: Efficiency with which a system and its resources have to be transferred to other hardware, software, or used in another environment. The front-end is easily exportable to Windows and Linux environments through a .zip file. A docker container allows the back-end to be easily integrated into a physical server.
- Reliability: How the system behaves to perform specified functions under specified conditions. How the system reacts if a fault occurs, incorrect data is provided, or some function stops or loops. CYBERVTI does not enter infinite loops. However, if there is an error in the processing, the API stops, and it is necessary to restart it. Regarding the incorrect data in the input files, they will not be loaded if they do not comply with the formatting. If the tool gets incorrect or undefined data, it will be loaded normally, but the results obtained may be incorrect. Since it is a prototype, these reliability aspects are considered acceptable.

B. Experimental Evaluation

We use the CSE-CIC-IDS2018 dataset, which was created to test and evaluate NIDSs [4]. This dataset is structured to represent the network of a medium enterprise, with six subnets deployed on the Amazon Web Services (AWS) cloud computing platform. We consider all attack scenarios in the dataset, which are: brute force attacks (FTP and SSH), Denial of Service (DoS) attacks, web attacks, infiltration attacks, port scans, and botnet attacks.

Regarding the evaluation and verification of intrusion detection algorithms, we considered only one case (the one that is common to the papers of the three algorithms): detecting the misbehavior of nodes internal to the network, either because they are attackers or victims. This means that only the IP addresses belonging to the network under analysis were considered, i.e., only the IPs with the prefix 172.31.0.0/16 were analysed to understand which machines on the network are suffering or carrying out cyber attacks.

It should be noted that DYNIDS was not implemented with the 3 clustering algorithms, only K-Means was implemented. Thus, the results obtained based on this detection algorithm do not fully match those described in the DYNIDS paper [8].

To analyse the performance of the algorithms, we consider the hosts detected as outliers. According to the attacks and their duration presented in the dataset, we considered: True Positives (TPs), hosts that are correctly classified; True Negatives (TNs), hosts that are not flagged as outliers; False Negatives (FNs), hosts that should have been classified as outliers but were not; and False Positives (FPs), hosts that were classified as outliers but were not threats.

The metrics that we privileged for the evaluation were the F_1-Score and the Matthews Correlation Coefficient (MCC) because they provide a summary of the performance of the algorithms. We did not calculate the Accuracy because the dataset is very unbalanced, as any network intrusion monitoring dataset (malicious traffic is just a small fraction of normal traffic). The metrics we obtained are:

Precision – the fraction of outliers that are real:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall – the fraction of outliers that are correctly classified by the algorithms:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

F_1-Score – global evaluation metric of the algorithm performance:

$$F_1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

MCC – used for the quality assessment of binary classifications:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4}$$

Figures 11, 12, 13 and 14 show the Precision, Recall, F_1-Score and MCC of the detection algorithms implemented in CYBERVTI. These were obtained by analysing the CSE-CIC-IDS2018 dataset in several time windows. We noticed that the implemented algorithms did not perform well in detecting botnet attacks, failing to detect any victim in the various time windows. Therefore, we start by presenting results for detection without day 10, when these attacks happened. The values presented in the graphs were calculated relative to the total values of TPs, TNs, FNs, and FPs.

The results with all algorithms are presented in Figures 13 and 14. Something that comes immediately to mind in the figures is that some results appear to be bad. For instance, precisions of 0.25 to 0.27 are quite bad (Figure 11). However, this is not an issue as the point is that we run the algorithms in several time windows, following OUTGENE's time stretching approach, in order to detect the malicious traffic in *any* of them, not in *all* of them.

From the figures, we conclude that DYNIDS is the algorithm that obtains the best results. As expected, the results of FLOWHACKER and OUTGENE are similar since they use a similar approach, differing only in one port from the list of ports analysed. The results obtained in the metrics to evaluate the performance of the algorithms vary a little due to the way the TPs, TNs, FNs and FPs were accounted for. The correlation between the 3 clustering algorithms was not used in our implementation of the DYNIDS so their performance decreased, as expected.

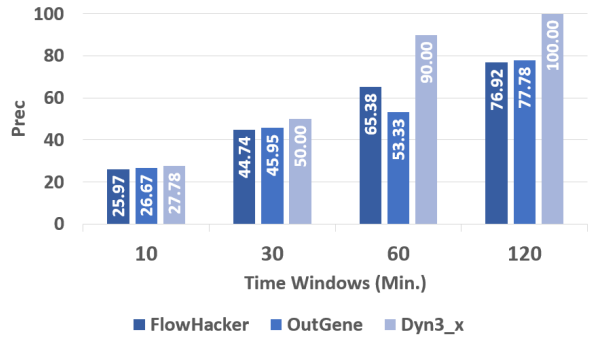

Fig. 11. Precision of the 3 algorithms for the dataset.

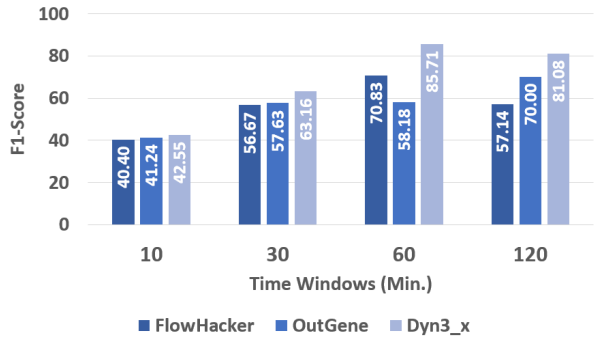

Fig. 13. F_1-Score of the 3 algorithms for the dataset.

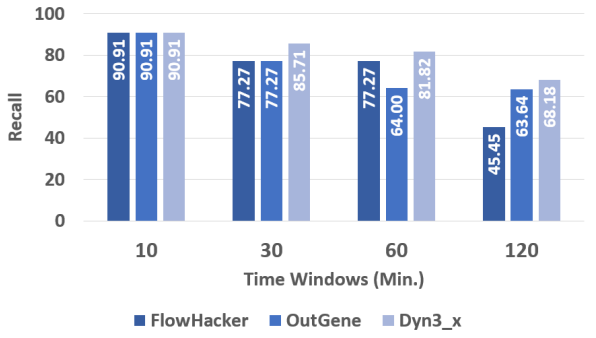

Fig. 12. Recall of the 3 algorithms for the dataset.

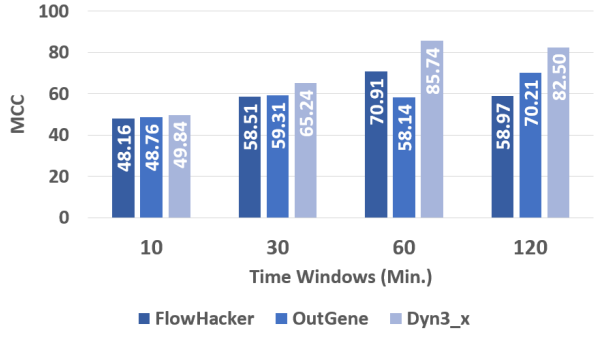

Fig. 14. MCC of the 3 algorithms for the dataset.

V. Related Work

This section presents visualization tools that are somehow related to CyberVTI.

Cyber Mission Impact Assessment (CMIA) [19] is a tool to evaluate the system/network mission impact that possible cyber incidents would have. The effects of cyber incidents on the mission are classified as degradation, disruption, modification, manufacturing, unauthorized use, and an interception. Regarding the visualization part, this tool graphically presents the results of the incident effects on the system mission and a diagram of the network infrastructure.

Graph-Based Analytics and Visualization for Cybersecurity (CyGraph) [21] is a system for analyzing and reasoning about network attack relationships. It correlates data from various sources into a common, normalized model. It builds a persistent graphical data store representing network attack relationships and associated network data, providing an interactive visualization of complex dependency relationships. CyGraph is based on a client-server architecture implemented through a REST API.

Cyber Attack Modelling and Impact Assessment tool (CMIAC) [12] is a tool for cyber attack modelling and impact assessment. That is based on representing potential malefactors behaviour, generating attack graphs, calculating security metrics, and providing risk analysis procedures. It applies a set of algorithms with different timelines and precision to get near-real-time event analysis and security and impact assessment predictions. The attack graph is modified according to the changes in the analysed network.

VisSecAnalyzer [13] is a visual analysis tool for network security assessment that aims to provide visual support to the cybersecurity officer. Its goal is to reveal the most vulnerable points of the information system, forming attack patterns. After assessing the network security and analyzing the severity of the detected vulnerabilities, this tool can suggest possible countermeasures. The GUI in VizSecAnalyzer consists of interactive graphs and colourful security reports, allowing an exploration of a large-scale network.

Rapid EvaLuation Of Anomaly Detection (RELOAD) [30] is a tool that implements network anomaly detection algorithms. Its goal is to evaluate the performance of the algorithms in anomaly detection. It allows the user to integrate new algorithms and make their evaluation. However, it provides previously integrated algorithms. This tool takes data from an existing dataset and identifies the essential features automatically. After this stage, the anomaly detection algorithms are run, and the classification results are obtained. In the final phase, metrics are generated relative to the performance of the algorithms. RELOAD is executed locally and has a simple GUI.

These tools were the basis of inspiration and knowledge acquisition for the development of our solution. They do not share our goal of being focused on unsupervised ML detection algorithms or their configuration. They also do not follow a client-server architecture similar to ours.

VI. Conclusions

We present a visualisation tool for intrusion detection, based on a client-server architecture and containing an interactive and simple GUI. In the back-end are the algorithms based on unsupervised ML, responsible for obtaining the data for the GUI and REST API, that allows the communication data between the back-end and the front-end through HTTP/HTTPS requests. The GUI of the tool has buttons, lists, graphs and menus, allowing its use by any user without previous knowledge. The user can easily change the hyperparameters of the algorithms, obtaining personalised results for different types of analysis.

Acknowledgements

This research was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID), by the Portuguese Army (CINAMIL), and by the European Commission under grant 830892 (SPARTA).

References

[1] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys Tutorials*, 16(1):303–336, 2014.

[2] P. Bowen, J. Hash, and M. Wilson. Information security handbook: a guide for managers. In *NIST special publication 800-100, National Institute of Standards and Technology*, 2007.

[3] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2015.

[4] Canadian Institute for Cybersecurity. CSE-CIC-IDS2018 on AWS. https://www.unb.ca/cic/datasets/ids-2018.html.

[5] P. Casas, J. Mazel, and P. Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.

[6] R. Copeland. *Essential SQL Alchemy*. O'Reilly Media, Inc., 2008.

[7] L. Dias, H. Reia, R. Neves, and M. Correia. Outgene: Detecting undefined network attacks with time stretching and genetic zooms. In *International Conference on Network and System Security*, pages 199–220. Springer, 2019.

[8] L. Dias, S. Valente, and M. Correia. Go With the Flow: Clustering Dynamically-Defined NetFlow Features for Network Intrusion Detection with DynIDS. In *19th IEEE International Symposium on Network Computing and Applications (NCA)*, 2020.

[9] J. Estdale and E. Georgiadou. Applying the ISO/IEC 25010 quality models to software product. In *European Conference on Software Process Improvement*, pages 492–503. Springer, 2018.

[10] T. Fernandes, L. Dias, and M. Correia. C2bid: Cluster change-based intrusion detection. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 310–319, 2020.

[11] S. Hakak, W. Z. Khan, M. Imran, K.-K. R. Choo, and M. Shoaib. Have you been a victim of COVID-19-related cyber incidents? Survey, taxonomy, and mitigation strategies. *IEEE Access*, 8:124134–124144, 2020.

[12] I. Kotenko and A. Chechulin. A cyber attack modeling and impact assessment framework. In *2013 5th International Conference on Cyber Conflict (CYCON 2013)*, pages 1–24, 2013.

[13] I. Kotenko and E. Novikova. Vissecanalyzer: A visual analytics tool for network security assessment. In *International Conference on Availability, Reliability, and Security*, pages 345–360. Springer, 2013.

[14] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 25–36. SIAM, 2003.

[15] F. Lundh. An introduction to tkinter. http://www.tcltk.co.kr/files/TclTk_Introduction_To_Tkiner.pdfs, 1999.

[16] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido. Analysis of high volumes of network traffic for advanced persistent threat detection. *Computer Networks*, 109:127–141, 2016.

[17] Matt Makai. Full Stack Python. https://www.fullstackpython.com/flask.html, Oct-2020. Accessed: 18-Nov-2020.

[18] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma. Su-ids: A semi-supervised and unsupervised framework for network intrusion detection. In *International Conference on Cloud Computing and Security*, pages 322–334. Springer, 2018.

[19] S. Musman and A. Temin. A cyber mission impact assessment tool. In *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–7, 2015.

[20] G. Nascimento and M. Correia. Anomaly-based intrusion detection in software as a service. In *1st International Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments*, pages 19–24, 2011.

[21] S. Noel, E. Harley, K. H. Tam, M. Limiero, and M. Share. Cygraph: graph-based analytics and visualization for cybersecurity. In *Handbook of Statistics*, volume 35, pages 117–167. Elsevier, 2016.

[22] M. Owens and G. Allen. *SQLite*. Springer, 2010.

[23] Python. WebFrameworks. https://wiki.python.org/moin/WebFrameworks, Nov-2020. Accessed: 20-Nov-2020.

[24] L. Richardson and S. Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[25] L. Sacramento, I. Medeiros, J. Bota, and M. Correia. Flowhacker: Detecting unknown network attacks in big traffic data using network flows. In *17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 567–572, 2018.

[26] D. Staheli, T. Yu, R. J. Crouser, S. Damodaran, K. Nam, D. O'Gwynn, S. McKenna, and L. Harrison. Visualization evaluation for cyber security: Trends and future directions. In *Proceedings of the Eleventh Workshop on Visualization for Cyber Security*, pages 49–56, 2014.

[27] D. Vohra. Apache Parquet. In *Practical Hadoop Ecosystem*. Springer, 2016.

[28] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning*, volume 1, pages 577–584, 2001.

[29] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[30] T. Zoppi, A. Ceccarelli, and A. Bondavalli. Evaluation of anomaly detection algorithms made easy with reload. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 446–455, 2019.