Chapter 5
VizAttack: An Extensible Open-Source Visualization Framework for Cyberattacks

Savvas Karasavvas, Ioanna Dionysiou, and Harald Gjermundrød

5.1 Introduction

The global proliferation of IoT devices and technology has introduced new security challenges that span the devices themselves, their communication channels and the systems that are connected to. Preventing security breaches in such a heterogeneous and diverse environment is nontrivial, despite the abundance of security products and technologies in the market, as the attack surface is simply too broad. The frequency of cyberattacks is increasing dramatically and organizations from both public and private sectors are struggling to identify and respond to those security breaches. Over the last few years, several instances of security breaches were brought to light with at least one thing in common: the response time was too long. According to the 2019 IBM *"Cost of a Data Breach"* report [1], the mean time to identify (MTTI) a breach in 2019 was 206 days and the mean time to contain (MTTC) was 73 days, with a notable 4.9% increase over the 2018 breach life cycle.

Taking into account the above-mentioned facts, one should expect that the security parameter of a system will be penetrated by an unauthorized party at some point, and the goal must be to identify the incident as quickly as possible and respond effectively. The identification of the security attacks relies on technological and human factors: the former one being the security tools that are integrated in the organization's network infrastructure and the latter one being the in-house security and system administrators who have the ultimate responsibility of all decision-making. *Defense-in-depth*, a multilayered strategy that supports defensive mechanisms on various layers, is a popular approach to protect the network. A sustained *defense-in-depth* strategy uses a multivendor approach, making the

S. Karasavvas · I. Dionysiou (✉) · H. Gjermundrød
Department of Computer Science, School of Sciences and Engineering, University of Nicosia, Nicosia, Cyprus
e-mail: karasavvas.k@live.unic.ac.cy; dionysiou.i@unic.ac.cy; gjermundrod.h@unic.ac.cy

K. Daimi, C. Peoples (eds.), *Advances in Cybersecurity Management*,
https://doi.org/10.1007/978-3-030-71381-2_5

management and maintenance of such layered defense stance nontrivial. In addition, each security technology usually operates independently and the lack of a standard format for log files affects unfavorably the global security situational awareness of the network that requires the correlation of data stored in all log files. Meaningful cyber security situational awareness leverages security management, as it provides the global security state of the administrative domain, thus allowing for informed decision-making on security matters. The quality of the security management of an administrative domain is likely to rise, as the number of security data sources increases.

This chapter presents *VizAttack*, an extensible and open-source visualization framework for data generated by security technologies. Visualization of cyberattacks is gaining popularity as an intuitive technique to present attack data without overwhelming the user. A cyberattack map is an example of security attack visualization, graphically depicting attack metadata such as geographical data and attack data type for malicious attempts to penetrate the external lines of defense of a network. *VizAttack* makes provisions to visualize attack metadata residing in unstructured heterogeneous log files, and in addition, it reconstructs the steps followed during an attack execution (command sets prior the attack penetration and after bypassing the security parameter). Attack profiling is critical to attack monitoring, a process not supported by current attack visualization tools. *VizAttack* supports predefined and customized queries on the imported attack data sets as well as on-demand queries that are constructed *on the fly* during the investigation of attack profiles. The latter is a novel feature that could be utilized by security management modules that detect attacks.

The chapter contributions are twofold as shown below:

- Design an extensible framework with visualization and analysis capabilities for attack data sets generated by heterogeneous sources.
- Implement a prototype version of *VizAttack*, demonstrating the functionality of the framework. The prototype version of *VizAttack* uses log files from honeypot technologies, but the extensible nature of *VizAttack* allows the seamless integration of data sets from other security technologies.

The rest of the chapter is organized as follows: Sect. 5.2 discusses related work in the area of cyberattack visualization as part of the overall cyber situational awareness process, followed by Sect. 5.3 that introduces *VizAttack* and presents its design principles and objectives. Section 5.4 discusses the implementation of *VizAttack* and Sect. 5.5 concludes with future directions.

5.2 Cyberattack Visualization Approaches

Information visualization is a user-intuitive approach that transforms raw data into a visual form [2]. Incorporating this technique in the cyber situational awareness process is beneficial as, when used appropriately, could leverage the knowledge

obtained from various security data sources by providing visual analytics. Tools and visualizations for cyber situational awareness are given in [3, 4], with issues and challenges to enable cyber situational awareness discussed in [5]. Cyberattack maps and attack graphs are two popular techniques of security data visualization that provide attack data visualization and vulnerability exploitation pathways, respectively. In this section, a discussion on these two techniques is presented, followed by a discussion on visualization of attack data collected by honeypots as well as the current challenges in the area of cyberattack visualization.

5.2.1 Cyberattack Maps and Graphs

There are numerous security solution vendors that maintain publicly available cyberattack maps, showing global security incidents. Usually, attack metadata is illustrated, including origin IP, destination IP, type of attack, and time/date. The most popular cyberattack maps are the Arbor Networks DDoS Attack Map [6], Kaspersky Cyber Malware and DDoS Real-Time Map [7], Fortinet Threat Map [8], Sophos Threat Tracking Map [9], FireEye Cyber Threat Map [10], and Threat Cloud Live Cyber Attack Threat map [11]. Due to space limitations, three cyberattack maps are selected to elaborate more on their functionality, representing frequently updated maps, live maps, and static maps, respectively.

The Arbor Networks DDoS Attack Map [6] is a threat map that is hourly updated, and historical data are available to be replayed. It focuses exclusively on Distributed Denial of Service (DDoS) attacks, and the attack metadata provided are the attack source and destination, the size of the attack in Gbps, source and destination ports, and attack duration. The data sources used to render the threat map are more than 300 ISPs, giving 130 Tbps of live traffic. The Fortinet Threat Map [8] is a live map, giving information not only on DDoS but also on other types of attacks such as remote execution attacks and memory-related attacks. It shows live attack activity, but its feature set is not as rich as Arbor's. The Sophos Threat Tracking Map [9] is an example of a static map that depicts the daily malicious web requests, blocked malware, and web threats as collected by the Sophos Labs. Overall, the usefulness of a cyberattack map is under scrutiny, as most cyberattack maps do not show live data but archived data of past attacks, making it an ineffective way to mitigate attacks in real time. Additionally, the emphasis is usually on DDoS attacks, leaving out other forms of attacks. On the other hand, these threat maps could be used to study attack patterns and form attack profiles.

Attack graphs, unlike cyberattack maps, are not depicting data from real attack attempts, but they rather graphically show potential steps to carry out an attack by exploiting vulnerabilities. In a sense, it is a threat-modeling approach, illustrating all possible avenues of successfully executing a specific attack. The various pathways to execute the attack are usually high-level description of steps or commands [12]. There are no automated tools to develop an attack graph, and building a complete attack graph is very labor intensive for a complex system.

5.2.2 Honeypot Data Visualization

One source of cyberattack data and metadata is the honeypot system, a security resource whose value lies in being probed, attacked, or compromised. That's the ultimate goal of the honeypot: having it being probed and/or exploited due to the deliberate planting of vulnerabilities, analyze the compromised system and gain knowledge about the nature of the attack and the attacker patterns. A honeypot should be designed in such a way so that it looks real to the attacker but it is isolated from any production system.

There is a plethora of honeypot implementations, with a few providing visualization of the collected data. These tools include the NFlowVis [13], VIAssist [14], Dionaea [15] and its successor Nepenthes [16]. In general, honeypots could use the *kippo* family of graph modules to generate graphical representations of the data (*kippo-graph*) and annotated them with geolocation data (*kippo-geo*). A total of 24 graphs could be generated for username and password statistics, password length statistics, credentials combinations, successful logins rates, intensity map, and top commands used.

5.2.3 Attack Visualization Challenges

The visual depiction of cyberattack metadata, as described in the aforementioned approaches, facilitates the presentation of attack metadata in a user-intuitive simple manner. Indeed, an average user, without an extensive technical background, could interpret the data and derive basic conclusions. However, a security analyst needs to be presented with advanced features, allowing the correlation of the collected data to first yield interesting knowledge about the attack methodology itself and second utilize the newly acquired knowledge to improve the security processes of an administrative domain. There is need for a platform that integrates the features of an attack map, the principles of an attack graph, low-level system commands executed during attacks, and flexible queries into a single framework that aims not only in the visualization of attack metadata but also in the construction of attack pathways. In addition, multivendor security technologies must be supported to gain a more accurate insight into the current state of security of the network.

5.3 VizAttack Design Principles

This section discusses the design principles behind *VizAttack*, a novel approach to cyberattack visualization that aims to alleviate the attack visualization challenges mentioned earlier. Not only attack metadata analytics are supported, similar to the

existing approaches, but also attack command pathways and user-defined custom queries, leveraging the on-demand visualization.

5.3.1 Design Objectives

In order to realize the aim set for *VizAttack*, one needs to collect and correlate data from log files generated by various security technologies that protect not only the perimeter of the network infrastructure but also monitor activities inside the infrastructure itself. Situational awareness is an important element of security management as it mitigates security risks, but its effectiveness is rather limited if one were only allowed to use a single source of security data (e.g., firewall). The richer the pool of sources of security data, the more informed decisions could be taken. However, the unstructured nonstandard nature of log files is a serious impediment to the correlation of information from heterogeneous sources. *VizAttack* provides a modular approach that integrates unstructured log files in a single framework and performs analytics on the collected information. Below are the objectives that the *VizAttack* design must adhere to:

- **Extensible**—be able to import and analyze any type of attack log file from any security source (e.g. firewall, honeypot, intrusion detection system) running on a port, preferably using login credentials. As there is no standard format for log files, a custom parser must be implemented for each log file type. Provision must be made to allow the seamless integration of new parser modules.
- **Publicly accessible**—be able to use and modify as needed, thus released as open source software.
- **Configurable**—support predefined data analytics and customized queries to provide on-demand visualization of user-selected data.

5.3.2 High-Level Architectural Design

The baseline components of *VizAttack* are illustrated in Fig. 5.1. The back-end of the system takes raw data collected by security technologies and imports them into the *VizAttack* framework, where dedicated parsers perform the scanning and parsing of the imported data based on the security technology that collected the specific data set. This modular approach allows the seamless integration of new data sources by either building a new customized parser or use an existing one that is deemed to be suitable. Additionally, any future modifications in the structure of a log file will only require modifying the dedicated parser module for that file. The parsed data are stored in the *VizAttack* database, accessible by the front end of the system. Temporal analysis and queries (predefined and customized) are performed on the stored data,

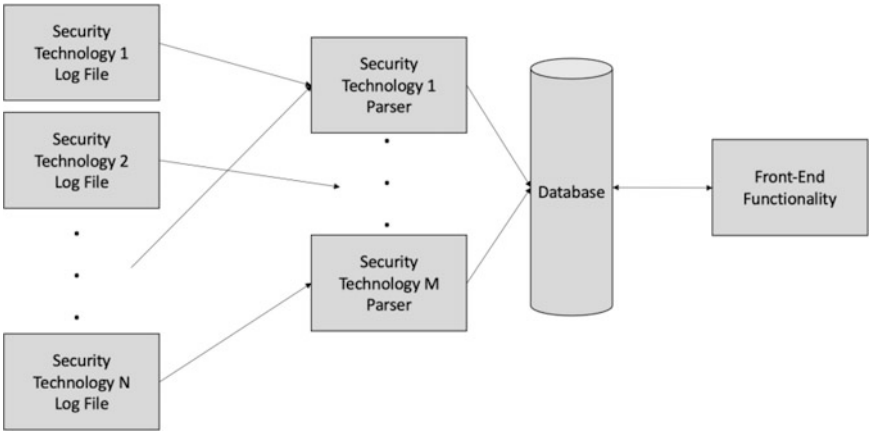

Fig. 5.1 VizAttack baseline components

with visualization capabilities not only for the stored data but also for the actual methodology followed in terms of low-level commands.

Figure 5.2 shows the interaction protocol of the system, which depicts the flow of interaction among the *VizAttack* entities along with the message exchanges. Emphasis is given on the sequence of the events. A user interacts with the UI window to load a new log file or open an existing one. In the former case, the parsing process will be executed, with the results stored in the database. This temporal analysis takes place in a transparent manner, without requiring any feedback from the user. In the latter scenario, the file is already parsed, and the results are fetched from the database. Predefined queries as well as customized ones could be run on the loaded file. In addition, the construction of attack pathways is possible, requiring a deeper analysis of the parsed entries in order to identity and link together log entries corresponding to a single attack.

Details on the front-end functionality is given next, along with description on the back-end algorithms, where applicable.

5.3.2.1 User Interface: Temporal Analysis

The temporal analysis is performed on every newly imported log file, and its primary goal is to parse the raw data and store them in the *VizAttack* database. The appropriate parser is selected based on the security technology that generated the log file. Figure 5.3 illustrates the results from the temporal analysis of *kippo.log*, a log file generated by the *kippo* honeypot. A color-coded scheme is supported to display session data in an intuitive manner. The parsing of the raw data allows a user to execute advanced queries on the data, including isolating the data related to a specific session, an important part for profiling the attack methodology followed during an attack.

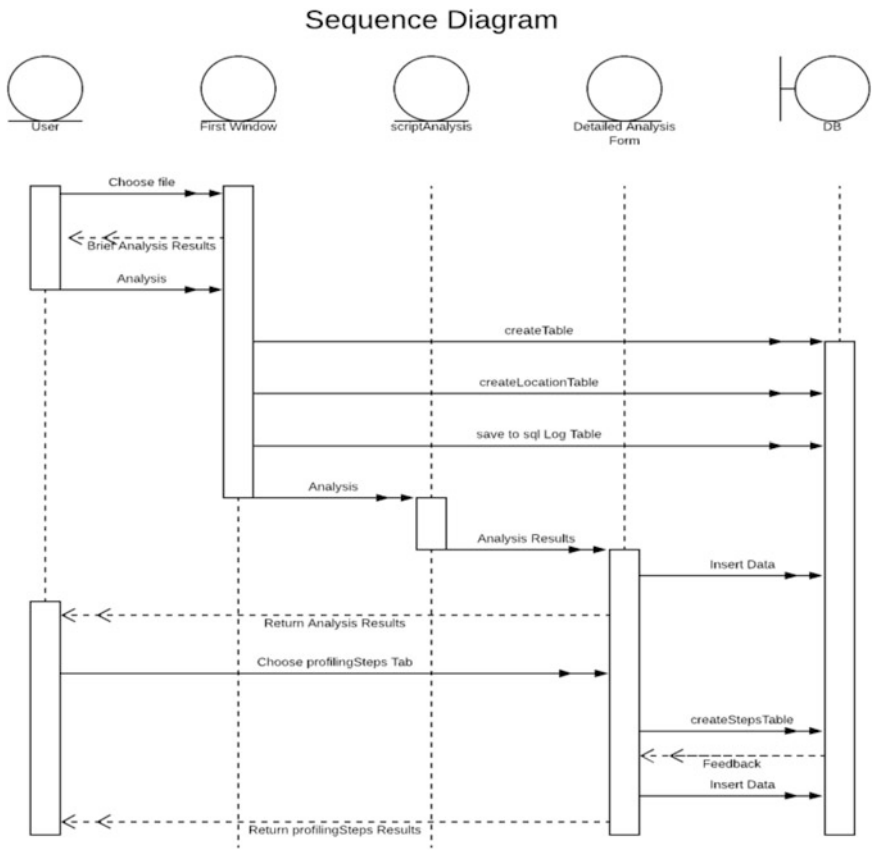

Fig. 5.2 VizAttack sequence diagram

5.3.2.2 User Interface: Predefined Queries

The front-end functionality of *VizAttack* is illustrated in Fig. 5.4. Each *VizAttack* feature could be utilized in the overall security management of an administrative domain. The user interface follows a simplistic design approach, having the various *VizAttack* features listed on a single menu row. The predefined queries are listed first (Location/IP Ports, Login Attempts, Map Location, Top Attached Dates, Special Dates), followed by the custom query and profiling steps features.

Starting with *the Location/IP Ports* option, information regarding the source of the attack as well as the target service is displayed, along with statistical information regarding the most and least appeared source IP addresses and the most and least used ports. The actual location related to an IP address is obtained by making a web request to the server *ipinfo.io*. The server reply includes, among other data, the hostname, city, region, country, organization information, latitude, and longitude related to the IP address. The latter two data fields are used by the *Map Location*

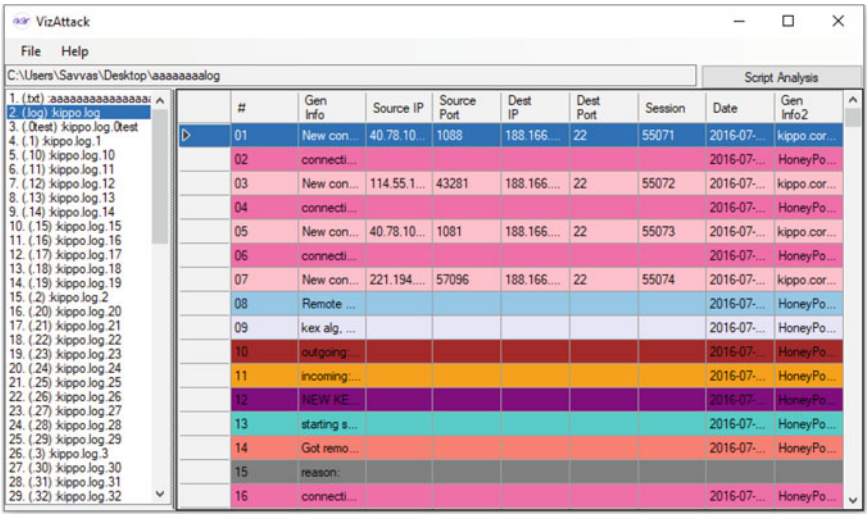

Fig. 5.3 Temporal analysis of *kippo.log*

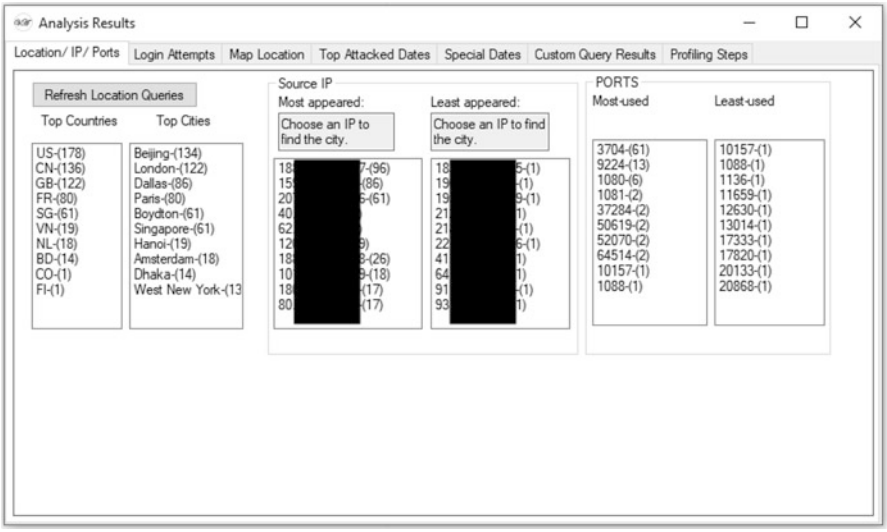

Fig. 5.4 VizAttack front-end features

option introduced later on. It is important to note that the attack attribution solely based on the source IP address is not an accurate method, as there are numerous techniques to spoof IP addresses or use compromised machines to launch an attack. Nevertheless, the information on the source IP address could still offer some insight on tracing the attack root. In addition, system administrators could monitor the

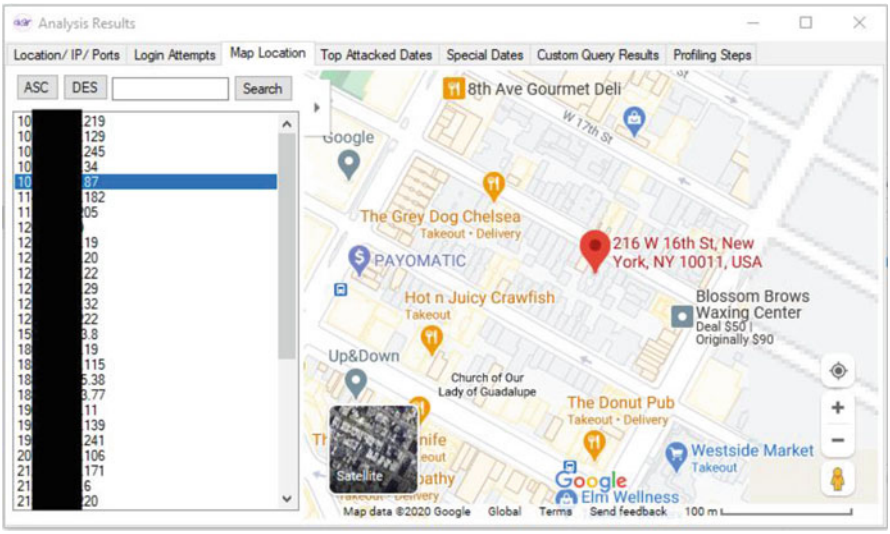

Fig. 5.5 VizAttack map location

targeted services, especially the least used ones. Those ports maybe were left opened unintentionally, posing a risk to the overall security perimeter of the system.

The *Map Location* feature uses information retrieved by *ipinfo.io* to display the location of the attack source in Google maps. More specifically, the latitude and longitude are passed as parameters to a Google Map iframe to render the location on the map. A user could either select an IP from an existing list populated with IP addresses retrieved from the log files (in ascending or descending order) or search for a particular IP address. Figure 5.5 illustrates the exact location for a selected IP address. As mentioned earlier, this information does not provide an undisputable evidence that the attack originated at that location. But it could still provide useful information, for example in the case that the IP address is located inside the organization itself.

The next menu option, *Login Attempts*, displays information regarding the credentials the attackers used for an unauthorized login to the system. The most used passwords and usernames are listed along with their frequency. Furthermore, a list is provided with the strings used as both username and password in single login attempt. The findings from this investigation could play a significant role in the password policy management of the system. Figure 5.6 lists the most frequently used (by the attacker) usernames and passwords, along with a list of strings used both as username and password.

The last two predefined query options correlate the source of the attack with the attack date. In the case of the *Top Attacked Dates*, the user selects a country, and a dynamic list is generated that displays the number of attacks originated from that country, in an ascending order, based on the attack date. Figure 5.7 lists the

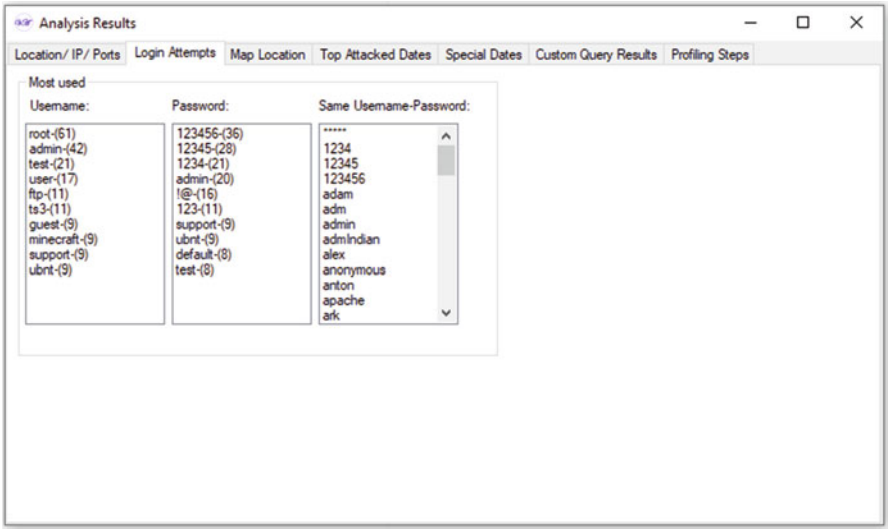

Fig. 5.6 VizAttack login attempts

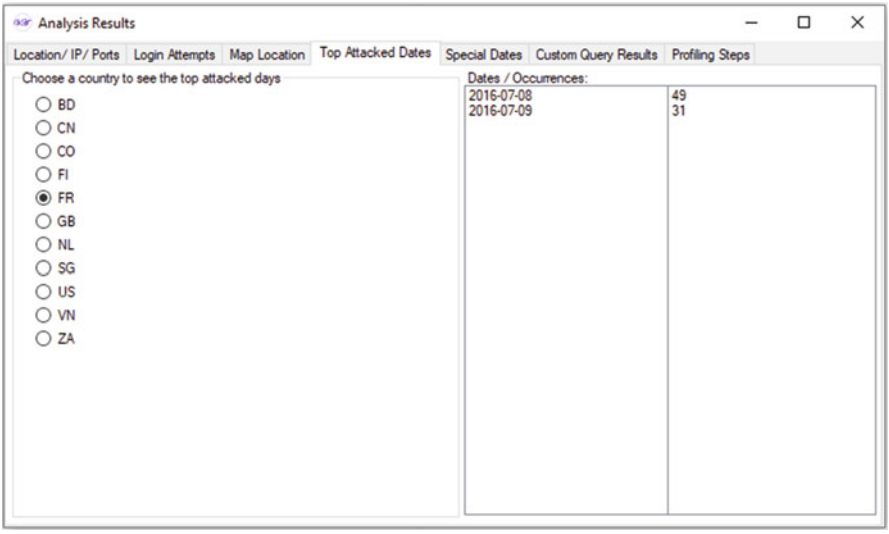

Fig. 5.7 Top attacked dates

attacks originated by a source IP located in France within a 2-day period. The *Special Dates* option is also country-specific, but the attack date range is restricted to country holidays (national, federal, etc.). *VizAttack* has made provisions to integrate these dates into its framework, drawing interesting conclusions on whether or not

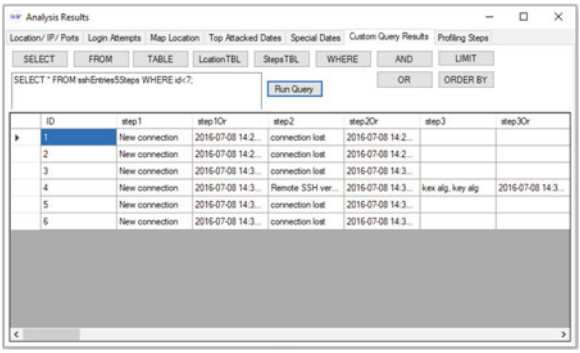

Fig. 5.8 VizAttack query editor commands

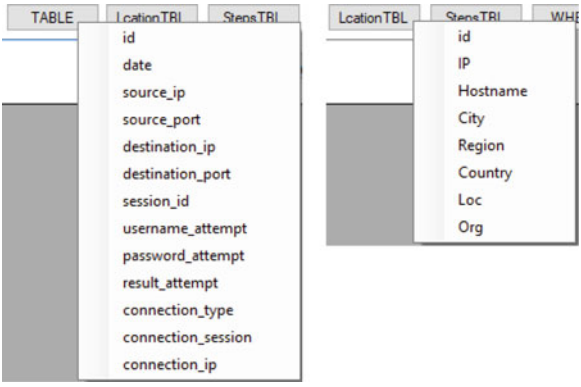

Fig. 5.9 VizAttack query database fields

there is a correlation between launching attacks and holidays, alerting the system administrators to be more cautious on those days.

5.3.2.3 User Interface: Customized Queries

Advanced customized queries are supported in *VizAttack* via a simple typical SQL query editor integrated in the system. The query construction is done in an error-free manner, where both SQL commands and database fields (Figs. 5.8 and 5.9, respectively) are at the user's disposal, thus eliminating query syntax and/or typo errors. A query could still be constructed from scratch. However, suitable safeguards are in place to prevent unauthorized modifications to the database. To be more specific, any user-specified query including commands from the command set {*INSERT, DELETE, DROP, ALTER, MODIFY*} is classified as invalid by the query compiler; thus, it is not executed.

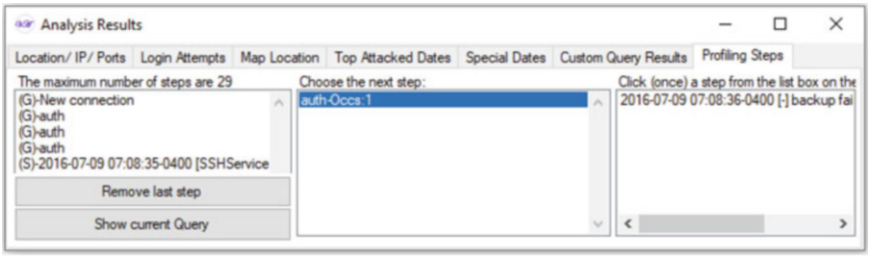

Fig. 5.10 Profiling attack steps

5.3.2.4 User Interface: Profiling Attacks

One of the novel features of *VizAttack* is its ability to reconstruct the steps taken during an attack using the raw data from log files. Attack profiling allows one to infer the attack methodology used in the particular attack and utilize the constructed attack profiles in the attack prevention/detection processes of the system. Once a log file is parsed, attack profiling could be initiated via *Profiling Steps* feature to establish the actions of the penetrator in a single session, including the attempts to penetrate the system and the actions taken after a successful breach. These are called attack *steps*. Not only the attack steps are linked together for a single attack but also several attacks could be correlated to find commonalities in their execution paths. Furthermore, the popularity of each step/action is listed, along with the maximum number of steps in a single attack session. Currently, the actions supported by VizAttack are {*New connection, Connection lost, Auth, Cmd, Command found, Command not found, Executing command/Running exec command, Login attempt, Incoming, Outgoing, Error, Opening TTY log, Remote SSH version, NEW KEYS, Kex alg, key alg*}.

Figure 5.10 illustrates a snapshot of the attack steps profiling process. For this particular archived data, there are five different attack sessions, and their first step is indicated on the leftmost column. The longest attack chain consists of 29 steps. The choices for the second step are listed in the middle column, and upon selection, the next step list is created. At all times, more information on the specific step is given by locating and displaying the corresponding entry in the log file (rightmost column).

VizAttack Profiling Steps feature empowers the security management processes of the system by:

- Determining the attack steps for a successful penetration and visualizing them in an intuitive manner
- Leveraging the integrated query module with "on-the-fly" queries that allow the construction of a query taking into consideration the current attack step
- Storing the abovementioned queries, which could be viewed as supervised training from the archived log data, and adding them into a continuous query engine with the intention to run them on newly imported log data

```
START PROGRAM
    SET maximum_number_of_steps_counter TO 0
    READ log file by lines
    FOREACH line read
        IF current line is the start of a new session THEN
            ADD list of steps TO Results list
            CLEAR list of steps
            SET steps_counter TO 0
            ADD current line TO list of steps
            INCREMENT steps_counter BY 1
        ENDIF
        IF current line is the end of a session THEN
            ADD current line TO list of steps
        ENDIF
        IF current line belongs to any other category THEN
            ADD current line TO list of steps
            INCREMENT steps_counter BY 1
        ENDIF
        IF steps_counter > maximum_number_of_steps_counter THEN
            SET maximum_number_of_steps_counter = steps_counter
        ENDIF
    END FOREACH
    IF list of steps is not empty THEN
        ADD list of steps TO Results list
    ENDIF
    CALL FUNCTION CreateStepsTable
    CALL FUNCTION StoreStepsInTable
END PROGRAM
```

Fig. 5.11 Profiling attack steps pseudo-algorithm

- Providing insight into attacks that may be currently running undetected (e.g., the attack's last step is not an action that indicates the termination of the session)

Details on the back-end algorithms that implement this feature can be found in [17]. The pseudocode algorithm that parses the entries of a log file and determines the number of distinct attack sessions and the steps associated with each attack is given in Fig. 5.11. The selected log file is parsed, one line at a time, and the action taken by the attack profile construction algorithm depends on the type of line that is currently parsed (*current line*) as follows:

1. If *current line* is a *New Connection*, then the current attack session (if it exists) is closed, and it gets added to the completed attack sessions list (*Results List*). Each entry in the *Results List* is a distinct attack located in the log file, along with a linked list of all commands (attack steps) executed during the attack. The *current line* will then serve as the first step of a new attack session.
2. If *current line* is not a *New Connection*, then it gets added as an attack step in the current attack session. In this case, the steps counter gets incremented by 1. The current step counter is compared against the maximum step counter, which always keeps track of the maximum number of attack steps from all attack sessions.

After the completion of the algorithm, the *Results List* consists of all attack sessions that were identified in the parsed log file, and all this information is stored in the appropriate *VizAttack* database.

5.4 VizAttack Implementation Details

The high-level architecture of the *VizAttack* framework is illustrated in Fig. 5.1. A prototype of the *VizAttack* framework was implemented to comply to its design principles and the required front-end functionality. This section gives implementation details on the prototype as well as the results of the experimental testing. A discussion on how *VizAttack* could be utilized to investigate specific attacks is also presented.

5.4.1 VizAttack Prototype

The current prototype was implemented using *Microsoft Visual Studio IDE* and the *C#* language. As the prototype mainly serves as a proof of concept, the *MS Access* database was used as the back-end database due to its simplicity in integrating the various components together within the development environment. It is anticipated that the next version of *VizAttack* will use an open-source database such as *MySQL* and also use the open-source project *Mono*, allowing *VizAttack* to run on a multi-platform environment (Linux, macOS, Windows).

Furthermore, it was decided to use log files that contain information, which allows the reconstruction of the attack steps once a network breach was successful. Honeypot log files fulfill this criterion, as they monitor and log the attacker behavior inside the compromised system; thus, the *VizAttack* prototype supports a parser for the honeypot *kippo* log files. A rich set of kippo log files was imported to *VizAttack* via *HoneyCY* [18]. *HoneyCY* is a comprehensive open-source system that integrates mature honeypot implementations into a single inexpensive framework that is easy to deploy, configure, and has provision for visualization and other management support via a web application and an Android app. Its distributed architecture supports the seamless deployment of a network of sensors (which could also be a set of *Raspberry Pi* boards with customized binary deployments for *HoneyCY*) to be probed and attacked.

5.4.2 Experimental Findings

The experiment setting consisted of a single machine with the following specifications: Intel i5(U) CPU fourth generation and 6GB of RAM on a 64-bit Windows 10. A total of 11 log files were randomly selected and imported from *HoneyCY*. The experiment objectives were twofold:

- Demonstrate the visualization capabilities of VizAttack using real log data.
- Assess the *correctness* of the two main VizAttack processes, namely the log file parsing and the attack steps construction for an attack. That was the primary

Table 5.1 Performance analysis

File ID	No. of entries	No. of sessions	Script analysis (min)	Profiling steps analysis (min)	File size (KB)
log	6057	4504	12.18	8.53	575
log1	10,175	6104	18.41	15.38	977
log2	10,144	5898	9.04	16.05	977
log3	10,200	5419	17.27	16.05	977
log4	10,606	1928	13.78	15.04	978
log5	10,504	2911	14.54	14.38	977
log6	10,262	3792	13.73	13.33	977
log7	10,265	3751	5.8	15.19	977
log8	10,180	6174	18.78	15.27	977
log9	10,140	6302	21.37	13.85	977
log10	10,055	7569	18.45	10.72	977

metric. As a secondary metric, the performance time of the two processes was measured (i.e. how long the parsing and the construction of the attack steps take) to get a preliminary estimation of their time execution.

Details on the attack data visualization and their statistical analysis could be found in [17]. Note that Figs. 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, and 5.10 are snapshots taken during the experiments and provide insight on the overall visualization design.

The focus of the experiments was primarily on the preliminary evaluation of the *VizAttack* prototype. Table 5.1 summarizes the performance evaluation for the log file parsing process and the attack steps reconstruction process, where:

- File ID: A unique identifier for each file
- No. of entries: Number of distinct entries in the file
- No. of sessions: Number of distinct attack sessions located in the file
- Script analysis: Execution time of the parsing process for the specific file (tokenization of the log file strings into entries shown in the #of entries column)
- Profiling steps Analysis: Execution time of the attack session construction process for the specific file (populating #of sessions column)
- File size: file size in KB (pure text file)

In order to demonstrate the correctness of the *VizAttack* algorithms, the two processes were performed manually, and the results (#of entries, #of sessions) from the manual analysis were compared against the *VizAttack* results. It was concluded that the results were identical. Furthermore, a similar manual approach was used to test the correctness of the attack step reconstruction visualization feature, with a positive outcome as well.

Figure 5.12a, b presents the Table 5.1 findings in a bar chart format. In both charts, *x*-axis represents the processed file, whereas *y*-axis indicates the number of entries/sessions as well as the execution time (in minutes) to perform the particular task for the specific file. Figure 5.12a depicts the analysis for the parsing

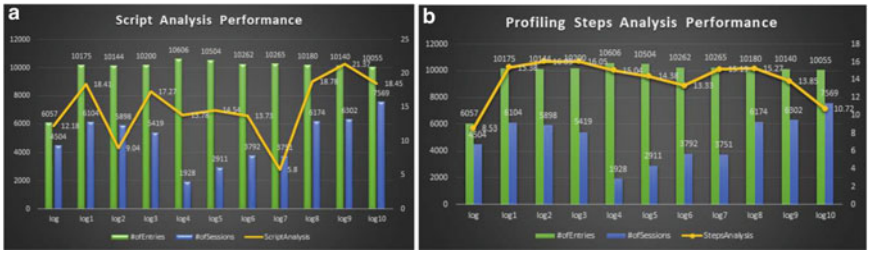

Fig. 5.12 (**a**) Parsing performance analysis. (**b**) Profiling steps performance analysis

process. The testing hypothesis was that the execution time of the parsing should be approximately linear to the number of the tokenized entries. This was based on the way the parsing algorithm executes: It processes line by line the log file, tokenizes the line into entries, further tokenizes the entry into SQL fields, and executes SQL insert queries to update the *VizAttack* database with the newly parsed data. However, as Fig. 5.12a shows, the number of entries is not linear with the elapsed time of the parsing process. Consider the execution times for files *log7* and *log8,* with approximately the same number of entries (the percentage difference of the number of entries between the two files is only 0.83%). Processing *log8* takes approximately 3 times longer to complete when compared to *log 7* elapsed time.

Figure 5.12b illustrates the step profiling process performance. The log file entries are read and linked together into unique sessions. A new session starts when the entry category on the particular log file line is either *New Connection* or *Connection Lost*. The *depth* of each session, thus the attack steps, depends on the number of log file entries that exist between the current session and the next one that gets identified. Similar to the parsing process, the step profiling task is not directly linear to the number of sessions.

The nonlinear nature of the performance as well as the processing time in the order of minutes could be explained by taking into consideration the factors listed below:

- Different types of log entries contain different information; hence, the processing time for different types of log entries is different. That means, the processing time of two log files with the same number of entries will be different unless their corresponding entries are of the same type.
- The parsing algorithm uses the *string comparison* function rather frequently, an operation that is considered to be relatively time-consuming. For example, special symbols (*':', '[', ']', '/'*) are used to tokenize the strings. Tokens are further processed, requiring conversions from string tokens to integers (e.g., convert a string to an integer port number).
- The parsing process is not parallelized, having a single thread parsing one log file entry at a time.
- Requests for insertions and queries to the back-end database are performed frequently. In the current implementation, these are blocking calls.

- There are external calls made to the *ipinfo.io* server during the parsing process. For every session, a new call is made to the *ipinfo.io* server which can partially explain the duration of the parsing of the log file. Additionally, it may attribute to some of the inconsistencies in the parsing duration of the different files. When the calls to the external server were deactivated (i.e., no IP information was requested from the *ipinfo.io* server), the execution time was reduced by approximately 55%.
- The prototype code was developed to adhere to good code readability practices to allow the smooth evolution of the codebase. As a prototype system, the emphasis was on the proof of concept rather than on optimizing algorithms and processes.

The prototype testing findings will be utilized in the next version of *VizAttack*, which will parallelize and optimize both the parsing and the step profiling algorithms.

5.4.3 Attack Postmortem Investigation

VizAttack could be utilized to profile and analyze cyberattacks by reconstructing the steps the attacker followed to execute particular attacks. Executing dynamic queries, one step at a time, allows the construction of an attack tree in a top-down manner. Consider an internal node *X* in the attack tree. All children nodes of *X*, representing the next step taken in an attack, have the same common attack profile up to *X*. A security analyst could derive common patterns in attacks and further investigate their unrelated sub-paths.

Based on the experiments conducted, the maximum height of an attack tree was found to be 59, representing the 59 distinct steps taken to execute an attack. The second highest attack tree was comprised with 50 levels. In order to illustrate the utilization of *VizAttack* in a postmortem attack investigation, details are presented in constructing the attack steps for a *successful unauthorized login into the server that resulted in uploading files to the server and changing their permissions*. The idea is to start with the first step and proceed with the top-down construction of the tree by selecting the next step from a list of available steps.

Step 1:

Figure 5.13 shows the first step of the attack tree: the *(G)-New connection* (leftmost column). This is the root of the attack tree. Command *auth-Occs: 2* (center column) is the only choice to be selected as the second step. It seems that all the successful attacks stored in this particular log file have these two initial steps. The rightmost column lists the attack sessions that resulted in a successful unauthorized login.

Step 2:

Double-clicking *auth-Occs: 2* dynamically creates the options for the next step. Figure 5.14 lists one option, *auth-Occs: 2*. Meanwhile, the attack pathway is updated with the second step (leftmost column).

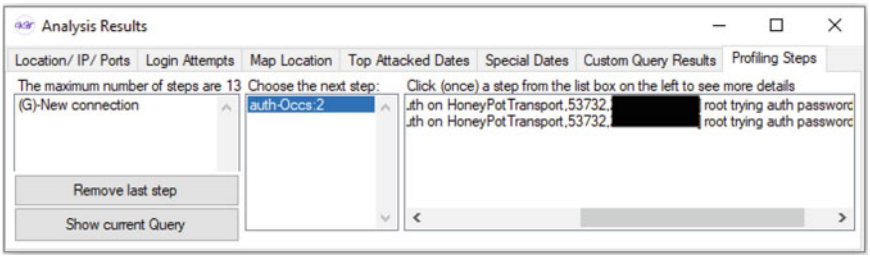

Fig. 5.13 Attack step 1

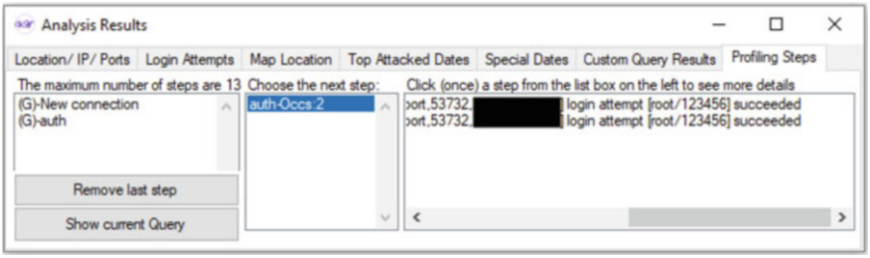

Fig. 5.14 Attack step 2

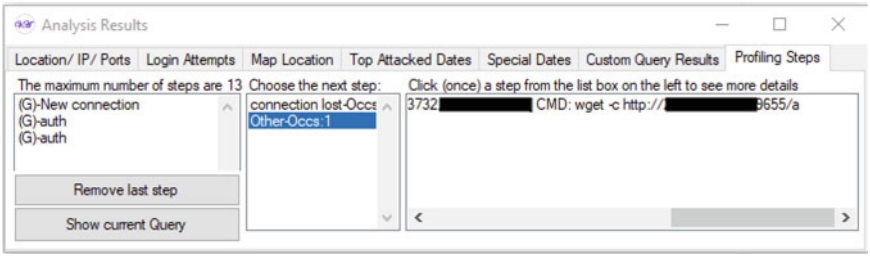

Fig. 5.15 Attack step 3

Step 3:

Double clicking on *auth-Occs: 2* results in an updated attack pathway (three steps) as shown in Fig. 5.15. Once the connection is established, there are two actions available: *Other-Occs: 1* (the specific commands that the attacker is performing are shown in the rightmost column) and *connection lost-Occs*. The former one is the step taken by the attacker who proceeded with uploading files on the server and that's the one that gets selected.

Steps 4–7:

Figure 5.16a illustrates the attack path after the execution of six command steps. The last step designates a lost connection, and thus the attack session is terminated. At any point during the pathway construction, backtracking is allowed by removing steps from the path and selecting new ones. The dynamic execution of queries along

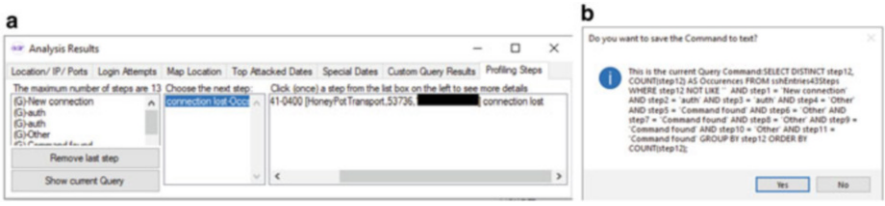

Fig. 5.16 (**a**) Attack steps 4–7. (**b**) Resulting query

with the possibility of backtracking allows the comparison of different strategies followed by attackers. The resulting query that created the attack steps could be saved and reused at a later time (Fig. 5.16b).

5.5 Conclusion

Global security situational awareness is closely coupled with security management, and its effectiveness lies on using a large aperture that process data from multiple multivendor technologies. This chapter presented *VizAttack*, an extensible open-source attack visualization and analysis tool that addresses the challenges currently faced by attack visualization technologies. *VizAttack* integrates in a single framework a set of mechanisms to (1) import and parse attack data sets collected by heterogeneous sources, (2) visualize the data in an intuitive manner, and (3) support execution of customized queries on the archived data. *VizAttack* leverages the attack visualization process by constructing attack profiles, linking the actions executed during an attack session. The step-by-step construction of an attack pathway could be further exploited to simulate an attack tree based on real data, showing all possible ways an attack was realized.

VizAttack could play a significant role in the cyber security management education. Technology integration in course curricula could extend learning in powerful ways, demonstrating the application of theoretical concepts in practice. The *VizAttack* framework could be an integral part of any security-oriented undergraduate course that aims in providing students with the sought-after technical knowledge and skills in attack profiling, emphasizing the importance of analyzing archived attack data to infer the attack methodology used in various attack sessions. The postmortem analysis of an attack offers a useful insight into the attack pathway, allowing the formulation of an attack profile that could be utilized to prevent future attacks based on the same or similar profile. Students should be able not only to form a timeline of the attack steps but also determine what vulnerability was exploited and how and when was it exploited.

A prototype based on the *VizAttack* design principles and objectives was implemented, and its initial performance assessment is promising. It provides a solid foundation to accommodate new *VizAttack* features, such as supporting the

explicit command set of a specific attack profile and correlating attack profiles to extract commonalities. It is planned to integrate additional log file parsers, thus enriching the data sets currently available to *VizAttack*. Enhancements/upgrades to the visualization algorithms as well as optimizations of the parsing and step profiling algorithms are also expected in the new version of *VizAttack.* It is envisioned to unify *VizAttack* and two other related technologies, *HoneyCy* [18] and *SMAD* [19, 20], into a single platform. *SMAD* is novel framework that monitors kernel and system resources data (e.g., system calls, network connections, and process info) based on user-defined configurations that initiate nonintrusive actions when alerts are triggered. The three technologies complement each other and will constitute a solid foundation for the new platform offering advanced attack visualization, analysis, and monitoring capabilities.

References

1. IBM Security and Ponemon Institute LLC. (2019). *2019 cost of a data breach report*. Retrieved December 21, 2020, from https://www.ibm.com/security/data-breach
2. McNabb, L., & Laramee, R. S. (2017). Survey of Surveys (SoS)—Mapping the landscape of survey papers in information visualization. *Computer Graphics Forum, 36*(2), 589–617.
3. Franke, U., & Brynielsson, J. (2014). Cyber situational awareness—A systematic review of the literature. *Computers & Security, 46*(2014), 18–31.
4. Jajodia, S., & Albanese, M. (2017). An integrated framework for cyber situation awareness. In *Theory and models for cyber situation awareness. Lecture Notes in Computer Science* (Vol. 10030, pp. 29–46). Cham: Springer.
5. Husák, M., Jirsík, T., & Jay Yang, S. (2020, August). SoK: Contemporary issues and challenges to enable cyber situational awareness for network security. In *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES 2020)* (Article No.: 2, pp. 1–10). Virtual Event, Ireland.
6. Arbor Networks. (2020). *Arbor networks DDoS attack map*. Retrieved December 21, 2020, from https://www.digitalattackmap.com/
7. Kaspersky. (2020). *Cyber malware and DDoS real-time map*. Retrieved December 21, 2020, from https://cybermap.kaspersky.com
8. Fortinet. (2020). *Fortinet threat map*. Retrieved December 21, 2020, from https://threatmap.fortiguard.com
9. Sophos. (2020). *Sophos threat tracking map*. Retrieved December 21, 2020, from https://www.sophos.com/en-us/threat-center/threat-monitoring/threatdashboard.aspx
10. FireEye. (2020). *FireEye cyber threat map*. Retrieved December 21, 2020, from https://www.fireeye.com/cyber-map/threat-map.html
11. CheckPoint. (2020). *ThreatCloud live cyber-attack threatmap*. Retrieved December 21, 2020, from https://threatmap.checkpoint.com
12. Jajodia, S., & Noel, S. (2010, March). *Advanced cyber attack modeling, analysis, and visualization*. Technical report, George Mason University. Retrieved December 21, 2020, from https://csis.gmu.edu/noel/pubs/2009_AFRL.pdf
13. Fischer, F., Mansmann, F., Keim, D. A., Pietzko, S., & Waldvogel, M. (2008). Large-scale network monitoring for visual analysis of attacks. In *VizSec '08: Proceedings of the 5th International Workshop on Visualization for Computer Security* (pp. 111–118).
14. Goodall, J. R., & Sowul, M. (2009). VIAssist: Visual analytics for cyber defense. In *2009 IEEE Conference on Technologies for Homeland Security* (pp. 143–150). Boston, MA.

15. Dionaea. (2020). *Dionea documentation*. Retrieved December 21, 2020, from https://dionaea.readthedocs.io/en/latest/
16. Baecher, P., Koetter, M., Holz, T., Dornseif, M., & Freiling, F. The Nepenthes platform: An efficient approach to collect malware. In D. Zamboni & C. Kruegel (Eds.), *Recent advances in intrusion detection. RAID 2006. Lecture Notes in Computer Science* (Vol. 4219). Berlin, Heidelberg: Springer.
17. Karasavvas, S. (2020, December). *An extensible open-source framework for attack visualization*. MSc Thesis, Department of Computer Science, University of Nicosia.
18. Christoforou, A., Gjermundrod, H., & Dionysiou, I. (2015). Honeycy: A configurable unified management framework for open-source honeypot services. In *Proceedings of the 19th Panhellenic Conference on Informatics* (pp. 161–164). Athens, Greece, 1–3 October 2015.
19. Sababa, B., Avogian, K., Dionysiou, I., & Gjermundrod, H. (2020). SMAD—A configurable and extensible low-level system monitoring and anomaly detection framework. In K. Daimi & G. Francia (Eds.), *Innovations in cybersecurity education* (pp. 19–38). Cham: Springer International Publishing.
20. Avogian, K., Sababa, B., Dionysiou, I., & Gjermundrød, H. (2020). Leveraging security management with low-level system monitoring and visualization. In *The 2020 International Conference on Security and Management (SAM'20). Transactions on Computational Science & Computational Intelligence: Advances in Security, Networks, and Internet of Things* (pp. 1–14). Springer Nature, Las Vegas, Nevada, USA, 27–30 July 2020.