System for Continuous Collection of Contextual Information for Network Security Management and Incident Handling

Martin Husák
Institute of Computer Science
Masaryk University
Brno, Czech Republic
husakm@ics.muni.cz

Martin Laštovička
Institute of Computer Science
Masaryk University
Brno, Czech Republic
husakm@ics.muni.cz

Daniel Tovarňák
Institute of Computer Science
Masaryk University
Brno, Czech Republic
husakm@ics.muni.cz

ABSTRACT

In this paper, we describe a system for the continuous collection of data for the needs of network security management. When a cybersecurity incident occurs in the network, the contextual information on the involved assets facilitates estimating the severity and impact of the incident and selecting an appropriate incident response. We propose a system based on the combination of active and passive network measurements and the correlation of the data with third-party systems. The system enumerates devices and services in the network and their vulnerabilities via fingerprinting of operating systems and applications. Further, the system pairs the hosts in the network with contacts on responsible administrators and highlights critical infrastructure and its dependencies. The system concentrates all the information required for common incident handling procedures and aims to speed up incident response, reduce the time spent on the manual investigation, and prevent errors caused by negligence or lack of information.

CCS CONCEPTS

• **Security and privacy → Network security**; **Vulnerability management**; • **Applied computing → Operations research**.

KEYWORDS

Cybersecurity, Network Monitoring, Cyber Situational Awareness, Incident Response, Incident Handling

ACM Reference Format:
Martin Husák, Martin Laštovička, and Daniel Tovarňák. 2021. System for Continuous Collection of Contextual Information for Network Security Management and Incident Handling. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3465481.3470037

1 INTRODUCTION

Today's network's rising complexity makes it more and more challenging to be aware of all the devices, services, users, and other entities in the network. Namely, in large and heterogeneous networks of thousands of devices and users, even the network and system administrators may lose track of the assets under their control. The situation is harder for cybersecurity experts, such as the members of the incident response teams (CSIRT/CERT) and security operations centers (SOC). Without proper documentation, the expert knowledge, including the history of security incidents and awareness of the local environment, is also hardly transferable between members of the cybersecurity team.

A specific problem we approach is the continuous collection of contextual information for a security team's needs. Such a team may be equipped with a plethora of tools for intrusion detection or vulnerability assessment. A security incident typically starts with an alert from an intrusion detection system (IDS), discovering a vulnerability, or a report by a user or a third party. There are typically one or more devices involved in such cases, either as a source or target of an attack or as a vulnerable machine. To select the most appropriate incident response procedure, the incident handler (i.e., the member of the cybersecurity team responsible for resolving the incident) needs contextual information on all incident actors. In many cases, the incident handler starts only with an IP address or hostname stated in an IDS alert. Potentially helpful pieces of information include the device's location, contact on its administrator or primary user, name and version of the operating system, and a list of software, services, and vulnerabilities on the system. Further, the incident handler may find it helpful to know if the device is a part or a dependency of the critical infrastructure of an organization and if it has a history of security incidents. By knowing such information, the incident handler may select an appropriate incident response, which may be as simple as sending a warning to the user or administrator, or restrictive, leading to turning the device off or filtering its network communication at a firewall, depending on the severity and potential impact of the incident. Failure to acknowledge the contextual information may lead to restricting devices vital to the infrastructure, wasting the capacities on low-importance incidents, or letting the severe incidents continue.

To approach the problem stated above, we propose a system[1] for the collection of contextual information on the network for the needs of incident response. The system is designed to continuously monitor the network, obtain and update information on hosts in the network, and provide the data in a comprehensive manner. The system is based on active and passive network monitoring, namely network flow monitoring and active probing, and uses commonly available tools and approaches. The system design goal is to use the existing network monitoring infrastructure to provide additional services with minimal overhead. The system is modular and consists of an orchestration service, database, and a set of data-collecting components. Some components use passive network monitoring to enumerate hosts and services in the network,

[1] https://is.muni.cz/publication/1724696/crusoe_observe.zip

ARES 2021, August 17–20, 2021, Vienna, Austria

estimate their dependencies, and fingerprint their operating system and applications, while other components complement them with active probing. Further, the data are correlated with internal and external knowledge, such as local partitioning of the network, history of local security incidents, and third-party vulnerability databases. The system is backed by a graph database, which allows comprehensive representation of the data and high flexibility in correlating the data and adding interesting entities and relations. The system provides the user, e.g., incident handler, with all the collected contextual information on any network entity. The data are timely, regularly updated and cleansed, and enable instantly achieving situational awareness in the incident response.

This paper is structured into five sections. After the introduction, we summarize the background and related work in Section 2. The design and implementation of the system are presented in Section 3. Section 4 summarizes the experience from the deployment of the system in a live environment. Section 5 concludes the paper and paves the way for future work.

2 BACKGROUND AND RELATED WORK

Background and related work to the topics discussed in this paper can be viewed in three dimensions, theoretical, procedural, and technological. The theoretical dimension is derived from the theory of decision-making and stresses the importance of having the right data at the right time. The procedural dimension emerges from the processes of incident response in cybersecurity teams and pinpoints particular requirements on the proposed system. The technological dimension contains the tools and approaches used to collect the required information via active and passive network monitoring.

2.1 Cyber Situational Awareness

This work was vastly inspired by the theoretical foundations of situational awareness and decision-making. Situational awareness was extensively studied in military and aviation and is gaining recognition in cybersecurity as cyber situational awareness (CSA) [7]. Following the widely-used definition, *(cyber) situational awareness is the perception of the elements in the (cyber) environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future* [3, 6, 10]. CSA builds upon the perception of the cyber environment, without which one cannot fully understand the situation nor project its changes. An alternative term is network-wide situational awareness [10, 17].

Similarly, decision-making theory describes models illustrating the need for monitoring and information collection and places it as a prerequisite for making a decision. For example, the incident handler may follow the OODA loop (Observe, Orient, Decide, Act) [24], in which there is first the need to collect the information (Observe), analyze them (Orient) before making a decision (Decide), and acting accordingly (Act). If the conditions change, the loop returns to the Observe phase and the perception of a new set of information.

Achieving CSA requires a vast range of heterogeneous information provided by a plethora of existing tools [4]. Liu et al. [17] stated that multi-source, heterogeneity, and real-time are the most important characteristics a network security situational awareness system should embody. However, it is difficult to correlate the data from heterogeneous sources in real-time, namely because many of them produce big data [6]. Nevertheless, complex tools to enable CSA were proposed in the past, such as Cauldron [8] or CyGraph [21]. However, such complex tools are often too complicated to be used effectively, namely due to the high number of required inputs that could not always be provided in practice [9].

2.2 Cybersecurity Incident Handling

Incident handling is a fundamental activity provided by a cybersecurity team. The procedures of incident handling are well structured and formalized in the literature and typically adjusted to the incident response teams' local environment. However, each environment is different, and each team uses a different set of tools to gather data, which makes it hard to come up with technical details, best practices, and supporting tools. Nevertheless, the key principles are well documented and widely adopted by practitioners.

The Computer Security Incident Handling Guide [2] by NIST promotes situational awareness in several aspects, mainly in terms of collaboration between cybersecurity teams and user awareness. The most important message is that *"an organization can best quantify the effect of its own incidents because of its situational awareness."* Further, there is a need to ensure that incident response procedures are in sync with business continuity processes. The business continuity planning professionals should be made aware of the incidents and their impact and are valuable in planning responses to certain situations. However, gaining and maintaining situational awareness, including the knowledge of the business continuity, does not scale well and is problematic in large-scale networks.

The Incident Handler's Handbook [11] by the SANS Institute does not mention situational awareness specifically but includes highly relevant entries in the incident handler's checklist. For example, the incident handlers are reminded to identify not only the source and location of the incident but also its business impact and its scale. Further, the incident handler should plan the following steps (containment, eradication, recovery) by questioning whether the impacted systems can be isolated, patched, turned off, or kept running. The incident handler has to be aware of the impacted hosts, their neighborhoods, and dependencies to properly plan the course of action. Similarly, the incident management guide by ENISA [19] provides practical examples of situational awareness-related issues of incident handling without addressing them specifically. Readers are kindly referred to all three documents for more information.

2.3 Active and Passive Network Monitoring

CSA can be achieved by processing a vast range of heterogeneous data from various tools, including IDS, vulnerability scanners, and network traffic analysis. For example, the information on vulnerabilities involves discovering vulnerabilities in the network (e.g., via dedicated vulnerability scanners) and consulting third-party databases with detailed information on known vulnerabilities. An example of a measurable event is an attack signature detected and reported by an IDS.

In this work, we propose an approach based on network-wide active and passive monitoring. Passive monitoring is represented by network flow (NetFlow) technology [5], a popular tool among the cybersecurity community. NetFlow monitoring aggregates network connections into 5-tuples (source IP, destination IP, source

port, destination port, protocol) and collects additional information, such as length of the connection or the number of bytes and packets exchanged. Although NetFlow does not process the packet payload, it provides sufficient visibility into network traffic even in high-speed and large-scale networks; and can be used for intrusion detection [22] and traffic analysis, including the analysis of encrypted traffic [16]. It is worth mentioning that the tools to support situational awareness using NetFlow were proposed in the past. However, such tools focused more on the visualization of network connections, such as in NVisionIP [12] and VisFlowConnect [23], or the combination of intrusion detection, service detection, and visualization, such as in Nfsight [1]. In our work, we focus on the combination of data from multiple sources and their correlation, not solely on NetFlow data, and we leave visualization for future work as it is a hard task on its own.

Active network monitoring is represented by a well-known Nmap tool [18]. Active probing can be used to check the liveness of a host, discover open network services, or get a host's fingerprint. It is also a popular choice for vulnerability scanners, e.g., tools that discover vulnerabilities on hosts in the network [15]. Both active and passive network monitoring have their benefits and drawbacks, but the combination of their outputs can be very powerful.

3 SYSTEM DESIGN AND IMPLEMENTATION

In this section, we describe the proposed system for the continuous collection of contextual information for network security management. First, we provide a design overview and a description of common components. Second, we describe the design of the data collection component that perform the crucial tasks.

The requirements on the proposed system were distilled from the requirements on CSA systems in the NATO Cyber Defense Situational Awareness Request for Information [20]. The 35 use cases frequently mention views on assets and their interconnectivity, dependencies, and historical incidents. We also embraced the issues of system modularity and deployability. On the contrary, we did not consider the use cases related to visualization, training, and simulation because we perceive them as out of the scope of our work or candidates for future work. The interviews with practitioners [9] added two more requirements. First, the contacts on persons responsible for the administration of hosts and services in the network should be easily accessible for incident handlers, which is an issue in large networks with many departments and their local IT staff. The incident handlers often lose time on finding the right person to resolve an incident with. Second, the incident handlers would appreciate the automation of triage, i.e., prioritizing the incidents by their severity and significance [19].

It is worth mentioning that the requirements are not strict. Even with lower confidence, any piece of information is valuable as it provides at least some context. The goal is not to provide the incident handler with all the information at the highest quality because that would be enormously difficult to achieve. The goal is to gather the basic information on hosts in the network that would otherwise be unknown to the incident handler. Thus, in this case, we prefer a network-wide solution with lower precision over a specialized solution with limited scope.

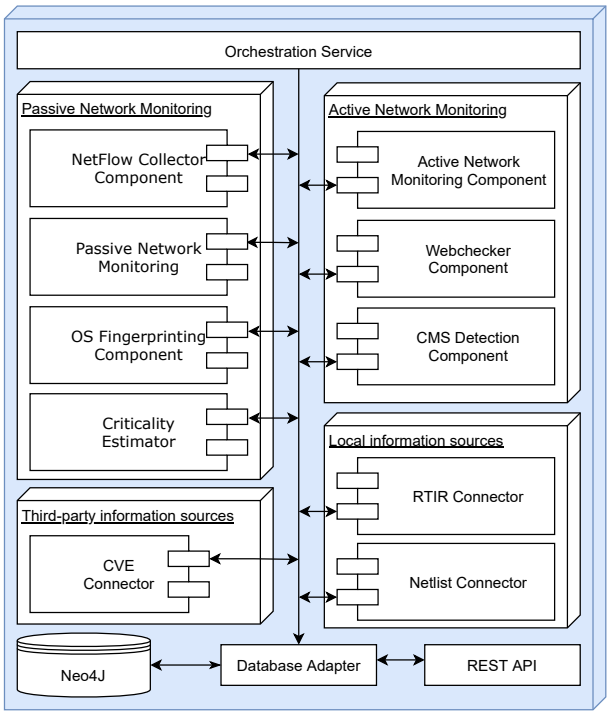

Figure 1: System scheme and its components.

A typical user of our proposed system is a cybersecurity team of an organization that operates a large computer network (e.g., thousands of computers and users). Such networks are often heterogeneous and partitioned into many segments of various characteristics (e.g., servers, desktops, organization's departments). While the network, hosts, and services can be operated by numerous local or network-wide IT administrators, we assume a single cybersecurity team responsible for the security of the whole network or organization. Further, we assume that the team has the capabilities to monitor the network via passive network traffic monitoring and is allowed to perform active network monitoring (scanning, probing).

3.1 Design and Common Components

The software is modular and consists of common components and a set of data collection components. The scheme of the system is depicted in Figure 1. The common components are the orchestration service, database, and database adapter. The orchestration service schedules and executes the runs of the data collection components. The data produced by the components are saved in the shared database via the database adapter. The data collection components are grouped in several logical blocks; each block's components provide similar functionality (e.g., asset discovery or vulnerability assessment) or process the same input data (e.g., network flows or incident history). The data collection components are described in detail in the following subsection.

The orchestration service is responsible for running the data collection components; each run is referred to as a task. Each task has its role, which defines its timing and execution parameters. The records on the execution are logged, so it is possible to trace if the

task finished or failed, execution time, output, and possible errors. When the orchestration service runs a task and the task finishes successfully, the orchestration service is responsible for storing the outputs into the database by calling the database interface. The orchestration service itself does not implement any business logic or data processing apart from forwarding the data from components to the database.

The implementation of the orchestration service is based on Celery task queue[2], supported by the monitoring system Flower[3] and in-memory database Redis[4]. An interesting feature of the orchestration system is managing a computing cluster using the master-worker architecture. One master orchestrates the tasks in such a case, and one or more workers run the tasks. This is advantageous for several components, namely, for the components of active and passive network monitoring. For example, the NetFlow data are large in volume, so it is more efficient to run the worker on the NetFlow collector than access the data remotely from the master. In the case of active network monitoring, we may take advantage of several observation points in the network and run the workers on one or more hosts that provide better visibility in the network, e.g., due to a lack of firewalls between the observation points and the scanned hosts.

The system is backed by a graph database, in which the data are stored in the form of a graph with identified entities (e.g., IP addresses, software instances, and vulnerabilities) as nodes and their relations as edges. The graph-based approach facilitates data modeling and extensibility of the database and is easy to visualize and comprehend. In our system, we use the Neo4j database[5], one of the most technologically mature solutions at the moment. A database adapter was implemented to receive new data from data collection components via REST API and insert them into the database using the Cypher language used in Neo4j. Thus, the data collection components are independent of the database implementation (they do not have to create their own Cypher commands), and the database can be fully replaced if needed. The data (entities and relations) are structured according to the CRUSOE data model [9]. In practice, however, several minor corrections had to be made in the data model, and many particular parameters of nodes and edges had to be specified, namely to facilitate data input from the tools and components used for data collection. Further, the data are accessible via another REST API that implements the endpoints providing CRUD (Create, Read, Update, Delete) functionality for the entities specified by the data model. Both REST APIs are implemented using the Django framework[6].

3.2 Data Collection Components

The data collection components conduct the collection of the particular pieces of information. Each component operates independently, except for the NetFlow-processing components that rely on the data provided by the NetFlow collector component. However, the components often overlap in the types of data they insert into the database. In most cases, they fill in additional parameters to entities and relations identified by an arbitrary component. In several cases, the same information can be obtained by more components, such as in the case of OS or service fingerprinting and vulnerability discovery. However, this is more of an advantage; the components rather complement each other; some may provide more detailed or precise information, while others may have a broader range. Further, it is up to the operators to select which components to use, replace, or implement on their own. The list and description of existing components follow.

3.2.1 NetFlow Collector Connector. This connector is a supporting component that facilitates NetFlow data processing. While most of the components either collect the data on their own or run a tool to collect them, it is not straightforward to do so with the NetFlow data used in passive network monitoring. In a typical setting, NetFlow records are generated by a router or a dedicated device (NetFlow probe) and forwarded to a NetFlow collector, where they are stored and analyzed. The NetFlow data are large in volume and, thus, difficult to handle. If our proposed system is deployed on a dedicated host, there is a need to either enable remote access to the NetFlow collector's data or copy them to the dedicated host. We implemented a component that remotely accesses the NetFlow collector via SSH and queries the NetFlow data via the Nfdump tool that should be installed on the collector. Thus, the NetFlow-processing components can access and query the data remotely and receive only the data that are required by the component. Although commercial NetFlow collectors provide REST API or other options, SSH remains a usable, vendor-independent solution.

3.2.2 Passive Network Monitoring Component. This component analyzes network traffic records to detect running hosts in the network and identify their services and software. The component's output is a list of active IP addresses and their identified services and pieces of software. The component consists of three modules (*run*, *core*, and *rules*) that logically separate the data processing steps. Further, the component contains three subcomponents that implement particular analytical features. The subcomponents are operated separately, and their outputs are then merged into one common output forwarded to the database.

First, the *run* module loads input data, initializes and runs the subcomponents while providing them with input data, and saves the subcomponents' outputs in a file. Second, the *core* module provides common functions to the subcomponents, such as manipulating the input data and the functions and data structures to save the outputs. The software identification is formatted in CPE[7] strings (vendor:product:version) to facilitate their later processing. Finally, the *rules* module provides the support for subcomponents that use algorithmically sophisticated techniques. The module enables specifying a set of rules in a JSON file. Each rule specifies a set of conditions in the form of regular expressions, numerical values, and value ranges. If the conditions are met (i.e., the regular expressions are matched in the network flows and numerical values fall in a specified range), a set of information keys specified in the rules is put on the output.

[2] https://docs.celeryproject.org/en/stable/
[3] https://flower.readthedocs.io/en/latest/
[4] https://redis.io/
[5] https://neo4j.com/
[6] https://www.djangoproject.com/
[7] https://nvd.nist.gov/products/cpe

The first subcomponent detects which antivirus software is running on hosts in the network. The detection method uses a predefined list of rules describing network connections to hosts and services associated with particular antivirus software, such as a network connection to an update repository. For example, if a machine initiates a connection with the vendor's server, the machine is marked as running the vendor's antivirus software. If more rules are matched, the antivirus associated with the most frequently matched rule is assigned to the machine.

The second subcomponent identifies the web browser used on the machine. Two methods are used here. First, similarly to antivirus detection, a communication with IP addresses, domain names, and services specific to a particular browser (update server, cloud services) is described in the rules. Second, the User-Agent can be extracted from HTTP communication. The User-Agent string is parsed, and the identification of the browser is extracted.

Finally, the third subcomponent uses machine learning to identify services in the network. The machine learning method is first trained using the network flows containing NBAR2 signatures[8]. A classifier is constructed using the features, such as traffic volume, packet size, packet distribution over time, and TCP/IP header settings. Thus, the method is capable of estimating the services and software running on hosts in the network.

3.2.3 Passive OS Fingerprinting Component. Passive OS fingerprinting could be implemented as a subcomponent of the passive network monitoring component. However, the task is an interesting research problem going through rapid development [13, 14, 16], and, thus, the component is developed separately. The goal is to identify the operating system of a machine in the network by analyzing the machine's network traffic. Three major methods were described in related work, detection of communication with specific domains, analysis of HTTP User-Agent, and analysis of default values of TCP/IP header fields [13]. The TCP/IP header analysis was further improved with machine learning [14]. The implementation uses all three of them, their outputs are combined and the most frequently identified OS is selected as a result. The specific domains include Internet connection check services, update servers, telemetry, and other communication types, based on expert knowledge. The method based on the analysis of TCP/IP headers uses three parameters, IP TTL, TCP SYN Size, and TCP Win Size, which all have default values specific to particular operating systems. The User-Agent analysis is getting obsoleted and will soon be replaced due to the still growing adoption of encrypted network traffic, which prevents passive analysis of HTTP traffic parameters [16].

3.2.4 Active Network Monitoring Component. Active network monitoring (scanning, probing) complements passive network measurements; the two approaches overlap in the enumeration of active hosts and services in the network and their fingerprinting. The active probing further enables network topology discovery. Nmap [18] is the well-known and widely-used tool for active probing with many additional features. We designed a component consisting of three modules.

[8]https://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html

The *Scanner module* was designed to scale for large networks, in which the scanning may take a non-trivial amount of time. The design takes advantage of parallel scanning from multiple devices and splitting the scanned IP range into smaller parts that can be processed separately and merged into a common output. Two types of scans are performed, network scan and topology scan. First, the network scan is executed to discover running hosts and services in the network and to identify the software they use. The 100 most commonly used ports (Nmap's *--top-ports* [18]) are scanned on each IP address. If an open port is found, the advanced features of Nmap are used to further probe the service on the port to identify its software and version and export it as a CPE string. In the context of this work, the most important use case is the search of the CPE in vulnerability databases to find vulnerabilities that may be present on a system identified by the CPE [15]. Second, the topology scan is executed to map the network topology and distinguish active network devices from end hosts. The scan is executed using Nmap's traceroute method, not via the *traceroute* Linux command, to allow for scanning on particular ports. In order to bypass firewall rules and cover as much IP address space as possible, the scanning uses TCP ports 80 and 443. Only if those ports are filtered, the default ICMP traceroute is used. The scanning from one observation point discovers the paths between the scanning host and the scanned hosts. However, such a scan produces only a tree structure that does not reflect redundant paths and cycles in the network topology. Thus, there is a need to execute the scanning multiple times from multiple observation points located in distinct physical and logical segments of the network. The more observation points and repetitions are used, the more is the network topology map similar to the actual network topology.

The *Parser module* reads the outputs of Nmap and searches the entities and relations specified by the database model. The scanning is performed at multiple locations, and, thus, the parser has to merge the outputs and correctly add new entities and update existing ones.

Finally, the *Cleaner module* deals with the problem related to scanning in discrete time intervals. The hosts and services in the network can be arbitrarily deactivated or stop responding. Thus, the scans may be interrupted in the process. Namely, in the case of traceroute scanning, which takes a longer time, there is a need to ensure that lost connections and unfinished scans will not be included in the results.

3.2.5 Webchecker Component. In recent years, due to the wide adoption of HTTPS, a new task for cybersecurity teams emerged; it became important to check if the websites are accessible via HTTP or HTTPS and if it has a valid TLS certificate. Certain websites should be accessible only via HTTPS, while expired certificates denote a poorly maintained system that poses a security risk. Information on the accessibility of the website and validity of their certificates are important contextual information and be collected continuously. Thus, we designed and implemented a component that performs such checks.

The component uses the data from passive network monitoring to enumerate the active websites in the network. For each established network connection over HTTP(S) observed in the network, it adds the destination IP address to the list of webservers. Subsequently, the component actively checks if the webserver listens on

ports 80 (HTTP) or 443 (HTTPS). If the server responds on HTTPS, the provided certificate is validated. Thus, the component provides a list of all webservers in the network and marks if they use HTTP, HTTPS, or both, and if they have a valid certificate and when the certificate will expire.

3.2.6 CMS Detection Component. This component complements the active network monitoring via Nmap, which is capable of discovering webservers and identifying their software (e.g., Apache or IIS) but cannot analyze the web content or web application. Content Management Systems (CMS) are popular tools to manage web content and can be found on numerous websites. However, they are also frequent targets of cyber attacks due to their wide deployment, numerous vulnerabilities in them and their plugins, and the fact that they are often out of date and unpatched. Therefore, it is advantageous to know if there is a CMS operated on the websites in the network, at what version, and with what plugins. Fortunately, identification or fingerprinting of a CMS is not a complicated task, and there are many available tools, such as *WhatWeb*[9]. The component retrieves the list of target websites and then runs WhatWeb to actively scan the web and identify any deployed CMS. The component then maps CMS names and versions to the CPE format.

3.2.7 CVE Connector. This component gathers information on vulnerabilities in the CVE format[10] from several external sources. The purpose of this component is to put together available information on vulnerabilities discovered in the network so that the incident handler can be immediately provided with a human-readable description and technical details on each vulnerability with an assigned CVE identifier. NVD[11], is a *de facto* standard library of vulnerabilities. The data on vulnerabilities are regularly downloaded from the *Data Feeds* on the NIST website. Although NVD information is sufficient in most cases, several databases are operated by software vendors that focus on providing more detailed information on particular vulnerabilities in the vendor's products. The descriptions are often much more detailed than in NVD, and they typically contain timely information on the vulnerability status, including the exact software version or patch that fixes the vulnerability. The date of publication of the vulnerability may also differ from NVD. The vulnerabilities are indexed by the same CVE identifier, except for cases where the vulnerability is first published in a vendor's database. It is highly advantageous to use such information; the extended technical descriptions and information on patches' availability facilitate and speed up vulnerability handling. Our system implements modules that download data from vulnerability databases of Adobe, Android, Apple, Cisco, Lenovo, Microsoft, Oracle, and RedHat.

3.2.8 Criticality Estimator. This component implements an experimental approach to the automated detection of critical points in the computer network. The goal of such a task is to identify which network hosts are critical for the organization. A critical host may be part of the organization's critical infrastructure, provider of a frequently used service, or otherwise important host. It is possible to annotate hosts in the network manually, but it is laborious, does not scale well, and is prone to omissions. Thus, we designed an automated tool that uses graph theory to identify critical hosts in the network. The underlying methods are interesting from the research perspective and shall be published separately in the future.

The network topology forms a graph that is a subgraph of the complex graph stored in the database. We define *critical host* in the network as a host that has a high impact on the flow of information in the network. A *critical host* is represented as a *critical node* in the network topology graph. We assume that removing a *critical node* would decrease or remove the graph's connectivity in terms of graph theory. We apply two graph-theory algorithms to assess nodes' features. First, the algorithm to assess the flow of information through the node is based on the betweenness algorithm implemented in Neo4j. The algorithm computes the shortest paths between each pair of nodes via breadth-first search. Subsequently, each node is weighted by the number of shortest paths going through that node. Nodes with the highest weight are considered the most important (critical) for the network. Second, the node popularity algorithm is based on enumerating the incident edges of each node. The implementation is based on Neo4j's degree centrality algorithm; it first selects the set of nodes and then enumerates all the incoming and outgoing edges of eligible types. Each node is weighted by the number of incident edges. Nodes with higher weight are considered popular and, therefore, possibly critical.

3.2.9 RTIR Connector. The history of cybersecurity incidents associated with a network entity is highly interesting contextual information. Cybersecurity teams typically use incident management or ticketing system to keep track of currently handled incidents. A widely known example of such a tool is RTIR[12]. In RTIR, there is a ticket created for each security incident. Each ticket contains a history of communication between the incident handlers and users or IT administrators. The RTIR component periodically checks the status of incidents and updates the records on the entities' incident history in the database. Although the incident handler would work primarily with RTIR, which provides a search engine and all the details on the incidents, it is still advantageous to keep track of a brief history of incidents in the database so that it is kept together with other contextual information and linked with other entities.

3.2.10 Netlist Connector. This connector is a simple network partitioning connector that provides information on network segments and contacts of local IT administrators. Such tasks fall into asset management, which is highly organization-specific and hard to transfer. In the sample implementation, the connector parses a CSV file that contains a list of network segments with names and responsible contacts attached. The segment may correspond to IP address ranges assigned to particular departments, special use segments (e.g., VPN, Wi-Fi), or any other network's administrative segmentation. Potential users of the system are free to implement their own component or connect it to their asset management system.

4 EVALUATION IN LIVE ENVIRONMENT

The system was deployed in a live environment for evaluation. Herein, we first briefly describe the testing environment. Subsequently, we discuss the volume of the collected data in the database

[9] https://github.com/urbanadventurer/WhatWeb
[10] https://cve.mitre.org/
[11] https://nvd.nist.gov/
[12] https://bestpractical.com/rtir

and its performance. Finally, we comment on the crucial issue in the system, which is the vulnerability assessment quality.

4.1 Testing Environment

The system was deployed in a campus network of Masaryk University in collaboration with the university's cybersecurity team CSIRT-MU. The campus network is large, heterogeneous, administered by multiple local IT teams at ten faculties, two research institutes, and thirteen other departments. Altogether, the network serves 30,000 students and 4,000 employees. Each day, around 15,000 IP addresses are active in the university /16 IPv4 address range. The number of active IP addresses observed during one month rises to 25,000. The network generates around 19,000 network flows per second, which corresponds to the data flow of 20 Gb/s. The network is open and publicly addressable by default. However, there are over 500 network segments with their own firewall rules and policies.

The security team operates a network monitoring infrastructure. Dedicated NetFlow probes are located at all the points that connect the campus network with the Internet and selected points inside the network. The NetFlow data are exported to the collectors that were remotely accessible by our system. The measurement infrastructure collected NetFlow records extended by HTTP parsers, which allowed for the monitoring of User-Agents.

Our system's testing took place on two servers with different hardware configurations. The first, more powerful server had eight cores of Intel Xeon CPU E5-2680 v3 2.50GHz and 32 GB of RAM and served as the master node running the orchestration service, the database, and several components. The second server, called the worker, had four Intel Xeon CPU E5-2680 v2 2.80GHz cores and 16 GB of RAM and fulfilled the role of the observation point located in a distinct network segment to verify the active scanning components. The data collection components were set to run every five minutes due to the typical setting of NetFlow collectors [5] and the reliance of many of the components on the NetFlow data. Only the active network monitoring took too long to finish and, thus, was run every 16 hours.

4.2 Database Content and Performance

The database is a central component of the proposed system, and its performance is crucial for the whole system. Neo4j turned out to be a technologically mature choice with sufficient performance on the hosting machine. From the performance perspective, the most important task is the insertion of new data collected by passive network monitoring components. Every five minutes, thousands of nodes and their incident edges are added or updated. The system was under development for more than one year without deleting older entries. Over time, we figured out that if the database contains the data from the past three months (2-3 GB for the particular network), there are no performance issues. Larger volumes of data cause delays and reduce the quality of the service. Thus, we added a cleanup component to the orchestration service that deletes records older than three months.

It is worth mentioning that the evaluation used the Community edition of Neo4j on a single host. We also conducted an experiment in which we used the Enterprise edition of Neo4j, which turned out to be more powerful in terms of indexing capabilities and memory management. There were no performance differences in inserting small amounts of data (tens to hundreds of nodes or edges) or querying the data regardless of query complexity and result size. However, it took the Community edition 70 % more time to insert the data from the passive network monitoring component and 110 % more time to insert data from OS fingerprinting component; both components add or update up to 29,000 nodes and tens of thousands of edges every five minutes. Further, the Community edition often crashed on low memory when more than three months of data were stored in the database. Running the database in a cluster of multiple hosts did not provide any acceleration.

Our measurement shows that three months of data collected in the last quarter of 2020 consist of 697,783 nodes and 22,119,299 edges, and the overall size of the database is 2.5 GB. Out of this number, we collected information on 29,535 unique IP addresses from the /16 range of the network that hosted 76,763 network services. We also observed 483,449 security events, including the discovery of 2,404 vulnerabilities mapped to 563 unique CPE identifiers.

4.3 Vulnerability Assessment

Vulnerability assessment based on the methods implemented in the system was described in our previous work [15]. Herein, we would like to pinpoint the major differences between the vulnerability assessment conducted by different components and combinations of their outputs. Passive or active network monitoring components provide the CPE strings describing the software identified on the network hosts. The CPE strings are matched with the CPE strings attached to CVE records. If a match is found, the host is declared as potentially vulnerable.

Our measurements confirm the results of related work, namely that there are major differences in coverage and reliability of the data and vulnerability types. While passive network monitoring can detect and identify almost all the running hosts in the network throughout the measurement period, the vulnerability mapping is imprecise and with fewer details. In most cases, such methods correctly estimate the operating system of a host but not its version. Thus, only generic vulnerabilities of OS are assigned to the host. For example, passive network monitoring identified 18,749 actively communicating IP addresses (they sent at least one UDP or TCP SYN packet), out of which 18,018 could be fingerprinted to detect its OS. In 10,039 cases, vendor, product, and version were identified, while in 7,813 cases, only the vendor and product were identified. However, the CPE strings like *cpe:/o:linux:linux_kernel:** or *cpe:/o:microsoft:windows_10:** are not descriptive enough to allow meaningful correlation with CPE strings in CVE records. Further, it was shown in our previous work [13] that the precision of OS fingerprinting is only around 85 %, which should be considered when interpreting the results.

On the contrary, active network monitoring better identifies the OS and software behind the host's network services, including their versions. However, the scanned hosts need to be running and accessible over the network, e.g., not behind a firewall or in a private network segment. Thus, only a fraction of hosts in the network is scanned. For example, the scanning with Nmap version 6.47 and *-sV -n -T5* parameters took over 16 hours to discover 5,771 running hosts, out of which 3,152 were observed to have all ports

closed or *filtered*. Thus, only 1,620 hosts could be assigned a CPE string with the vendor, product, and version, and additional 252 hosts could be assigned a CPE with only vendor and product. As we can see, the numbers are significantly lower than with passive monitoring. A similar approach was also tested with the WhatWeb scanner (version 0.4.9) that discovered 382 websites hosted in the network and generated 304 CPE strings describing vendor, product, and version and 56 CPE strings describing only vendor and product of a CMS or similar software. Although such numbers appear too low, the tool's coverage is excellent and sufficient for vulnerability assessment of web applications.

Dedicated vulnerability scanners were not used in the evaluation. Although popular tools exist, they could not be easily used in large-scale networks due to the long time they would need to scan the network or their pricing, which makes them too expensive for experiments. Nevertheless, we expect them to provide higher precision and level of detail than generic active scans, but at most the same number of outputs.

The third-party tools and services, such as Shodan[13] or Shadowserver[14], have even lower visibility into the network. There is a varying level of data quality among such services, and the information might be outdated, but the major problem is the low number of entries. In our experience, tens of vulnerable hosts, at most, can be discovered per month using such services. Nevertheless, vulnerable hosts accessible from the Internet may pose a more prominent threat than an unconfirmed vulnerability on a host in a private network, and, thus, using such tools should be encouraged.

5 CONCLUSIONS

In this paper, we presented a design and implementation of a system that facilitates CSA for incident handlers of a CSIRT/CERT or SOC. The system continuously collects contextual information on the computer network, namely the enumeration of hosts, services, dependencies, and vulnerabilities. Such information enables the incident handlers to resolve incidents quickly, speed up decision-making, and prevent mistakes caused by low situational awareness. The system effectively combines the data from existing network monitoring infrastructure, tools, and local knowledge to collect the essential information on the whole network.

In our future work, we will elaborate more on the procedural aspects of incident handling in relation to CSA. We are going to formally integrate the system into incident handling procedures and establish metrics to qualify and quantify the influence of the proposed system on incident handling practice.

ACKNOWLEDGMENTS

This research was supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

REFERENCES

[1] Robin Berthier, Michel Cukier, Matti Hiltunen, Dave Kormann, Gregg Vesonder, and Dan Sheleheda. 2010. Nfsight: NetFlow-based Network Awareness Tool. In *Proceedings of LISA'10: 24th Large Installation System Administration Conference*. 119–134.

[2] Paul Cichonski, Tom Millar, Tim Grance, and Karen Scarfone. 2012. Computer Security Incident Handling Guide: Recommendations of the National Institute of Standards and Technology. *NIST Special Publication* 800, 61 (2012), 1–147.

[3] Mica R. Endsley. 1995. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors* 37, 1 (1995), 32–64.

[4] Antti Evesti, Teemu Kanstrén, Tapio Frantti, Teemu Kanstren, and Tapio Frantti. 2017. Cybersecurity Situational Awareness Taxonomy. In *2017 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)*. IEEE.

[5] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. 2014. Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2037–2064.

[6] Martin Husák, Tomáš Jirsík, and Shanchieh Jay Yang. 2020. 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* (Virtual Event, Ireland). ACM, Article 2, 10 pages.

[7] Sushil Jajodia, Peng Liu, Vipin Swarup, and Cliff Wang. 2010. *Cyber situational awareness*. Vol. 14. Springer.

[8] S. Jajodia, S. Noel, P. Kalapa, M. Albanese, and J. Williams. 2011. Cauldron Mission-centric Cyber Situational Awareness with Defense in Depth. In *2011 - MILCOM 2011 Military Communications Conference*. 1339–1344.

[9] Jana Komárková, Martin Husák, Martin Laštovička, and Daniel Tovarňák. 2018. CRUSOE: Data Model for Cyber Situational Awareness. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM, Article 36, 10 pages.

[10] Alexander Kott, Norbou Buchler, and Kristin E. Schaefer. 2014. *Cyber Defense and Situational Awareness*. Vol. 62. Springer. 29–45 pages.

[11] Patrick Kral. 2012. The Incident Handler's Handbook. https://www.sans.org/reading-room/whitepapers/incident/incident-handlers-handbook-33901.

[12] Kiran Lakkaraju, William Yurcik, and Adam J. Lee. 2004. NVisionIP: Netflow Visualizations of System State for Security Situational Awareness. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security* (Washington DC, USA). ACM, 65–72.

[13] Martin Lastovicka, Tomas Jirsik, Pavel Celeda, Stanislav Spacek, and Daniel Filakovsky. 2018. Passive OS fingerprinting methods in the jungle of wireless networks. In *2018 IEEE/IFIP Network Operations and Management Symposium*.

[14] Martin Laštovička, Antonín Dufka, and Jana Komárková. 2018. Machine Learning Fingerprinting Methods in Cyber Security Domain: Which one to Use?. In *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*. 542–547.

[15] Martin Laštovička, Martin Husák, and Lukáš Sadlek. 2020. Network Monitoring and Enumerating Vulnerabilities in Large Heterogeneous Networks. In *2020 IEEE/IFIP Network Operations and Management Symposium*.

[16] Martin Laštovička, Stanislav Špaček, Petr Velan, and Pavel Čeleda. 2020. Using TLS Fingerprints for OS Identification in Encrypted Traffic. In *2020 IEEE/IFIP Network Operations and Management Symposium*.

[17] Xiaowu Liu, Huiqiang Wang, Jibao Lai, and Ying Liang. 2007. Network security situation awareness model based on heterogeneous multi-sensor data fusion. In *2007 22nd international symposium on computer and information sciences*.

[18] Gordon Fyodor Lyon. 2008. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US).

[19] Miroslaw Maj, Roeland Reijers, and Don Stikvoort. 2010. Good Practice Guide for Incident Management. https://www.enisa.europa.eu/publications/good-practice-guide-for-incident-management.

[20] Tamsin Moye, Reginald Sawilla, Rodney Sullivan, and Philippe Lagadec. 2015. Cyber Defence Situational Awareness Demonstration/Request for Information (RFI) from Industry and Government (CO-14068-MNCD2). *NCI Agency Acquisition* (2015).

[21] S. Noel, E. Harley, K. H. Tam, M. Limiero, and M. Share. 2016. CyGraph: Graph-Based Analytics and Visualization for Cybersecurity. *Handbook of Statistics* 35 (2016), 117–167.

[22] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. 2017. Flow-based intrusion detection: Techniques and challenges. *Computers & Security* 70 (2017), 238–254.

[23] Xiaoxin Yin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju. 2004. VisFlowConnect: Netflow Visualizations of Link Relationships for Security Situational Awareness. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security* (Washington DC, USA). ACM, 26–34.

[24] Robert Zager and John Zager. 2017. OODA loops in cyberspace: A new cyber-defense model. *Small Wars Journal* 20, 11 (21 October 2017).

[13] https://www.shodan.io/

[14] https://www.shadowserver.org/