# Intrusion Detection System for Home Windows based Computers

**Matej Zuzčák[1*], Tomáš Sochor[1], and Milan Zenka[1]**

[1] Department of Informatics and Computers, Faculty of Science University of Ostrava, Ostrava 70103, Czech Republic.
[e-mail:{matej.zuzcak, tomas.sochor, milan.zenka}@osu.cz]
*Corresponding author: Matej Zuzcak

*Received September 9, 2018; revised December 24, 2018; accepted January 17, 2019; published September 30, 2019*

## Abstract

The paper is devoted to the detailed description of the distributed system for gathering data from Windows-based workstations and servers. The research presented in the beginning demonstrates that neither a solution for gathering data on attacks against Windows based PCs is available at present nor other security tools and supplementary programs can be combined in order to achieve the required attack data gathering from Windows computers. The design of the newly proposed system named Colander is presented, too. It is based on a client-server architecture while taking much inspiration from previous attempts for designing systems with similar purpose, as well as from IDS systems like Snort. Colander emphasizes its ease of use and minimum demand for system resources. Although the resource usage is usually low, it still requires further optimization, as is noted in the performance testing. Colander's ability to detect threats has been tested by real malware, and it has undergone a pilot field application. Future prospects and development are also proposed.

***Keywords:*** Network intrusion detection system, IDS, packet, threat, threat analysis, signature.

## 1. Introduction

At present, a host of security threats come from the Internet (e.g. in 2016, about 357 million new malware variants emerged according to [1]. The threats are classified according to their dissemination vector, and/or according to aimed targets (victims). For instance, there are web threats (distributed from web pages to any visitor), network attacks targeting to corporate infrastructure (e.g. servers), or malware targeting the client computers or mobile devices via their users. The occurrence and/or dissemination of network threats, primarily worms, can be efficiently monitored using honeypots. Similarly, even web-based attacks can be monitored by means of the application of a web application firewall (WAF) technology and/or via specific web-based honeypots. The most complicated situation occurs when a researcher needs to learn about threats attacking end-stations (e.g. PC workstations, laptops). Usually, companies providing security software have information on such threats, but it is not publicly available, or such information is published with a delay. Therefore, researchers have to find other sources of information and options to get the data from end-stations.

This paper describes a new project in this field called Colander. This is a newly developed tool for collecting data from ordinary end users (and namely their computers, including workstations and laptops). The gathered data is selected according to predefined rule sets. The rules are designed so that only data suspicious data in the user's network communications is gathered. Such suspicious data are often caused by malware or other more targeted cybernetic attacks.

There are many techniques and tools detecting many types of threats (i.e. various types of malware) attacking computers from the Internet and LANs. The long-term proven tools include antivirus and antimalware software. However, such software is beyond the scope of this research. This is primarily because of the fact that antivirus (and similar software) main focus is protection against malicious code. On the contrary, this research does not focus on any type of protection against threats. The objective of the research was just to analyse techniques for threat detection. Therefore, only tools for sole collecting data about detected threats (not including any preventive measures) are reviewed here. However, firewalls could seem to present an exception from this limitation. Nevertheless, only detecting abilities of firewalls are reviewed in this paper.

## 2. Network Threat Monitoring Fundamentals

There are various approaches to the detection of network threats. This task is complex because the threat "landscape" varies and evolves. Threats can be detected passively via monitoring of incoming threats. This is a prevalent approach (as confirmed e.g. in [2]) in most systems for both detection and prevention against various threats (e.g. antivirus software, antimalware etc.). Passive threat-detecting systems usually employ certain self-adaptability (e.g [3]). Recently, together with the quickly growing market of Internet of Things (IoT), new industrial systems connected to the Internet emerge that must be protected as well. This presents a new part of "threat surface" (covered in reviews [4] or [5]). In addition, there are many single-purpose software tools performing certain collection (direct or indirect) of data about threats and/or attacks coming through connection. However, the data collection is often not the primary purpose thereof. Such tools include firewalls, IDS/IPS, as well as other security projects. The potential of open-source tools for the

endpoint attack detection in the context of Windows-based workstations is analysed in the following sections.

## 2.1 Firewalls

Software firewall are classified into client device firewalls (personal firewalls), server firewalls, and network firewalls (independent appliances combining software and hardware protecting the whole network or its segment). Various commercial products prevail among personal firewalls, frequently combining various security features (like antivirus) into single software. Personal firewalls often gather data on detected threats that are subsequently processed in producer's private cloud [6] and recently even SDN concept is applied [7]. The users can usually decide either to allow or deny the firewall cloud extensions. When allowed, the firewall improves the promptness of the response to new threats.
Many server firewalls are based on iptables as proven packet filtering software. Iptables can also collect logs about selected network traffic; the logs can be easily shared in a specific community. For instance, data on connections targeted to port 22 (SSH protocol used for remote administration of unix systems) is often logged (see [8]). An example of popular tools using iptables is Fail2BAN (see [9]). Unlike personal and server firewalls running on a machine to be protected, network firewalls are intended to protect the whole private network or its segment, from potentially malicious communication (see [10]). Network firewalls are not subject of a detailed analysis here.

## 2.2 Intrusion Detection and Prevention Systems

Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) operation is based on the network traffic analysis. IDS/IPS principles are described e.g. in [11] where artificial intelligence application in detection is reviewed thoroughly, while [12] reviewed available solutions, both commercial and open-source, etc. Open-source IDS/IPS often use threat detection rules delivered by the product community but commercial supply of rule updates or the combination also exist. Here, the most frequently used systems are analysed from the point of view of the techniques of network traffic analysis and possibility for sharing gathered data about potential threats/attacks. Certain attempts exist to standardize of IDS events. The most notable is the IETF standardization of Intrusion Detection Message Exchange Format (IDEMF) in [14] and Intrusion Detection Exchange Protocol (IDXP) in [10]. These proposals were not implemented in practice, though, because of many open issues when sharing such sensitive data like attacks against network are to be shared. So far, sharing of data about attacks is rather uncommon.
IDS can detect threats based on the three following detection approaches:

**Signature based detection** – IDS has rules identifying threats. A rule identifies a specific threat based on one or more signatures it contains. A threat can be identified by multiple rules, since it can be identifiable by various sets of signatures. Rules usually pertain to individual packets. Typical signatures are a sender IP address, suspicious patterns in packet payload, ports used for non-standard purposes, or non-standard packet layout (e.g. a header field beyond standard definitions, or data in fields where it is usually not inserted, or failure to meet RFCs in general).

**Script based detection** – scripts written in a language can potentially specify detection parameters more accurately than signatures. In certain cases (e.g. a DDoS attack) a threat can only be identified by multiple subsequent packets, necessitating use of scripts.

**Anomaly based detection** – builds on the analysis of an "ordinary" network traffic and, subsequently, the IDS looks for patterns of packet behaviour that are unusual in the "ordinary" traffic. A thorough overview of anomaly-based IDS is presented in [15]. This approach requires a period when the IDS analyses a common traffic and derives the data for detection of anomalies (learning period). After this period, however, the IDS can detect even attacks that have been unknown as of yet (i.e. signatures are not available for them yet like "zero-day" attacks).

An example of three common intrusion detection systems follows:
**Snort**[1] seems to be the most frequently used open source IDS (about half a million users). It uses signature based detection as described in [16]. Signatures and rules are created by the user community with subsequent processing of the project staff. Snort is distributed as GNUv2-licensed open-source. The main advantage of Snort is its cross-platform nature – it can be implemented having compiled from source codes as well from binary packages available both for various Linux distributions and FreeBSD and for Windows as well. There is an issue with Snort, however, a limited optimisation causing problems with running it especially on low-capacity devices. This is due to the absence of multithreading support expected to come in version 3.0, which is now in the alpha testing phase. Rules for evaluation of the network traffic are essential for Snort operation. Besides the ruleset provided by Snort itself, there is also a ruleset called Community ruleset (Talos) is freely available and it is utilised in other systems, too, e.g. Suricata.
Rules are distributed as a text file where every line represents a single rule for a specific threat. Every rule is comprised of a header and its detection conditions. Each rule has a single header and can have multiple conditions, e.g. the protocol type, the source and destination ports, the packet direction (in or out), parts of the payload to match, etc.

Complete Snort rule example:
# alert tcp $HOME_NET 2589 -> $EXTERNAL_NET any (msg:"MALWARE-BACKDOOR - Dagger_1.4.0"; flow:to_client,established; content:"2|00 00 00 06 00 00 00|Drives|24 00|"; depth:16; metadata:ruleset community; classtype:misc-activity; sid:105; rev:14;)

New rules are created primarily by the community and shared dynamically. In addition, every user can create their own rules for their purposes, and such a rule can be provided to Snort that can integrate the rule into the community ruleset. The rules and their format are used by the Colander IDS, as they were the most wide spread and available rule set.
**Suricata**[2] software is backed by Open Information Security Foundation (OISF)[3]. Its first beta version was released in 2009 while the first stable release in July 2010. The Suricata paradigm as well as technical background is similar to Snort. Nevertheless, it brings some new elements. like numerous extensions, both by the producer and third-party. Suricata is

[1] Available at: https://snort.org
[2] Available at https://suricata-ids.org
[3] Details are available at https://oisf.net

focused primarily on Linux-based servers in commercial segment but the support for FreeBSD and Windows is still available. The snort performance on the latter platforms is lower, nonetheless (see [17]). The main advantage of Suricata is its high performance and the ability to analyse the traffic at high speeds in multiple Gbps. This is enabled by multithreaded architecture and GPU acceleration [17]. Suricata is designed primarily for high-performance servers with multiple CPUs and massive multithreading as well for a high-speed communication. It allows variable logging, for instance of HTTP requests, TLS certificates, or extracting files from TCP traffic flow. More complex types of detection that are beyond the description in the form of rule syntax are possible using Lua scripting language. There is another filtering layer allowed thanks to the support for IP address reputation assessment using public reputation lists listing a huge number of IP addresses and their reputation evaluations. When the automatic logging or communication rejection from untrusted IP addresses is applied, lots of computational resources can be saved.

Suricata makes use of two main sources for detection rules, Talos/Snort ruleset[4] with the rules described in the section Snort (2.3.1) above, and Emerging Threats[5], which are similar to Talos community rules, except for the fact that from a different community called Emerging Threats.
This necessarily means that duplicate rules emerge when both rulesets are used.

**Bro** system was originally designed by Vern Paxson[6] who has been steering the project still in cooperation with researchers and developers at the International Computer Science Institute in Berkeley and in the National Center for Supercomputing Applications[7]. Bro applies a signature-based approach too, but its main emphasis is given to an anomaly-based detection. Bro was designed as a research tool not intended for the use in a corporate environment, so its basic approach and architecture is slightly different from the above Snort and Suricata. The main difference consists in the possible use of alternative cluster architecture called Bro Cluster[8]. It supports Linux, FreeBSD and MacOS, but it does not support Windows. Bro provides extensive log files containing not only a simple record for each connection but e.g. all details about each HTTP session with all requested URLs, key headers, used MIME types, and server responses, and SSL certificates, or the significant contents of each SMTP connection etc. Fundamentally, Bro represents a platform for the network traffic analysis where IDS is just one of many functions available. The most important Bro's component is its domain-specific scripting language able to express arbitrary analytic tasks. Thanks to it, Bro can be considered as a "domain-specific Python". It has a huge range of functionalities even with its standard library while their extending using own-made or community libraries is possible. Other Bro capabilities include file extraction directly from the http session, malware detection using the interface of external registers, notification about vulnerable software version in the network, identification of frequently used web applications, SSH brute-force attack detection, and many others. There is an important option from the implementation point of view consisting in forming so-called Bro Cluster. This is a means for balancing the load of Bro between multiple nodes (any

4 Available at https://www.snort.org/downloads/#rule-downloads

5 Available at http://doc.emergingthreats.net/bin/view/Main/AllRulesets

6 Details can be found at https://www.bro.org/sphinx/intro/index.html

7 See also http://www.ncsa.illinois.edu

8 Detailed description is available at https://www.bro.org/sphinx/cluster/index.html

computers in the network). It consists of the central manager coordinating and synchronising processes for any number of workstations that analyse the traffic that passes through them. Bro is distributed in a form of multiple modules and frameworks that can be used separately or combined as desired. Among such modules, the following worth noticing: File Analysis, GeoLocation, Intelligence Framework, Signature Framework, NetControl Framework, and many others.
Bro uses two types of detection rules when used as IDS. Namely:

- Talos/ Snort ruleset[6] - rules described in the section Snort above.
- Scripts of Bro scripting language[9] - Bro scripting language has the expressivity comparable to Python or Pearl, i.e. it can define any conditions for the traffic analysis that go beyond the description by signature-based rules.

**Comparison of analysed network IDS** shows that each of the IDS mentioned above takes a different approach. While Snort's main focus is to spread among the IT security community and enthusiasts as much as possible, trying to persuade them by their low resource requirements and availability for almost any operating system, its community is the largest one. However, its long history demonstrates, among others, in its outdated technological basis, primarily in the absence of multithreading and detection of anomalies.
Suricata originated as Snort's direct competitor offering improved function range, consisting primarily in the anomaly detection function. Nevertheless, its improved functions demonstrated in higher hardware and system requirements as well. Together with gradual widening of available functions, the main focus of the project was changing. At present, Suricata is primarily focused on high-performance servers with multiple powerful CPUs containing many cores.
Bro was designed from its very origin as a research platform. The most significant difference from Sort and Suricata consists in the fact that it can be implemented in a LAN as a whole where individual nodes communicate and cooperate. Regarding its high complexity, Bro is more suitable for corporations rather for enthusiasts. The **Table 1** summarises the main aspects of Snort, Suricata and Bro.

**Table 1.** Comparison of main aspects of analysed IDS systems.

| **Parameter** | **Snort** | **Suricata** | **Bro** |
|---|---|---|---|
| Signature rules | Snort/Talos | Snort/Talos and Emerging Threats | Bro signature language |
| Anomaly detection | No | Yes, by Lua scripting language | Yes, by Bro scripting language |
| Multithreading | No | Yes | Yes |
| Installation | Manual | Package based or manual | Package based, manual for extensions |
| Documentation | Expansive online manual on the project web pages, extensive community support | Fledgling online manual, relatively small community, limited Lua documentation | Expansive online manual, detailed installation manual |
| IPv6 support | Supported after installing the IPv6 module | Fully supported | Fully supported |

9 For details see https://www.bro.org/sphinx/scripting/

| | | | |
|---|---|---|---|
| Operating systems | Linux, FreeBSD, Mac OS, Windows | Linux, FreeBSD, Mac OS, limited Windows | Linux, FreeBSD, Mac OS |
| Scripting language | No | Lua scripting language | Bro scripting language |
| Target hardware | Servers, common workstations, network nodes | High end servers | Servers, common workstations, network nodes, has variable workload |
| Last stable release | 2.9.11.1, released on 04.01.2018 | 3.2.2, released on 07.06.2017 | 2.5.4, released on 30.05.2018 |

The above software tools have the main aim in protecting a specific workstation or a server. However, the presented Colander project primarily demands a tool that is able to detect the network traffic passively while informing the central point about potential threats and the user's hassle is as little as possible. This is not the case of these IDS systems. The reason why their analysis is presented here is that certain parts of their functions, namely Snort's signature detection, is required for our project.

### 2.3 Similar threat data gathering projects

The following projects with similar goals to Colander provided inspiration for its inception, as they both collect data from home PCs. There are various other solutions that provided additional inspiration, such as the specialized rule-based multiagent IDS described in [19] or the botnet netflow-based detector based on machine learning pattern identification demonstrated in [20], but their impact on our system is quite limited.

**Turris project**[10] was established by the association CZ-NIC. The project is devoted to the design, production and improvement of special home-network edge-routers for networks with advanced capabilities related to the collection of traffic data aimed at subsequent improvement of the security level of the network where the router operates. The main difference between Turris and an ordinary IDS lies in the fact that Turris analyses only packet headers while the packet payload, the data, is intentionally ignored in order to protect user privacy. Turris router operates as an ordinary integrated home router. If a potential threat is identified, the record is sent to the project server where it is assessed and compared to other similar threats. If it is confirmed that an attack is detected, a preventive measure protecting against it will be incorporated in a subsequent update, thus protecting all the users.

**European Network of Affined Honeypots** (NoAH) [18] is an already discontinued project[11], but it served as an inspiration for Colander. The project's main aims were collecting and analysing attacks from the Internet and forming an infrastructure for collecting the captured data from such attacks and storing it to be used in further research by CSIRT and CERT teams.

Honey@home represents a client implementation of NOAH for ordinary home PCs. It operated by emulating services on network ports not being used by the any other program on the PC. Any traffic through these ports was forwarded to project's servers for further analysis. No selection or analysis was performed on the home PCs.

[10] Details about the project are available at https://www.turris.cz/en/

[11] Details about the project are available at https://www.fp6-noah.org and http://www.honeyathome.org

**Distributed intrusion detection with intelligent network interfaces for future networks** [26], a paper that outlines an idea and presents the results of a prototype distributed IDS system. The presented IDS is layered, with the first layer conducting preprocessing of network traffic on network nodes, such as routers, and redirecting it to one of several servers running second layer of the IDS based on the results of the preprocessing. Every server of the second layer is running the IDS with a different set of rules smaller than the overall set used by all servers together. This is done so that the redirected traffic is only checked against relevant rules, saving computing time. In contrast, Colander is meant for user PCs, not servers, only operates at the end points of networks, and uses small set of relevant rules.

## 3. Problem formulation

The above analysis demonstrates that certain projects partially meet the objectives set up for Colander but none fulfil them completely. Therefore, a decision was made to apply certain approaches (e.g. rule sets) from them but to develop completely new tool to meet all the requirements. The most similar tool is Honey©home client by NoAH. The similarity lies mainly in it being a software solution for collecting data about threats from home PCs, and in splitting into functional-analogous client part for data gathering and analytical server part functions. Honey©home, however, can only gather data about communication on ports not commonly used by the given PC, while Colander's goal is to analyse all of the network communication, regardless of the port used. Lastly, Honey©home does not analyse the communication on the client, while Colander does.
Attacks against client computers are the object of interest of the Turris project. It analyses the communication on the client side, as Colander does, but it only analyses the packet headers. Turris is a separate appliance serving as a router, not just software.
Among the IDS mentioned above, Snort was the main inspiration for this project, from the point of view of functionality. Due to the absence of any specific focus, it provides rules with the widest possible scope that are much more useful for Colander project comparing to Emerging Threats rules having their specific focus niches.
Based on the analysisit was decided to use modified Talos/Snort rules in Colander.Also, the Suricata's focus to maximise the utilisation of the highest possible number of cores was the motivation for the maximum possible application of multithreading due to the modern proliferation of multi-core processors even in low-end computers. There was another inspiring aspect in Honey©home where it was demonstrated that a properly designed graphic user interface could increase the users‘ motivation towards using of such a tool. Therefore, Colander is equipped with a basic graphic interface as well.

### 3.1 Colander Project Philosophy

Based on the analysis, a conclusion was made that virtually all existing (alive and discontinued) research NIDS tools and projects aimed at analysing network threats are oriented to run on networked nodes, servers, or even personal computers of enthusiasts having certain entry level of knowledge. On the other hand, there are commercial firewalls and antivirus/antimalware software not requiring much knowledge to operate but they do not publish their data, or they are late, or incomplete. Due to this fact, their research results (even if it is performed by commercial producers) are not applicable in academic research.

Therefore, there is a “blind spot“ in the field of cyberattack research in a missing collection of data required for subsequent research. This is primarily represented by residential (home) networks where the majority of ordinary users run Windows. Despite the Turris project mentioned above focuses on attack data collection from home networks, this solution does not aim at users‘ computers and their systems.
In general, there could be two main reasons why this “blind spot“ has not been filled yet. The first is a lack of users’ motivation. Why a user should do something that will not bring any immediate benefit to them? A contribution to research is a vague and non-motivation idea. Also, the inprovement of the user’s own IT security does not work either because of low awareness on IT risks among users. Another reason lies in demands on users. Standard IDSs are not designed for the use by ordinary users and their operation is beyond the capabilities of such users,. Moreover, common IDSs usually require certain configuration (e.g. selection of rules) that affect the IDS operation significantly. And the collected data should be analysed or at least shared with the community, which could be another obstacle for an unqualified user. Data merely accumulated and not shared with experts for analysis is useless.
In order to attract ordinary users to use Colander, both of the above-mentioned issues were addressed properly. Therefore Colander tries to provide a maximum ease of use and demands on the user were reduced as much as possible. The only thing that the user have to do is to run the installer with appropriate privileges and let the installation complete. Then they can forget about the program forever. In case the users are interested in controlling the program, a simple and easy to use graphic user interface is available where they can control the program slightly, and statistics are displayed. It is expected nevertheless that for most users, this program is just represented by a new, never used icon in the hidden icons bar.
The motivation of a potential user seems to be a much more difficult issue. As mentioned above, no immediate incentive means no motivation in most cases. Therefore, for now, the dissemination is mainly among people with a relation to the authors, for whom the motivation consists of helping the IT security improvement.

## 4. Technical Design

Several issues had to be addressed in the Colander program design. Among them, the lowest achievable demands both for users and system resources were the most important. Other issues included the method of communication of the client with the central server, and mainly detection of potential threats identified in the network traffic destined to the client.

### 4.1 Colander Architecture

Colander is built on the client-server architecture where the data collection is performed on the client and the server part’s main task is to store the captured data and control client’s behaviour. The Colander architecture is shown in **Fig. 1**.
**Client side.** Regarding the client target platform that is Windows, .NET framework seemed to be an obvious choice for programming the client’s code, where C# language was chosen. Due to the fact that C# language is supported in Microsoft .NET as a primary language for application programming for Windows, it provides virtually all required tools including Windows service programming and configuration as well as GUI design. More specifically, .NET version 4.0 was chosen, which allows the code to be executed on any

version of Windows Vista and newer without the need for installing .NET framework updates in case of Windows 8 and 10. The packet capturing is performed using a well-known and proven solution WinPcap[12].

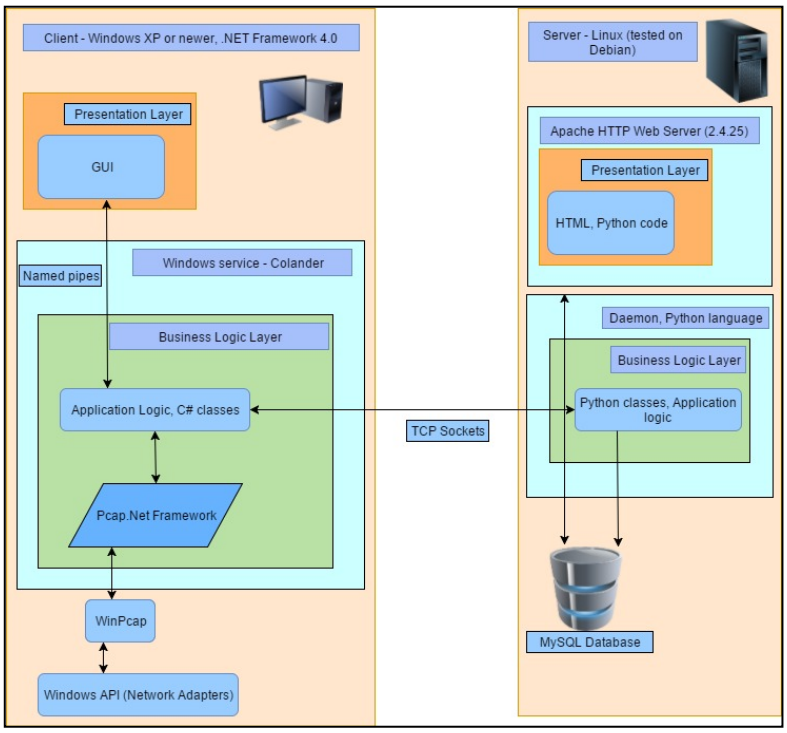



**Fig. 1.** Colander system architecture.

WinPcap serves as an interface between Windows API providing a low-level access to network interfaces. It consists of a driver that broadens the OS's functions in providing the low-level access to the network adapter, and a library allowing to access and control the driver. In order to make the operation of WinPcap more efficient, .NET wrapper for WinPcap designed for C# - Pcap.Net[13] was used. It allows direct calling of most WinPcap functions from C# language. The Colander client program is installed as a Windows service. A GUI is included in the distributed package serving for the client control.

**Server side.** The server side is intended for storing the captured threats and their analysis. It was developed for Linux OS (Debian distribution is preferred). The Linux-based solution was chosen compared to a Windows-based server side because of Linux's significantly lower hardware requirements resulting in lower cost. The server side's main task is to wait for connection from clients and after being connected to store their captured packets to a MySQL database. Those packets were captured on a client because it evaluated the data as containing a potential threat. An integral part of the server side is the Apache server providing a hosting service for web pages designed in the Flask Python framework

**Model of the database.** The database contains only two tables as they are shown in **Fig. 2**. The Packet table was designed to store captured suspicious packets. The attributes are rather

12 Available at https://www.winpcap.org

13 Available at https://github.com/PcapDotNet/Pcap.Net

self explanatory.

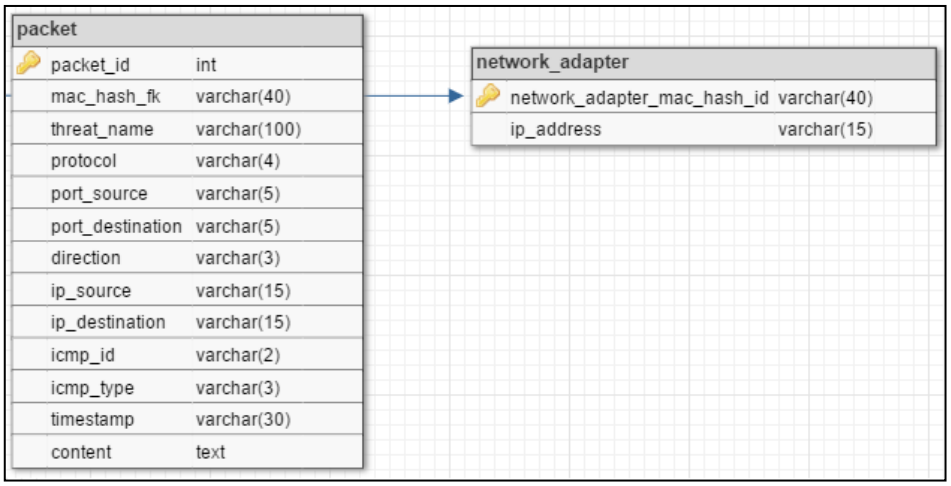



**Fig. 2.** Database Model Diagram.

The other table called Network_adapter serves for a unique identification of network adapters that are involved in the packet capture. The Colander client allows capturing from multiple network adapters, e.g. a laptop can be connected to the Internet via Ethernet cabling at home and through Wi-fi otherwise. The table contains a SHA-1 hashed MAC address of the interface in Network_adapter_mac_hash_id attribute. It serves as a primary key making use of the uniqueness of MAC addresses. The table contains a single record for each single network adapter. The table contains the IP address assigned to the interface but its storing is subject to the user consent and the program can operate without the local IP address storing.

**Rules for detection of potential threats in traffic.** As mentioned above, Colander makes use Talos/Snort rules for signature based detection. Before applying, the rules are modified, mainly by removing rules not intended for Windows, such as Linux specific rules. Also rules dealing with specific types of threats that are unlikely to be encountered on a home PC are removed, for example DDoS attack rules.

However, the modularity of the system will allow ruleset modification in the future (based on an analyst decision), consisting either in adding new rules, or in removing the rules that are no longer considered necessary. Following are some of the most common options of the Talos/Snort rules for illustration:

**Action: alert** – "alert" is the keyword of the most common action, meaning that this rule generates a notification. Only rules with this action are being used, although they are already a vast majority.

**Protocols** – protocols, such as TCP, UDP, or ICMP.

**Port numbers** – source and destination ports used for UDP and TCP protocols.

**Direction** – direction of communication is often a key characteristic in assessing threats.

**Msg** – threat name used for identification.

**Content** – the packet payload that indicates the threat. A single rule can contain multiple "content" elements.

**Pcre** – regular expression for matching the contents of the payload. A single rule can contain multiple "pcre" elements.

**User interface.** Having the Colander client program installed, no user interaction is required for the service to become operational. The service is designed specifically so that the user can install it and forget about it. GUI, as shown in **Fig. 3** serves just as an added value for such users who are interested in having more details. Its use is purely optional.

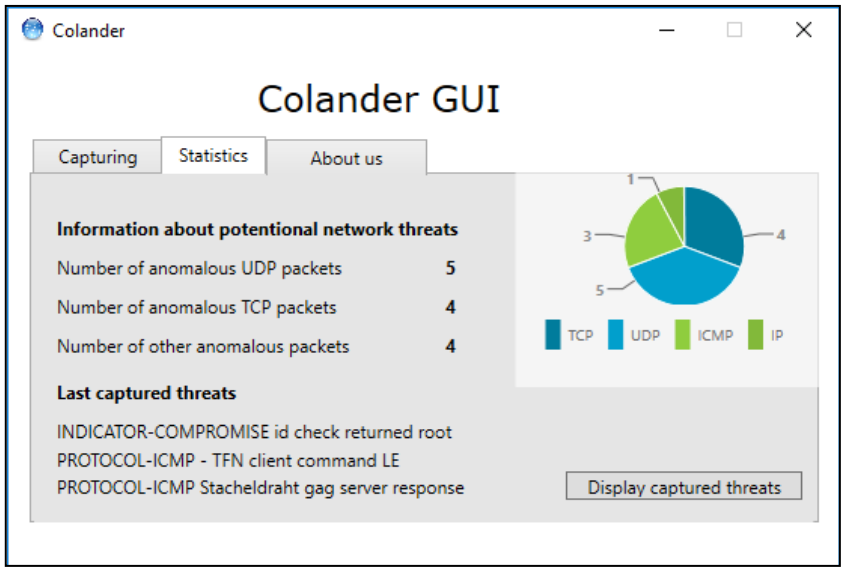



**Fig. 3.** Colander – Basic statistics.

### 4.2 Colander internal design

Colander service and GUI were coded using Microsoft Visual Studio 2017. The source code was written in C# language. The serve side was coded in Python 3 language using the development tool JetBrains Pycharm and web framework Flask.
After being started, the Colander service runs a short 2 minute test on all available network interfaces to determine which ones are active. When that is determined, it starts its scanning cycle on active interfaces. The scanning cycle is composed of 10 minutes of scanning the traffic on the given interface, after which it stops to release the RAM it was using and to send the captured threats to the server, after which the cycle restarts. In multi-threaded version, a thread is created for each available core, and each thread is assigned a packed for analysis as they are received.

## 5. System Testing and Results

The program has been tested from the point of view of its performance, i.e. the ability to capture several threats represented by a real malware code that was executed in a laboratory environment with subsequent spreading of the malicious code among other users. From approx. 10 malware specimens, only three were active as detailed in Section 5.2.

### 5.1 Performance benchmark testing of the Colander service

Regarding the fact that the Colander client runs on common client computers (e.g. workstations or laptops), the minimisation of system resource demands was one of the primary goals as stated above. Otherwise, a significant slowing down of the system could results in lower user motivation to keep running the client program.
Tests were focused on the CPU Load expressed as a percentage of the total CPU load at the specific moment. This expresses the CPU load in relation to the other processes. This is beneficial when measurements from different CPUs are to be compared whent he absolute metric could be difficult to compare. This means that when the total CPU load in a specific moment was e.g. 30% and the the load of Colander was 5%, the load of Colander was 5% of the 30% of the total CPU potential, which would be 1.5% in this case.
Packet loss, the percentage of packets that passed the specific network interface without being analysed, is also considered in the tests.
The tests consisted in a TCP stream and UDP flow download at various average transmission speeds. The tests were performed on three different computers differing in CPUs. The CPUs in the test were the following:

- Intel Core i5-6400 @ 2.7GHz, 4 cores (denoted “i5”),
- Intel Core i7-26302QM @ 2GHz, 4 cores with Hyper Threading into 8 threads (denoted “i7”), and
- Intel Atom Z520 @ 1.33 GHz, single core (denoted “Atom”).

The transmission speeds used in the tests were 2 Mbps, 3.2 Mbps, 6.4 Mbps and 16 Mbps. Tests were made using “Windows Performance Recorder“ tool that can record the utilization of CPU by individual processes. The duration of every single test was 5 minutes.
There is an important difference in resource requirements for processing between TCP and UDP packets because of a significantly different number of rules. While the number of UDP rules is just 355, the number of TCP rules exceeds 2500 (2546 in fact). It is obvious that the higher the number of rules, the longer CPU time is required for single packet evaluation. Therefore, the number of rules limits the maximum number of packets that can be evaluated in a specific time.
The following table describe a set of selected representative tests.

**Table 2.** System testing results

| Test | Configuration | Test result |
|---|---|---|
| Test 1: PC was in an idle state (i.e. only testing download and common automatic background processes were running in the system). | CPU i5, Protocol UDP, Transmission speed 2 Mbps, Program version was single-thread, Average CPU load of all processes was 20%. | CPU load of 5% average with no packet loss detected. |
| Test 2: PC was in an idle state like the one in the previous test. | CPU i7, Protocol UDP, Transmission speed 2 Mbps, Program version was single-thread, Average CPU load of all processes was 20%. | CPU load of 5% average with no packet loss detected, as expected due to the previous test. |
| Test 3: The CPU load was high in order to demonstrate Colander’s acceptable CPU load even under heavy overall CPU load. | CPU Atom, Protocol UDP, Transmission speed 2 Mbps, Program version was single-thread, Average CPU load of all processes was 90%. | Acceptable CPU load at average of 10% with still no packet loss even on the slowest CPU. |

| | | |
|---|---|---|
| Test 4: Transmission speed was heightened compared to the test No.1. | CPU i5, Protocol UDP, Transmission speed 3.2 Mbps, Program version was single-thread, Average CPU load of all processes was 20%. | Average CPU load remained at 5% with no packet loss detected. 5% seems to be the minimum CPU load for this configuration regardless of the transmission speed. |
| Test 5: Transmission speed was heightened with normal background CPU load for the given CPU. | CPU Atom, Protocol UDP, Transmission speed 3.2 Mbps, Program version was single-thread, Average CPU load of all processes was 50%. | Average CPU load rose to an average of 10% with no packet loss detected, an acceptable and expected increase. |
| Test 6: Transmission speed was doubled from the previous test in order to go over the maximum transmission speed at which this CPU can analyse the traffic without packet loss. | CPU Atom, Protocol UDP, Transmission speed 6.4 Mbps, Program version was single-thread, Average CPU load of all processes was 60%. | Average CPU load rose to an average of 17% with packet loss of 4%. The maximum UDP transmission speed of the Atom-based computer is approx. 6.16 Mbps. |
| Test 7: Transmission speed was much higher in order to demonstrate the capacity limit of Colander at the specific CPU. | CPU Atom, Protocol UDP, Transmission speed 16 Mbps, Program version was single-thread, Average CPU load of all processes was 60%. | The CPU load held at the reachable maximum of 17%. Packet loss increased to 63%. An example of goin over the maximum speed of a CPU, when packets exceeding the limit are discarded. |
| Test 8: A comparative test to the test No.7, with the same speed but a faster CPU. | CPU i5, Protocol UDP, Transmission speed 6.4 Mbps, Program version was single-thread, Average CPU load of all processes was 20% | Average CPU load rose to an average of 6% with no packet loss detected, showing that the clock speed of the CPU matters significantly. |
| Test 9: A comparative test to the test No.8, with the same speed but a higher load to demonstrate that the efficiency of Colander is unaffected by the high overall CPU load. | CPU Intel i5, Protocol UDP, Transmission speed 6.4 Mbps, Program version was single-thread, Average CPU load of all processes was 70%. | CPU load of Colander relative to the overall load was 2%, demonstrating that Colander's CPU load is low under heavy overall load as well as under light one. |
| Test 10: Test demonstrating the difference between tests using TCP and UDP protocols. It is a comparison to the test No.8, but using TCP. | CPU i5, Protocol TCP, Transmission speed 6.4 Mbps, Program version was single-thread, Average CPU load of all processes was 15%. | There is no difference in CPU loads at the same speed, but there is a 12% packet loss. The maximum speed limit for i5 CPU when capturing TCP, approx. 3.5 Mbps. |
| Test 11: The single-thread service version limit was demonstrated again at higher transmission speed. | CPU i5, Protocol TCP, Transmission speed 16 Mbps, Program version was single-thread, Average CPU load of all processes was 20%. | The single-thread service is efficient up to a specific packet transmission speed. After it is exceeded, a significant packet loss occurs, in this case even up to 66%. |
| Test 12: A test of the single threaded Atom CPU using the multi-threaded version of Colander. | CPU Atom, Protocol UDP, Transmission speed 6.4 Mbps, Program version was multi-thread, Average CPU load of all processes 90%. | The multi-thread version is completely inadvisable to be used on single-thread CPU. The packet loss was unacceptably high, a programming issue to be |

| | | |
|---|---|---|
| | | fixed in later versions. The packet loss was up to 83%. |
| Test 13: A test demonstrating the usefulness of the multi-threaded version of Colander when used on an appropriate CPU. | CPU i7, Protocol UDP, Transmission speed 16 Mbps, Program version was multi-thread,<br>Average CPU load of all processes 90%. | The CPU i7 with eight threads is optimal here as the zero packet loss shows. CPU load of 30% under heavy load is rather high, but the efficiency is arguably worth it, and it is decreasing with further thread optimisation. The multi-thread version has a need for additional RAM due to the packet buffer. In this test, the RAM consumption reached 400 MB in 5 minutes. |

**Benchmark test summary.** The tests partly described-above demonstrated the advantages and disadvantages of both versions of the service. The single-thread version has proven as a safer selection in general, always using less computer resources, at the expense of the high packet loss rate in the case of high number of packets transferred. It definitely is a more suitable choice for users who do not like their computer capacity to be affected. It is also the only choice for computers equipped with a single-core CPU.
The multi-thread version has proven as a riskier choice that affects the computer performance as the expense of the packet loss minimising. Obviously, this can be a good choice primarily for computers with a multiple-core CPU.
At this moment, the single-thread version seems more convenient for potential users. However, subsequent development of the application will focus on the multi-thread version optimisation and improvement, with the single-thread version being more of a proof of concept, or a prototype.

## 5.2 Performance benchmark testing of Snort

Examples of performance of Snort under the same conditions and configuration as some of the Colander tests are included for comparison. Snort's setting have been modified to eliminate extra tasks in comparison to Colander, such as extensive logging, and to perform virtually the same tasks as Colander. Colander tests No.1, No.9 and No.11 were chosen as representative.
The comparative tests to the tests No.1 and No.9 had the same average CPU loads of 5% and 2% respectively, both with no packet loss.
The comparative test to Colander test No.11 shows an average CPU use of about 7%, however with considerably lower packet loss of only 9% compared to Colander's 66%. The result is better likely due to Snort's use of more advanced pre-processing of rules, where it decides to only use only rules it estimates to be relevant. This result only confirms the necessity of further development and optimisation of the multi-thread version, as shown when comparing the result with test no. 13, where it has no packet loss, but quite high CPU usage.

### 5.3 Capturing the communication of an infected system

In addition to the performance tests described above, the Colander program was tested by analysing three different cyber threats captured as described below in detail.

**Trojan Banker FTC**. The first threat captured in the tests was malware identified as Trojan Banker FTC[14]. The malware sample that was detected is described in detail in [22]. The rule identification number or SID where the match was found, is 25829. The matching packet is shown in the capture portion in **Fig. 5** while the contents that is defined in the rule as: content:"/listas/out/si.php"; content:"HTTP/1.0|0D 0A|" is clearly visible in the lower part of **Fig. 6**.

| | | | |
|---|---|---|---|
| 192.168.137.73 | 192.168.137.1 | DNS | Standard query 0x456b A dc.services.visualstudio.com |
| 192.168.137.73 | 192.168.137.1 | DNS | Standard query 0x8f87 A mobile.pipe.aria.microsoft.com |
| 192.168.137.73 | 67.228.96.203 | TCP | [TCP Retransmission] 2589 → 25 [ACK] Seq=1 Ack=1 Win=100 Len=122 |
| 192.168.137.73 | 192.168.137.1 | DNS | Standard query 0x6e1a A dns.msftncsi.com |
| 192.168.137.73 | 104.96.95.202 | TCP | 53289 → 443 [FIN, ACK] Seq=1 Ack=1 Win=258 Len=0 |

**Fig. 4.** Selected portion of communication on the client. The Black highlighted line contents the packet that was found to match to the sample in the rule.

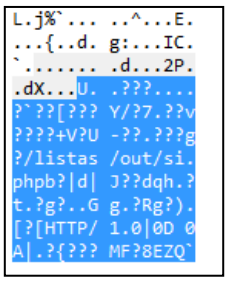



**Fig. 5.** Data part (payload) of a packet matching the rule.

**NETBIOS DCERPC.** The second thread active and captured in the tests was identified as the "NETBIOS DCERPC NCACN-IP-TCP winreg InitiateSystemShutdown" attack and started to demonstrate after the activation of the sample identified as VirutNetwork having SHA-1 hash of "e6c0ac72ac520f7d7def04d5f59edb58e2693246ce10c70754baa5bbb5005208". The threat was captured by the other computer in the local network where the service was active just for the cases when an attack occurs in the local area network.

Regarding the contents of the communication payload where only specific SMB - Server message block messages are detected that were exchanged between computers in the local network, it is unlikely that the direct communication of the malware with its C&C server but the assessment of the activity as suspicious as a part of SMB protocol. The thread Sid is: 2942. The matching packet is shown in the capture portion in **Fig. 7**.

---

[14] The malware SHA-1 hash value is e449da37ed87e7643a3d177380927aad49158f4487d63bf5924a97e0f6d83514

| | | | |
|---|---|---|---|
| 192.168.137.73 | 192.168.137.1 | NBSS | Session request, to DESKTOP-MT66MBE<20> from DESKTOP-DCV9888<00> |
| 192.168.137.1 | 192.168.137.73 | NBSS | Positive session response |
| 192.168.137.73 | 192.168.137.1 | SMB | Negotiate Protocol Request |
| 192.168.137.1 | 192.168.137.73 | SMB | Negotiate Protocol Response |
| 192.168.137.73 | 192.168.137.1 | SMB | Session Setup AndX Request, NTLMSSP_NEGOTIATE |
| 192.168.137.1 | 192.168.137.73 | SMB | Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MOR |
| 192.168.137.73 | 192.168.137.1 | SMB | Session Setup AndX Request, NTLMSSP_AUTH, User: \ |
| 192.168.137.1 | 192.168.137.73 | SMB | Session Setup AndX Response |
| 192.168.137.73 | 192.168.137.1 | SMB | Tree Connect AndX Request, Path: \\DESKTOP-MT66MBE\IPC$ |

**Fig. 6.** Selected portion of communication on the client. The green highlighted line contents the packet that was found to match to the sample in the rule.

**WannaCry.** During the period when the third testing phase was performed, there was a massive spread of the malware known as WannaCry as detailed in McAfee website [23], which made use of the vulnerability of SMB protocol identified as ETERNALBLUE MS17-010 described on Microsoft website [24]. The captured specimen had a SHA-1 hash: cd79b536868efb8b2edd2db4e4100f0bd2f69e28. The program was thus tested how it can deal with capturing and identification of this threat. The payload that was identified based on the use of signatures with SID: 2024220 (SMB Echo Request) and SID2024218 (SMB Echo Response) is shown in **Fig. 8**.

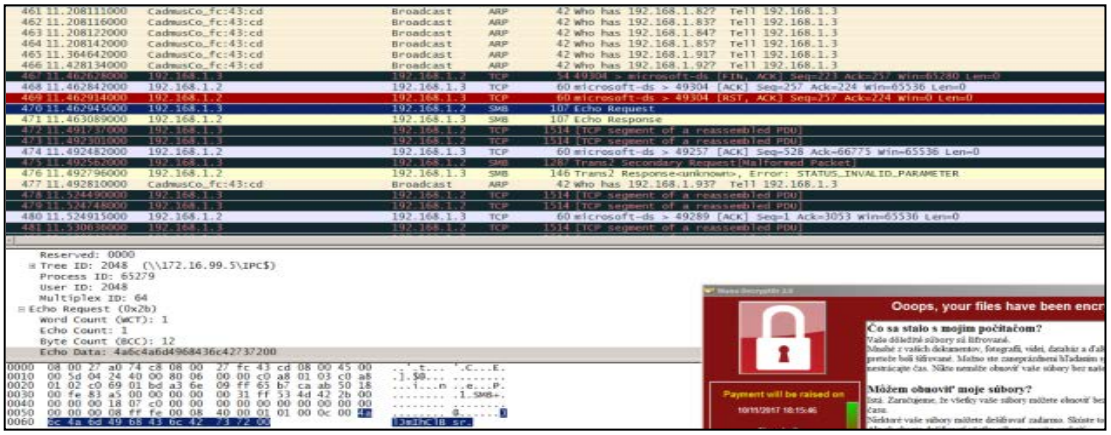



**Fig. 7.** WannaCry malware as captured by Colander . Print Screen of Wireshark with highlighted payload matching one of the WannCry rules, and the WannaCry window asking for ransom

The output on the victim side looked as follows:
Threat name: ET EXPLOIT Possible ETERNALBLUE MS17-010 Echo Request (set)
Source Port: 449275
Destination Port: 445
Direction: incoming
Source IP: 192.168.1.3
Destination IP:192.168.1.2
Content: 1ÿSMB+Àÿþ@JlJmIhClBsr
Threat name: ET EXPLOIT Possible ETERNALBLUE MS17-010 Echo Response
Source Port: 445
Destination Port: 449275
Direction: outgoing
Source IP: 192.168.1.2
Destination IP:192.168.1.3
Content:1ÿSMB+ˆÀÿþ@JlJmIhClBsr
All the above tests described in this section using the three known vulnerabilities have demonstrated the functionality of Colander program because they were recorded correctly.

### 5.4 Pilot implementation among a set of users

The pilot implementation had three test goals. To test the system as a whole, the communication between clients and the server, storing captured threats in the database, simultaneous control of multiple clients. Second goal was obtaining a feedback from users, what the usage of the application brought them, whether the installation was easy, how much the GUI was understandable, etc. The last goal was capturing real threats. It should be observed whether false positives happened and/or anything interesting was captured.

Regarding the gradual development and code improvement, three test phases were performed. Primarily, the last phase provides relevant results. This phase took more than two months, i.e. from March 4 till May 12, 2017. In this phase, the service already had a complete capability to capture threats. In all testing phases, the single-thread version of the program was used.
The testing was performed on 12 clients with various activity. The users selected for the test were those with a minimal or no consideration with their computer security so that the test was as close to the real implementation as possible.
Users appreciated easy use of the program consisting just in Colander installation together with WinPcap. The GUI was evaluated as clear and well-arranged. Only two users complained of the absence of any network adapters in the adapter list. It was shown that both of them used USB WiFi adapters that WinPcap could not recognise. Therefore, this problem could not be eliminated directly by changing the Colander client code. None of the users observed an increased CPU load or a feeling of the computer power decrease. The Colander implementation has proven that the operation of the system (i.e. threat capturing, sending and storing) is proper as designed.
**Captured threats.** During the service testing phase, the number of captured threats was decreasing gradually, thus eliminating primarily false positive detections due to imperfections of the detection algorithm. Therefore, only threats detected in the third pilot phase were taken into consideration.
The following threats belong among those that were the most frequently captured in the testing phase or were most interesting:

**INDICATOR-SCAN ipEye SYN scan, synscan portscan** – capturing of these two threats was announced by each client running under Windows 10. After more detailed local testing, it was observed that in this case Windows 10 updates were captured that meet the rules. This case just emphasises the necessity to make a proper manual selection of rules for a specific situation.

**PROTOCOL-ICMP PING Microsoft Windows** – This is an example of a threat that may not be a real threat. In this case, a simple ICMP echo request is captured that can be sent in a legitimate way, e.g. for a network issue diagnosing. It can be considered as a real threat only when it is used by an attacker in order to verify the activity of the computer as an attack target.

**OS-WINDOWS SMB startup folder access** – this is an attempt to access to system files via SMB protocol. It can be a real attack that could happen if the user allowed the file or printer sharing including system files.

**FILE-IDENTIFY Microsoft Windows Media download detected** – the rule that triggered the threat is intended primarily for ingress points of corporate networks. Here, downloading of certain multimedia files is considered as a threat because it is often restricted by corporate regulations. This could be considered as a threat because such files are able to disseminate malicious codes like Trojans under specific circumstances.

**MALWARE-BACKDOOR MISC Linux rootkit attempt** – an example of apparently false positive detection when the communication met the conditions of the rule for evaluation as a Trojan login into the Linux system. Again, this emphasises the necessity of proper rule selection.

**OS-WINDOWS SMB Session Setup NTLMSSP unicode asn1 overflow attempt** – the threat specific for Windows XP. The error in a system library on an unpatched Windows XP OS allows executing the commends sent on the host system.

As the above list of captured threats demonstrates, the service is able to capture threats on different computers when operated in common real networks.

### 5.5 Evaluation of test results

The testing pilot phase has demonstrated the functionality of the system in the whole range of its functions. Capturing, sending to the server and storing was done without problems. The feedback from users was positive, the installation and control was considered as sufficiently non-demanding. The only problem was in the WinPcap inability to identify certain network adapters. The Colander service demonstrated the ability to capture potential threats in real operation. The uniqueness in threat evaluation can be improved by reducing the number of rules applied. After tests, Colander can be considered as a research tool ready to install and increase the users' knowledge and information on existing threats. All identified problems, except for the issue in WinPcap non-detecting of specific network adapters, can be suppressed or eliminated by reduction or better focusing of the ruleset applied.

## 6. Future prospects and development

Besides optimisation mentioned previously, the next main step in future research and the application development will focus on anomaly detection. That means that threats detection will not rely solely on fixed static rules but will be based on an automatic learning in the system where Colander is installed. In addition, wider testing and selection of suitable signature rules will be one of the focuses.

Moreover, the data gathered from Colander clients will be analysed further. The result of the analysis can be not only detailed information about previous attacks but the prediction of attackers' behaviour in future attacks using various predictive techniques. An example of such a technique is described by Hu et al. in [25].

## 7. Conclusions

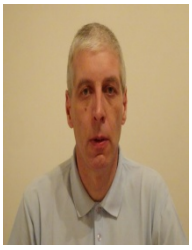
The main objective of the paper was to describe a newly developed tool that can be used for gathering attack data for further research. The program architecture and design was presented. The main emphasis was given to pilot and benchmark testing results. The results demonstrated both the efficiency of the tool, from the point of view of the system resource demand, and primarily the ability to capture known threats (based on signatures) as demonstrated mailny on the case of WannaCry.

## References

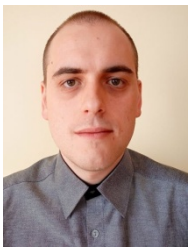
[1] Symantec, “Internet Security Threat Report,” April 2017. https://www.symantec.com/security-center/threat-report
[2] E. Cooke, M. Bailey, D. Watson, F. Jahanian, and J. Nazario, “The Internet motion sensor: A distributed global scoped Internet threat monitoring system,” *Technical Report CSE-TR-491-04, University of Michigan, Electrical Engineering and Computer Science*, 2004.
[3] M. Sourour, B. Adel, and A. Tarek, “Ensuring security in depth based on heterogeneous network security technologies,” *Int. J. Inf. Secur*., vol. 8, pp. 233-246, 2009. Article (CrossRef Link).
[4] M. Iturbe, I-aki Garitano, Urko Zurutuza, and Roberto Uribeetxeberria, “Towards Large-Scale, Heterogeneous Anomaly Detection Systems in Industrial Networks: A Survey of Current Trends,” *Security and Communication Networks*, Vol. 2017, 2017. Article (CrossRef Link).
[5] Khan, M.A., Salah, K., “IoT security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395-411, 2018. Article (CrossRef Link).
[6] W. Huang and J. Yang, “New network security based on cloud computing,” in *Proc. of Education Technology and Computer Science (ETCS), 2010 Second International Workshop on. IEEE*, pp. 604-609, 2010. Article (CrossRef Link).
[7] Rengaraju, P., Ramanan, V. R., and Lung, C. H., “Detection and prevention of DoS attacks in Software-Defined Cloud networks,” in *Proc. of Dependable and Secure Computing, (2017) IEEE Conference* on (pp. 217-223), IEEE, 2017. Article (CrossRef Link).
[8] R. Russell, “iptables (8) - Linux man page,”. https://linux.die.net/man/8/iptables
[9] Fail2ban. https://www.fail2ban.org/wiki/index.php/Main_Page
[10] PN. Ayuso, RM Gasca and L. Lefevre, “FT-FW: A cluster-based fault-tolerant architecture for stateful firewalls,” *COMPUTERS & SECURITY*, Vol. 31, Issue. 4 pp. 524-539, 2012. Article (CrossRef Link).
[11] S. X. Wu and W. Banzhaf, “The use of computational intelligence in intrusion detection systems: A review,” *Applied Soft Computing*, 10(1), pp 1-35, 2010. Article (CrossRef Link).
[12] F. Hock, and P- Kortis, “Commercial and open-source based Intrusion Detection System and Intrusion Prevention System (IDS/IPS) design for an IP networks,” in *Proc. of Emerging eLearning Technologies and Applications (ICETA), 2015 13th International Conference on (pp. 1-4). IEEE*, 2015. Article (CrossRef Link).
[13] H. Debar, D. Curry and B. Feinstein, “The Intrusion Detection Message Exchange Format (IDMEF),” *IETF*, 2007. Article (CrossRef Link).
[14] B. Feinstein and G. Matthews, “The Intrusion Detection Exchange Protocol (IDXP),” *IETF*, 2007. Article (CrossRef Link).
[15] P. Garcia-Teodoro, J. Dıaz-Verdejoa, G. Macia-Fernandeza, E. Vazquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges” *Computers & security*, vol. 28, no. 1-2, pp 18-28, 2009. Article (CrossRef Link).
[16] The Snort Project, “SNORT Users Manual 2.9.9,” chapter 3. Revision 2016. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node27.html
[17] J. T. Rodfoss, “Comparison of Open Source Network Intrusion Detection Systems,” 2011. https://www.duo.uio.no/bitstream/handle/10852/8951/Rodfoss.pdf

[18] S. Antonatos, K. Anagnostakis, and E. Markatos, "Honey@ home: a new approach to large-scale threat monitoring," in *Proc. of the 2007 ACM workshop on recurring malcode*, pp. 38-45, ACM, 2007. Article (CrossRef Link).
[19] D. K. Sadhasivan and K Balasubramanian, "A Fusion of Multiagent Functionalities for Effective Intrusion Detection System," *Security and Communication Networks*, Vol. 2017, 2017. Article (CrossRef Link).
[20] R. Kozik and M Choras, "Pattern Extraction Algorithm for NetFlow-Based Botnet Activities Detection," *Security and Communication Networks*, Vol. 2017, 2017. Article (CrossRef Link).
[21] Netmarketshare, "Market Share Reports,". http://www.netmarketshare.com
[22] Sophos, "Troj/Banker-FTC," 2017. https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj~Banker-FTC/detailed-analysis.aspx
[23] McAfee, "An Analysis of the WannaCry Ransomware Outbreak," 2017. https://securingtomorrow.mcafee.com/executive-perspectives/analysis-wannacry-ransomware-outbreak/
[24] Microsoft, "Microsoft Security Bulletin MS17-010 – Critical," 2017. https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010.
[25] H. Hu, H. Zhang, Y. Liu and Y. Wang, "Quantitative Method for Network Security Situation Based on Attack Prediction," *Security and Communication Networks*, Vol 2017, 2017. Article (CrossRef Link).
[26] Luo, Y., Xiang, K., Fan, J., Zhang, C. "Distributed intrusion detection with intelligent network interfaces for future networks," in *Proc. of IEEE International Conference on Communications*, 2009. Article (CrossRef Link).

**Dr. Matej Zuzčák**
Matej Zuzčák attained his Mgr. and RNDr. degrees in Information Systems from Department of Informatics and Computers of the Faculty of Science at University of Ostrava in 2016 and 2017. Since 2016 he has been working on his dissertation thesis within his Ph.D study at the Department of Informatics and Computers. His scientific research is focused mainly on honeypots, honeynets, network security, expert systems and data analysis. Since 2017 he is the team leader of CSIRT OU. He is also a member of The Honeynet Project in Czech chapter.



**Dr. Tomáš Sochor**
Senior lecturer at the Department of Informatics and Computers of the University of Ostrava. His research has been recently focused on computer network security, honeypots, spam detection techniques, web anonymization and ad-hoc network routing.



**Milan Zenka**
Milan Zenka attained his Mgr. degree in Information Systems from Department of Informatics and Computers of the Faculty of Science at University of Ostrava in 2017. He has been working towards attaining Ph.D at the Department of Informatics and Computers since 2017. His research is based on development of Windows based IDS. He has been a member of CSIRT OU since 2017.