AUTOMATYKA • 2011 • Tom 15 • Zeszyt 3

Artur Sierszeń*, Łukasz Sturgulewski*

Traffic Analyzer Based on Data Flow Patterns

1. Introduction

Nowadays, there are many systems of Network Intrusion Detection System type or Network Intrusion Prevention System type, which play the role of a firewall, filtrating network traffic based on pre-defined rules and detecting system intrusions. There are also more refined tools, e.g. honeypots, which create virtual environments imitating real networks in order to lure an intruder, draw his attention from key, sensitive, and real systems, and collect as much data as possible on the intruder and the attack itself. Recently, Network Behavior Analysis mechanisms have appeared which, collecting data in an isolated model environment, create a baseline of a correctly operating network in order to verify its operation and detect all anomalies.

This work presents a traffic analyzer which is a functional core of a larger system being developed by the authors. This system is expected to combine all functional properties of the aforementioned tools, support the configuration of ready-made technological solutions, enable simple configuration by a transparent user interface, operate in heterogeneous network environments, and be able to be implemented as a dispersed system.

2. Design guidelines

A network traffic analyzer is, by assumption, a multi-thread application which is able to perform simultaneous analyses of performed attacks concerning various attack types.

It was also assumed that the application will be used on various platforms, what was made possible by using Java language (the application operates on all hardware platforms which support the virtual machine of this language) and jNetPcap library. Owing to the efforts of the authors of this library, the markup language object does not have much influence on application effectiveness. During work with sniffed packets, one does not work on the copy of memory fragments but on indices to native structures provided by the system

* Computer Engineering Department, Technical University of Lodz, Poland

(in the case of Windows systems, they are provided by the winpcap library; systems from the Unix family are based on libcap implementation). Furthermore, the library is accompanied by a powerful editing tool called Packet Decoding Framework, which contains classes describing headers structure of the majority of popular network protocols, what facilitates work with sniffed packets significantly (the programmer does not have to know the detailed distribution of fields in a packet mapped onto the memory; he only refers to individual fields and methods of the class responsible for work with the relevant protocol). Another advantage of using the jNetPcap library is its author's attention to repairing imperfections in work with system libraries (such as libcap or WinPcap).

A network traffic analyzer may work as an independent monitoring system on a single working station or it may be operated on a host using e.g. TAP tool (a passive network tool which sends copies of circulating packets to a monitoring server in a way which is transparent for network operation) and monitor the whole traffic in a pre-defined segment. In the Figure 1, an example of the analyzer implementation in practice is presented as a system which monitors and protects a single working station in a local network.

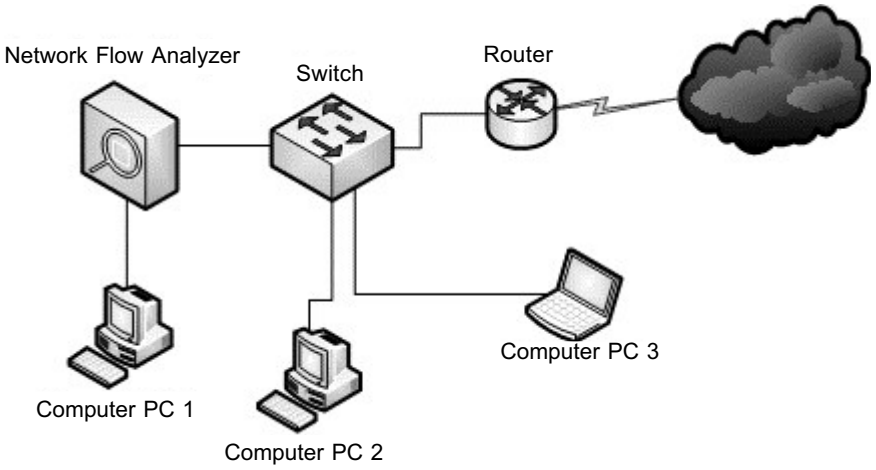

Fig. 1. Possible practical use of the traffic analyzer

The network analyzer system is based on statistical analyses of network traffic and detection of irregularities as compared to the assumed (or measured by the application) norm, what is similar to the operation of Network Behavior Analysis. Furthermore, it will detect anomalies in the form of signatures (ARP-spoofing, ICMP flooding, OS Fingerprinting attacks etc.) defined in the application. The second operation and detection system is similar to IDS/IPS Snort applications.

The application imports packets in the real time; other threads of the same application analyze incoming packet sequences, searching for attack patterns (analyzed by the application in search of sequences) or characteristic header values, what is presented in the Figure 2.

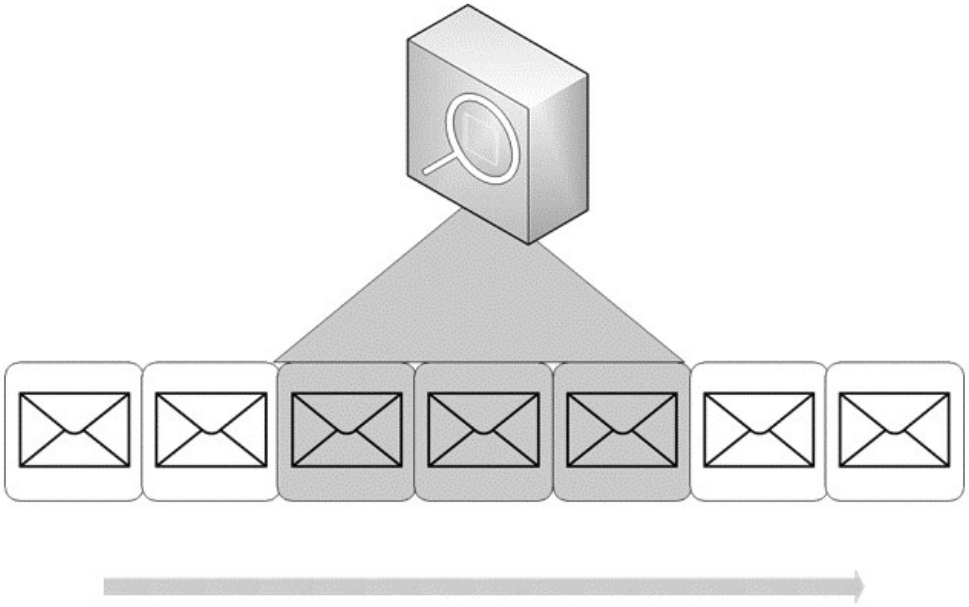

Fig. 2. Analysis of incoming packets by a network analyzer

An important issue was the construction of an intuitive user interface, which enables easy configuration and starting the application. Moreover, the network traffic analyzer can save current network statistics to a file (especially for the purposes of the analyzer, a file format was designed which contains application data – Network Analyzer Data (NAD)) in order to be able to use them as a network base profile to examine current sniffing or create statistical reports concerning measurements. The application generates Alerts if a known anomaly occurs, providing information about the time of occurrence and the potential source of disturbances.

3. Implementation of the Analyzer mechanisms

As has already been mentioned, Java language was used for implementation. The network traffic analyzer has the following protocol types implemented:

- Transmission Control Protocol [15],
- User Datagram Protocol [11],
- Internet Protocol version 4 [12],
- Internet Protocol version 6 [7],
- Hypertext Transfer Protocol [2],
- Internet Control Message Protocol [13],
- Address Resolution Protocol [14],

– IEEE802dot1q [1],
– Secure Socket Layer [10],
– Real-time Transport Protocol [5],
– Session Description Protocol [6],
– Session Initiation Protocol [9],
– Layer Two Tunneling Protocol [8],
– Point to Point Protocol [4].

As a part of his engineer thesis, the student developed a graphic interface and the application engine [3]. Main tabs concern the basic configuration, management of the current packet sniffing, management of collected statistics, notifications about alert situations in the network, user signatures configuration, network profile management, and the adaptation of the application appearance (graphic style used in the application) respectively.

The design utilizes the Swing library, components of which are diversified and rich in functions; therefore, they are sufficient for implementing all functions and they do not cause troubles with transfers between separate system platforms, in Figure 3.

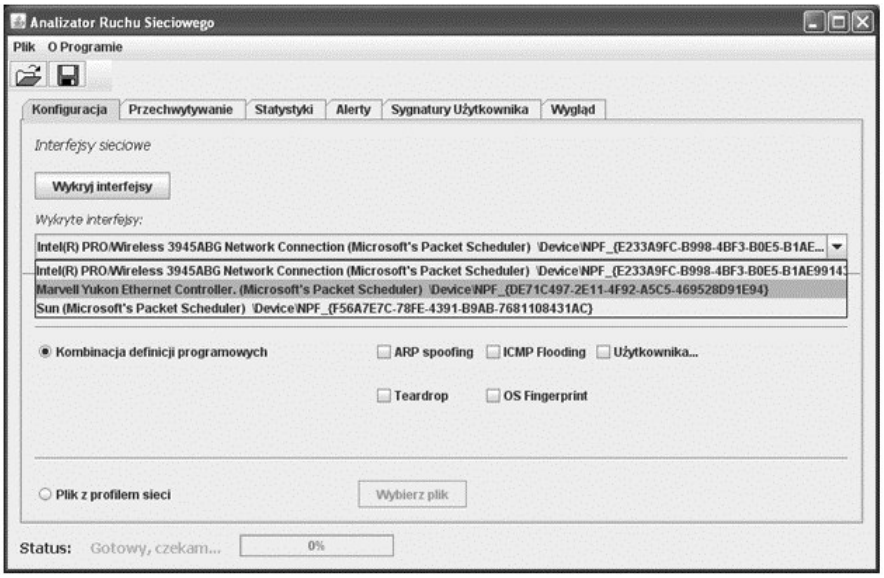

Fig. 3. Presentation of a list containing Windows XP network interfaces detected by the analyzer

Values entered into the list presented above were generated by a computer controlled by Windows XP SP 3. The list contains the interface of a wireless network card present in the system and the Ethernet card. As a comparison to the description of network interfaces presented above, the Figure 4 shows the list of interfaces obtained in the operating system from the Linux family (Ubuntu 10.10).

The Figures above indicate that the analyzer may be used on many different system platforms.

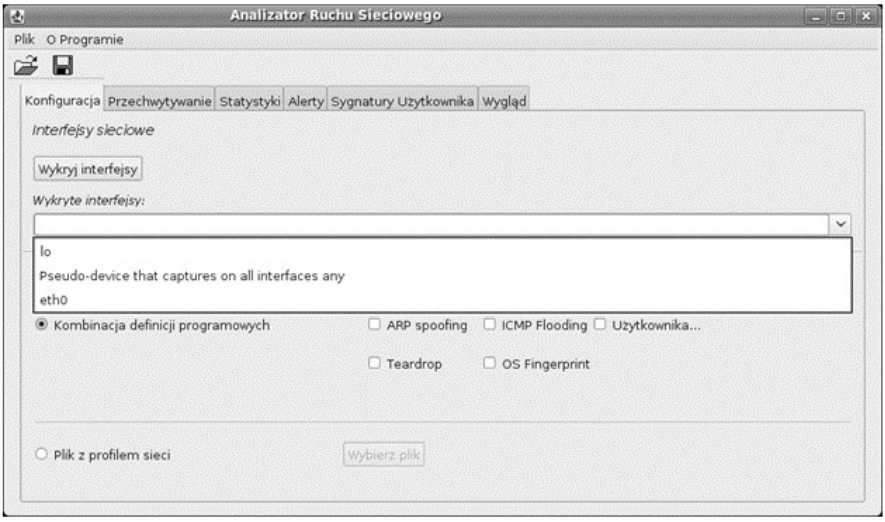

Fig. 4. The list of network interfaces of Ubuntu 10.10 detected by the analyzer

4. Implementation and tests of the network analyzer

For tests conducted using the Network Traffic Analyzer, the following test network topology was developed, presented in the Figure 5.

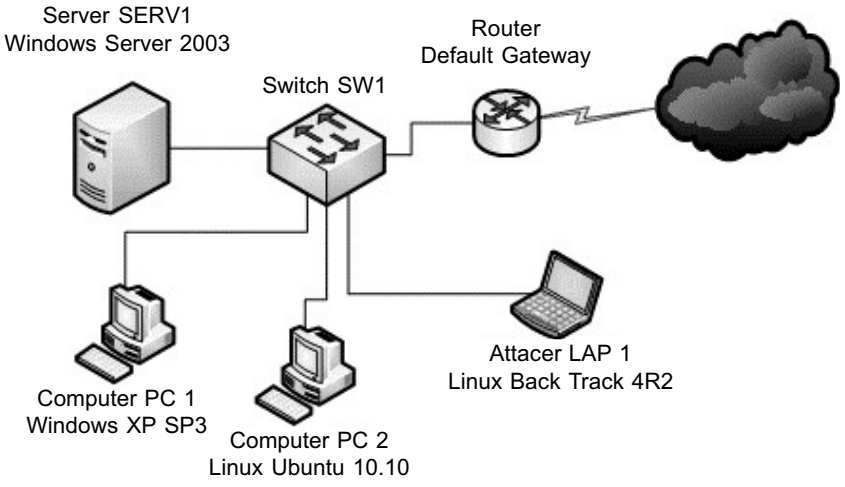

Fig. 5. Test network topology

The first test on the operation of a pattern recognition system was the test using a pattern generated based on the real traffic in the examined network, which was saved in

the form of the network default profile. The Network Traffic Analyzer was initiated on the server SERV1 marked in the diagram of the network topology, setting it to the monitoring mode for the loop sniffing packets endlessly; the time limit was set to 1800 seconds, i.e. a half-hour measurement. At the same time, a WWW server was operating on SERV1 with a homepage of 44 bytes which was downloaded by PC1 and PC2 cyclically every 3 minutes through queries directed by browsers installed on these computers. The statistic profile of the network created this way was saved to a NAD file. This profile was used as a network base profile. During the next half-hour session, simulation conditions were repeated but the LAP1 computer, which played the role of an attacker, conducted ICMP-Flooding attack on SERV1. The Network Traffic Analyzer was set to tolerance for profile deviation of 7%. After the analysis has been completed, the application notified that the profile norm concerning the percentage share of packets sent with TCP and ICMP protocols was exceeded. The difference concerning the latter was significant and it was 22.20%, what is a correctly recognized symptom of an ICMP-Flooding attack. Figure 6 presents the graphical comparison of values reported by the Analyzer.

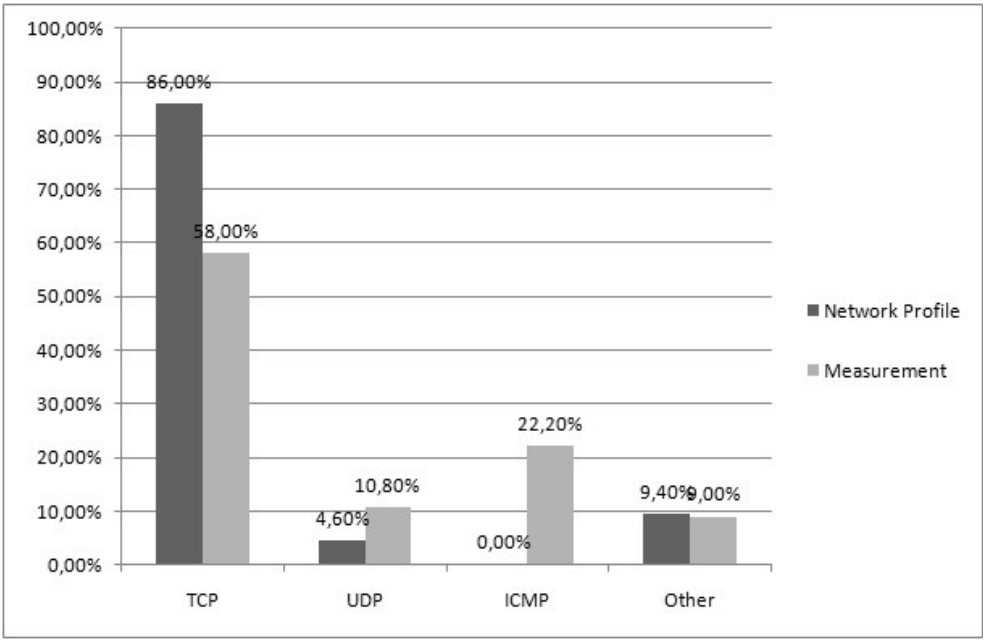

Fig. 6. Graphic presentation of the comparison between the Analyzer results and the network profile

The next test conducted using patterns based on the saved network profile was the attempt of detecting undesirable network traffic; in the test it consisted in starting P2P application services on one computer and downloading data using one of more popular clients of

this network. The testing environment was not changed significantly as compared to the previous test; however, an additional FTP service was started on SERV1. The network profile was created based on a 30-minutes' observation of the network by the Analyzer started on PC1. As in the previous test, PC1 and PC2 directed queries at SERV1 every 3 minutes alternately, choosing FTP or WWW services at random. After the profile has been created, the traffic analysis was started within the time period of 30 minutes during which a P2P client service was started on PC1 and files were being downloaded. After the observation has been completed, the application notified that the norm for traffic with UDP protocol was slightly exceeded (it increased by 12.4%) and that sent and received data increased significantly. This indicates anomalies in network use; it may be a symptom of unauthorized attempts of copying resources or using P2P client services (which we had defined as undesirable in the case of the test network). The chart in Figure 7 presents the percentage increase in sent and received data.

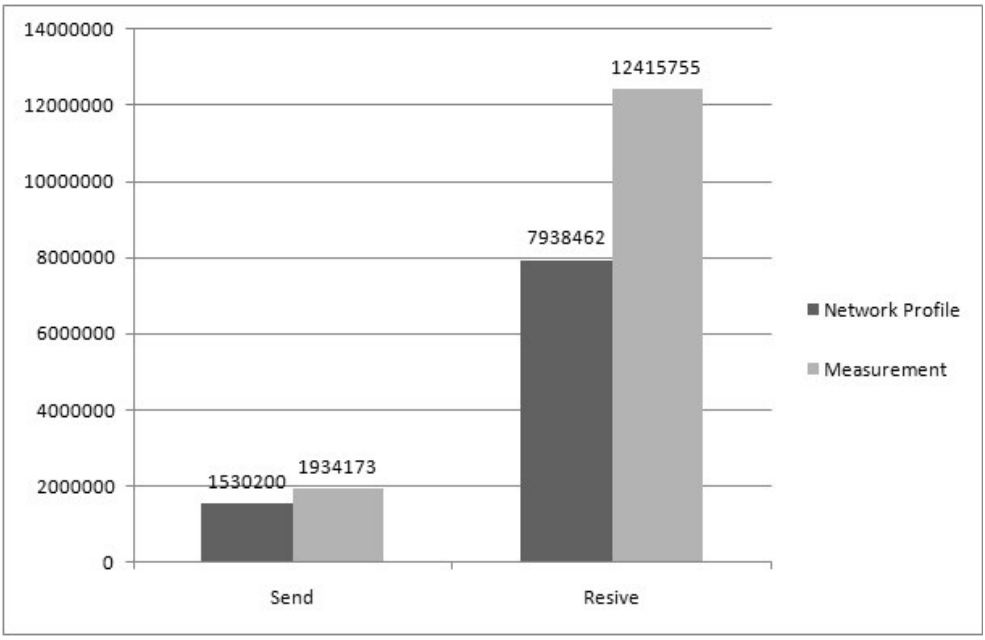

Fig. 7. Graphic presentation of the comparison between the Analyzer results and the network profile

After tests with network profiles, another series of tests was conducted which checked the functioning of application signature mechanisms, i.e. the detection of ARP Spoofing, ICMP Flooding, Teardrop, and OS Fingerprinting attacks. As in previous tests, the testing environment was started and WWW and FTP services were enabled on SERV1; these received cyclic queries from PC1 and PC2. The time of a single test was from 15 to

35 minutes. During this time, attacks were conducted from the computer which played the role of an aggressor using software which tests protections and network sensitivity available from Linux BackTrack 4R2. The results of the tests are presented in the Table 1.

Table 1
Test results concerning attack detection using patterns which were defined a priori

Testing time [s]	No. of attacks	Attack types	False alarms	Undetected attacks
1440	1	ARP Spoofing	1	0
1080	2	ICMP Flooding	0	0
1440	6	OS Fingerprinting	2	2
900	1	Teardrop	1	1
900	4	ARP Spoofing, ICMP Flooding	1	1
1860	5	ARP Spoofing, ICMP Flooding, Teardrop	2	1
600	2	ICMP Flooding, OS Fingerprinting	0	0
1860	4	ARP Spoofing, ICMP Flooding, OS Fingerprinting	0	1
1500	1	ARP Spoofing	0	0
2100	2	ICMP Flooding	0	0
900	6	OS Fingerprinting	2	1
960	4	ARP Spoofing, ICMP Flooding, Teardrop	2	1
1020	10	ICMP Flooding, OS Fingerprinting	3	2
1500	5	OS Fingerprinting	0	0
1560	1	ARP Spoofing	0	0

As a result of tests using patterns defined a priori (15 tests were carried out during which 54 simulated attacks were conducted on the computer with running Analyzer, which was supposed to detect these attacks), the application reached the efficiency levels concerning the detection of attacks of almost 81.5%, what may be regarded as a very good result and a high efficiency level. That is, of course, the efficiency level reached with respect to attack types known to the application. The authors did not manage to prevent false alarms (during the tests, they constituted app. 24.1% of all reported notifications on security breaches in the tested network), which are present in all modern systems protecting computer network. Elimination of false alarms is laborious; their complete elimination is impossible.

5. Conclusions

The Analyzer operates as a multi-thread application; it works in the real time and on various operating systems (it has been tested on Windows XP SP3, Windows Server 2003, and Linux Ubuntu 10.10). Measurements performed on the tested systems indicated that the processor load due to the Analyzer did not exceed the average of 5.5% of CPU usage. The Network Traffic Analyzer correctly collects packets from network interfaces, presents the collected data, and identifies network anomalies which were programmed as a priori patterns.

The Network Traffic Analyzer implements a basic function of behavioural analysis offered by the aforementioned NBA systems. However, an NBA module left on its own loses its usefullness; therefore, the cooperation of various systems seems necessary. Linking the analyzer with modules of a greater system should increase the detection and efficiency levels. The module construction of the application enables the implementation of further test types without disturbing the application core.

The Network Traffic Analyzer is a fully operational application for tasks related to network analysis and prevention of network threats. Its module construction (it is an object application with rich documentation of classes and its interface) makes it a potential basic component of a wider system for analyzing network traffic. It is possible to expand it with further implemented patterns, protection mechanisms, and strategies of reaction to events occurring in a network.

References

[1] ANSI/IEEE 802.1D-2004, American National Standards Institute /Institute of Electrical and Electronics Engineers, 2004.
[2] Berners-Lee T., *HyperText Transfer Protocol*. World Wide Web Consortium, retrieved 2010.
[3] Dubel M., *Network traffic analyzer based on data flow patterns* (supervisor: A. Sierszeń). Technical University of Łódź, 2011 (Master's thesis).
[4] RFC 1661 – The Point-to-Point Protocol (PPP), Internet Engineering Task Force, 1994.
[5] RFC 1889 – RTP: A Transport Protocol for Real-Time Applications, Internet Engineering Task Force, 1996.
[6] RFC 2327 – SDP: Session Description Protocol, Internet Engineering Task Force, 1998.
[7] RFC 2460 – Internet Protocol, Version 6 (IPv6) Specification, Internet Engineering Task Force, 1998.
[8] RFC 2661 – Layer Two Tunneling Protocol „L2TP", Internet Engineering Task Force, 1999.
[9] RFC 3261 – SIP: Session Initiation Protocol, Internet Engineering Task Force, 2002.
[10] RFC 5246 – The Transport Layer Security (TLS) Protocol, Internet Engineering Task Force, 2008.
[11] RFC 768 – User Datagram Protocol, Internet Engineering Task Force, 1980.

[12] RFC 791 – Internet Protocol, Darpa Internet Program, Protocol Specification, Internet Engineering Task Force, 1981.
[13] RFC 792 – Internet Control Message Protocol, Internet Engineering Task Force, 1981.
[14] RFC 826 – Ethernet Address Resolution Protocol, Internet Standard STD 37, Internet Engineering Task Force, 1982.
[15] RFC: 793 – Transmission Control Protocol, Darpa Internet Program, Protocol Specification, Internet Engineering Task Force, 1981.