PROJECT

Multi-tenant cybersecurity operations centre designed
as an open cloud service with machine learning elements

V9 Description of the selected algorithm meeting the specified criteria

KPB3 – Design of ML algorithms

type – documentation

month of submission – M17

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Contents

1 Introduction 3
2 Network anomaly detection 8
2.1 Detection of DoS/DDoS attacks 8
2.1.1 DoS attacks – datasets, experiments and results, description of the selected algorithm 11
2.1.1.1 Statistical parameters 14
2.1.1.2 3σ-Tunnel 19
2.2 Conclusion 21
3 Detection of phishing e-mails and spoofing 23
3.1 Experiments and results 24
3.2 Description of the selected algorithm 25
3.3 Conclusion 27
4 Spam/ham classification 28
4.1 Experiments and results 29
4.2 Description of the selected algorithm 29
4.3 Conclusion 30
5 Detection of advanced APT techniques 32
5.1 Experiments and results 33
5.2 Description of the selected algorithm 33
5.3 Conclusion 36
6 Classification and identification of malware samples 38
6.1 Experiments and results 39
6.2 Description of the selected algorithm 39
6.3 Conclusion 39
7 Conclusion – proposal 41
Annexes 42

1 INTRODUCTION

Machine learning (ML) and artificial intelligence (AI) in general have been used for data processing and analysis purposes for several decades in a wide range of applications, and cybersecurity is no exception.

A security operations centre (SOC) is a centralized unit responsible for monitoring, detecting, responding to and mitigating cyber threats in real time. The growing volume, speed and complexity of cyber threats have rendered traditional rule-based analysis and manual analysis insufficient.

AI/ML algorithms and methods make it possible to extend the SOC primarily with the following functions:

- automation of recurring analytical and operational tasks (e.g. alert enrichment, log normalization, incident triage)
- classification of security events and incidents (distinguishing benign vs. malicious network traffic, distinguishing known threats and suspicious anomalies)
- detection of unknown threats and deviations from normal behaviour (zero-day attacks, novel attack patterns, insider threats)
- reduction in the number of false positive reports or alerts
- prioritization and risk assessment of security alerts (risk scoring, threat scoring, contextual severity)
- support for predictive analysis of security risks (prediction of the probability of incidents or of attack escalation)
- faster response to security incidents (real-time decision support, recommendation of response steps).

AI/ML methods and algorithms are integrated into the SOC system workflow in several phases:

- **Monitoring**: In the monitoring phase, AI/ML methods are used primarily for the continuous collection, pre-processing and basic analytical interpretation of security data from various infrastructure sources. The aim of this phase is not the final identification of an attack, but the early detection of deviations from normal behaviour and the creation of context for subsequent detection mechanisms. The aim of AI/ML-supported monitoring is to minimize the volume of irrelevant data, identify potentially risky behaviour at an early stage and prepare high-quality inputs for the incident detection and analysis phase. In this phase, AI/ML supports in particular:
  - anomaly detection in logs and telemetry in real time (network logs, system and application logs, cloud and endpoint telemetry; statistical models, unsupervised learning, behavioural baseline models)
  - modelling the normal behaviour of systems, services and networks (temporal patterns, volume characteristics, frequency and seasonal behaviour)
  - behavioural analysis of user and entity behaviour, UEBA (User Entity Behavior Analytics) (profiling the behaviour of identities, devices and accounts; detection of non-standard activities and potential insider threats)
  - automated filtering and enrichment of monitored data (log normalization, noise reduction, contextualization of events)
- **Detection:** The detection phase follows on from monitoring and represents the process of identifying and confirming a security event or incident. In this phase,

AI/ML methods are used to evaluate suspicious signals that arose during monitoring and to decide whether the observed activity represents a security threat. The aim of the detection phase is to identify security incidents accurately, minimize the number of false positive reports and provide high-quality inputs for the subsequent triage and analysis phase.
In this phase, AI/ML supports in particular:

- classification of security events and network traffic (distinguishing benign events, known attacks and suspicious activities; supervised learning, ensemble methods)
- correlation of events across multiple data sources (network, application, endpoint and identity data; incident contextualization)
- ML-based intrusion detection and prevention systems, IDS (Intrusion Detection System)/IPS (Intrusion Prevention System)
- detection of both known and unknown threats without the use of signatures (behavioural models, anomalies, heuristics, zero-day attacks)

- **Triage and analysis:** The triage and analysis phase represents the transition between detection and detailed investigation of incidents. Its aim is to reduce the number of security alerts, determine their relevance and priority and optimize the use of the SOC's analytical capacity. In this phase, AI/ML methods are applied mainly for automated decision support and for reducing analysts' manual work. The aim of the triage and analysis phase is to ensure that the SOC's analytical capacity is focused on the most critical incidents, while less relevant or duplicate events are automatically suppressed or aggregated.
  In this phase, AI/ML supports in particular:
  - prioritization of alerts (risk scoring, severity prediction, regression and classification models taking into account context, historical data and the impact on the organization)
  - grouping and correlation of similar alerts (clustering, deduplication, temporal and contextual correlation of events)
  - reduction of false positive reports (false positives) (learning from analysts' historical decisions, adaptive thresholds)
  - enrichment of incidents with contextual information (integration of CTI, asset criticality, identities, geolocation, reputation score)
- **Investigation**: The investigation phase represents deeper analytical and forensic work aimed at confirming the incident, determining its scope, cause, course and impact, and obtaining evidence for corrective measures. In this phase, AI/ML methods are used mainly to correlate heterogeneous data, reconstruct the attack chain and support the analyst in decision-making, not as an autonomous replacement for the investigation.
  The aim of the investigation phase is to provide an explainable and reproducible analytical output (what happened, why, how, with what impact) that enables an accurate response and subsequent prevention.
  In this phase, AI/ML supports in particular:
  - event correlation and reconstruction of the course of the incident (timeline and attack chain reconstruction) (linking events across logs, endpoint telemetry, network traffic and identities; identification of sequences typical of TTPs)
  - natural language processing (NLP) for correlating threat information (triage and summarization of tickets, CTI reports and incident descriptions, correlation of IOCs/TTPs from text; extraction of entities and relevant indicators)

  - graph analysis for the purposes of visualizing attacks and relationships between entities (identity–host–process–network connection–object; detection of lateral movement, pivoting, visualization of attack paths)
  - similarity detection and clustering of incidents according to behavioural patterns (case similarity, grouping by TTP/IOC; support for repeated investigation and the SOC knowledge base)
- **Response and remediation:** The response and remediation phase follows on from the investigation, and its aim is to limit the impact of the incident, restore the system to a secure state and prevent recurrence. In this phase, AI/ML methods are used primarily as support for decision-making and automation within predefined processes, not as an autonomous substitute for human decision-making. The aim of this phase is to shorten the response time (MTTR), ensure consistent and repeatable response procedures and support the analyst in decision-making without losing control over the process.
  In this phase, AI/ML supports in particular:
  - automated response strategies via SOAR with ML decision support (endpoint isolation, blocking IP addresses, deactivating accounts, updating rules; execution only within approved playbooks)
  - adaptive adjustment of response strategies (learning from the results of previous interventions, optimization of the order and timing of responses)
  - predictive recommendations
- **Post-incident response:** The post-incident response phase is a retrospective analytical phase whose aim is to understand the causes of the incident, evaluate the effectiveness of the measures taken and improve the SOC's preparedness for future security threats. In this phase, AI/ML methods are used for the systematic analysis of historical incidents and to support continuous improvement of security processes. The aim of the post-incident response phase is to increase the resilience of the security environment, reduce the likelihood of similar incidents recurring and ensure that the SOC systematically learns from its own experience.
  In this phase, AI/ML supports in particular:
  - analysis of the root causes of incidents (root cause analysis) (identification of recurring patterns, vulnerabilities and weaknesses through clustering, correlation and retrospective analysis)
  - evaluation of the effectiveness of detection and response mechanisms (analysis of metrics such as MTTD, MTTR, the number of false positive/negative reports, feedback from incidents)
  - forecasting trends and the evolution of future attacks (time series, comparison of incidents over time, changes in TTPs and attack patterns)
  - support for updating detection models, rules and playbooks (using lessons learned from incidents to adapt ML models and operational procedures)

Typical areas of use of AI/ML algorithms and methods and their applications include the following:

- improving threat intelligence (Threat Intelligence Enhancement)
- improving security information and event management, SIEM (Security Information and Event Management (SIEM) Enhancement)
- endpoint detection and response, EDR (Endpoint Detection & Reaction) (behavioural analysis of processes, detection of lateral movement, automated interventions within approved policies)

- network security (network traffic analysis, anomaly detection, identification of C&C communication, DoS/DDoS and lateral activities)
- phishing and fraud detection (classification of e-mails and messages, analysis of content and metadata, behavioural communication patterns)
- detection of insider threats (Insider Threats) (analysis of user and entity behaviour, correlation of identity and system events)
- automated incident response (SOAR, optimization of response procedures, learning from previous interventions)
- vulnerability management (prioritization of vulnerabilities based on the context of the environment, the likelihood of exploitation and historical incidents).

Since a detailed analysis of AI/ML algorithms and methods is provided in the output document V7 Výsledok analýzy vhodných algoritmov.docx, here we only briefly summarize the most frequently used approaches and examples of use.

- supervised learning (Supervised Learning)
  - decision trees (Decision Tree), random forests (Random Forest), logistic regression, support vector machine (SVM)
  - malware detection, filtering of spam e-mail messages, classification of phishing e-mail messages
- unsupervised learning (Unsupervised Learning)
  - clustering algorithms k-means, HDBSCAN, isolation forest (Isolation Forest), one-class SVM
  - insider threat detection (Insider Threat Detection), unknown threat discovery (Unknown Threat Discovery)
- deep learning (Deep Learning)
  - recurrent neural networks RNN (Reccurent Neural Network), networks with memory LSTM (Long Short-term Memory), convolutional neural networks CNN (Convolutional Neural Networks), autoencoders
  - detection of advanced persistent threats (APT)
- natural language processing NLP (Natural Language Processing)
  - BERT, Transformer models
  - automation of threat information intake, analysis of data from the dark web
- reinforcement learning (Reinforcement Learning)
  - Q-learning, deep Q-networks DQN (Deep Q Networks)
  - dynamic adjustment of firewall rules, automated penetration tests
- graph-based methods
  - graph neural networks GNN (Graph Neural Networks)
  - mapping the spread of attacks, fraud detection

Within the proposed SOC solution, we envisage the use of security logs obtained from SIEM platforms and their associated collection components (ELK Stack, Wazuh or network sensors such as Suricata), which serve as the primary data sources for our analysis and detection of security events. The outputs of the proposed analytical layer based on AI/ML methods are conceived as a complementary analytical and evaluation mechanism that builds on the existing detection capabilities of the SIEM system. The aim of this layer is not to replace the SIEM, but to extend its functionality with advanced analytical capabilities, particularly in the areas of incident severity assessment, event correlation and the identification of complex or non-standard behavioural patterns.

The results of the ML analysis can be used for an additional assessment of the level of risk and the context of a security incident or threat that was previously detected by the existing SIEM mechanisms, thereby supporting the decision-making of SOC analysts in subsequent phases of incident handling.

In the next part of the document, we focus on specific categories of security threats that were identified, on the basis of an analysis of the environment and the available data, as the most significant from the perspective of the proposed innovative SOC system.

2 NETWORK ANOMALY DETECTION

The detection of behavioural anomalies in computer systems is an essential step in many defence mechanisms.

The main aim of anomaly detection (AD) is to learn the normal behaviour of a system over a certain period of time and then flag any activity that deviates from this learned behaviour. In many cases, learning normal behaviour requires monitoring and recording all activities performed in the system in order to extract a pattern or profile for each user/application.

Owing to their wide applicability, anomaly detectors are successfully used in many areas, such as the monitoring of networks or operating systems.

Anomaly detectors usually consist of two modules, namely a data collector and a data analyser. The first module (the data collector) gathers data from the protocols/logs of the monitored system, while the task of the second module is to analyse the gathered data. Depending on the location of the data collector, there are two types of anomaly detectors: network-based and host-based. These two types of detectors generate different types of logs, and therefore the features used for anomaly detection differ depending on the location of the detectors.

The given data analyser performs anomaly detection on the gathered logs, and various kinds of methods are used for this task, which can generally be divided into the categories of supervised learning, semi-supervised learning and unsupervised learning. More information can be found in the document V7 Výsledok analýzy vhodných algoritmov.docx. Within each category there are many different applicable algorithms. In addition, the parameters/features that can be used for AD vary depending on the types of logs. For example, from network packets the detector can use the inter-arrival time between packets or the number of packets in one direction as features. From host logs, the detector can use CPU/memory utilization. The features may also differ depending on the specific threat against which the anomaly detector is deployed. For example, an anomaly detector for detecting ransomware uses parameters that differ from those used by an anomaly detector for detecting cryptojacking.

There are several challenges in deploying AD systems. The main challenge is how to obtain normal behavioural patterns. It is difficult to define unambiguously what "normal behaviour" is, and therefore collecting such data is demanding. Another challenge is that the detector may suffer from a high number of false positive results and thus raise a warning for any unexpected pattern of network behaviour.

Unfortunately, unexpected patterns need not always mean that an intrusion has occurred. More precisely, there may be many unexpected normal behavioural patterns, for example as a result of network congestion or of obstacles or problems in systems. Reducing the number of false positive results is therefore key to the successful deployment of AD systems.

2.1 Detection of DoS/DDoS attacks

DoS (Denial of Service) and DDoS (Distributed Denial of Service) techniques occur in various types of attacks carried out in IP networks and in incidents at various levels of the network.

The abbreviation DoS translates as an attack causing the denial of some service. A very common example of a DoS attack is overwhelming a server's link (flooding), the server's CPU or its process/service with a flood of packets or requests. In this case, the attacker uses an attacking device with which they attempt to disrupt the normal operation of the target server or its service, or of the network leading to the server. DoS attacks very quickly became a cybersecurity threat. They are simple, but their impact can be devastating for individuals as

well as companies.

DoS attacks are often used to shut down devices and networks, and there are many different ways of using the above DoS attacks. These include, for example:

- Buffer overflow: A buffer is temporary storage for holding data. If a program or system process places more data than was predetermined, the additional data then overflow, and some of them may then leak into other buffers. This then results in the corruption or overwriting of other data. On the basis of this principle, an attacker can cause an application to crash and thus a DoS, although this attack is primarily used to manipulate the internal state of a program, inject and execute their own code (so-called shellcode) or gain unauthorized access or escalate privileges.
- ICMP attack: ICMP is a control protocol of IP networks that uses ICMP requests and responses. In general, echo requests and echo replies are used as an application, the so-called ping of a network device, in order to verify connectivity between the device and the sender. If an attacker flooded the target with ICMP request packets, the network must respond with the same number of packets, which makes the device unavailable for other incoming requests.
- UDP attack: The attack is based on sending UDP datagrams with a source IP address that the attacker can spoof, and the aim is to flood random ports on the target server. The advantage of the UDP protocol is that it does not need to establish a connection, as is the case with the TCP protocol. If the target server receives a UDP datagram on a port that is not used by any application, it informs the sender of this. If the attacker sends a sufficient number of packets to such a port, it may become overloaded.
- SYN attack (SYN flood): This is an attack that arises when a TCP connection is established between 2 devices. TCP uses a three-way handshake. Establishing a connection begins with sending a SYN message. If the server is active, a SYN/ACK message is sent back to the client that wanted to establish the connection. The server tries to respond until the time for establishing communication expires, while keeping the data in memory. If the client is an attacker with a spoofed and unreachable source address, sending a large number of such packets can exhaust the memory allocated for TCP connections, and all further communication attempts will then be ignored.
- Teardrop attack: During this attack, the attacker attempts to send fragmented packets, and the host's network layer then tries to reassemble these data into their original state. The attacker can exploit a vulnerability of an outdated operating system. It is common for many companies to still use older versions of systems in order to keep running an old application, and since some older versions of operating systems, such as OSs from the 1990s, contain a bug, a station may be compromised precisely because of this vulnerability when a fragmented packet is reassembled.

Unlike a simple DoS attack, a DDoS (Distributed Denial of Service) attack gains strength because the attacker uses multiple systems at once and thus increases the severity of the targeted attacks. The attacker can use malware, which they send over the network to devices with the intention of infecting them. They thus become part of a so-called botnet, and, following the attacker's commands via a command server, these bots then attack the targeted server in unison. DDoS attacks are very difficult to detect because the attacking bots may be located in various places around the world and their individual flow is not voluminous. The main and key difference between a DoS and a DDoS attack is that with DDoS, using bots, the attacker can send enormous volumes of traffic.

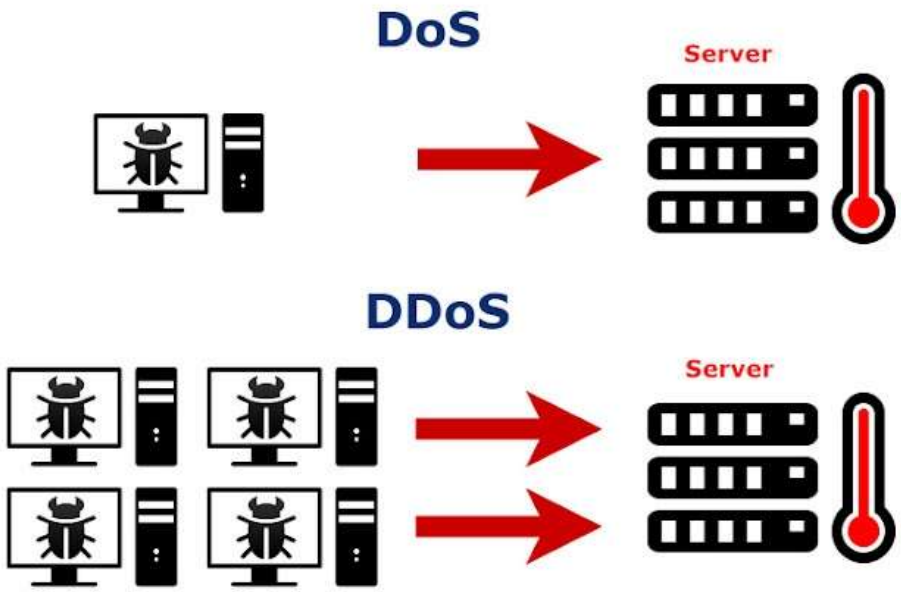


Fig. 1: DoS and DDoS attack

DDoS attacks are divided into three main categories:

- **Volumetric attack:** Volumetric attacks are a type of DDoS attack whose aim is to exhaust the available bandwidth of the target or its upstream provider. The attacker generates an extremely large amount of network traffic (e.g. UDP flood, ICMP flood), and may — but need not — also use reflection or amplification mechanisms via services such as NTP, DNS or SSDP. Modern volumetric attacks often reach hundreds of Gb/s to several Tb/s, which can lead to complete unavailability of the service or saturation of the transit even before the packets reach the target.
- **Protocol attack:** Protocol attacks are a type of DDoS attack that exploit the way in which network protocols and devices process traffic. The aim is not to fill the bandwidth, but to exhaust the state or system resources of a server, firewall, load balancer or other network element. A typical example is a SYN flood, in which the attacker generates a large number of TCP connections but never completes them. The device must therefore hold a large number of half-open states until it exhausts its connection-tracking table. In addition to SYN flood, protocol attacks include, for example, ACK flood / RST flood (firewall state exhaustion), Ping of Death, Smurf attack, etc. Unlike volumetric attacks, these types of attacks often do not require extreme bandwidth, but they can effectively paralyse the infrastructure by exhausting its internal resources
- **Application layer attack:** Application attacks target a specific service or application (e.g. a web server, an API) and attempt to exhaust its computational resources, database or application logic. The attacker generates requests that are valid from the protocol's point of view (e.g. HTTP GET/POST) but are computationally or database-intensive for the server. The aim is to prevent the application from serving legitimate users, even though the overall transmission capacity of the network is not exhausted. These attacks are usually measured in requests per second (RPS). Application attacks are dangerous in that they outwardly look like legitimate traffic and are harder to detect with classical DDoS filters.

There are several signs that may signal an ongoing DDoS attack on a network, an application or the entire infrastructure. The most common include:

- A sudden and unusual increase in network traffic (L3/L4) that does not correspond to normal behavioural patterns.

- Reduced application performance or delayed responses, even though the server's internal resources do not appear overloaded.
- Inability to communicate with the service, interrupted connections or frequent time-outs.
- A significant increase in requests to specific devices (L7), often in the hundreds to thousands of RPS, which do not originate from legitimate users.
- Abnormal utilization of network elements (firewall, load balancer, reverse proxy), e.g. full connection-tracking tables or high CPU.
- Sudden saturation of the provider's link, which manifests itself as packet loss and increased latency.

These signs by themselves need not confirm an attack, but they are a strong signal for immediate verification by means of NetFlow/IPFIX, firewall statistics, monitoring tools or telemetry.

2.1.1 DoS attacks – datasets, experiments and results, description of the selected algorithm

For our experiments, we used a large amount of different data obtained from multiple sources, such as ISCX 2012 or CIC-IDA 2017, but the main source was our own data obtained from network traffic not only within FRI UNIZA, but also across the entire network of the University of Žilina.

In both deterministic and stochastic models of an IP network, the cumulative process A(s, t), which describes the occurrence of packets in the time interval ⟨s, t⟩, is used to analyse an IP flow. For discrete-time models, certain time slots [ts] or sampling windows are given for the analysis, in which the number of occurring packets ai (increments at time i) is recorded:

$$\forall t,s \in R^{+}; \quad A(s,t) = A(0,t) - A(0,s) = \sum_{i=s}^{t} a_i$$

The records are sampled into given time windows (ts – time slot), mostly ts = 1ms. For the step-by-step calculation of the parameters, we used a shift of the so-called compute window (cw – compute window) by one ts. When carrying out the experiments, we used various powers of two (256, 512, 1024, 2048, 4096) for the compute window sizes because of the estimation of the Hurst exponent. After a subjective evaluation of the course of the values of the investigated parameters, we decided to deal further mainly with a compute window of size cw = 1024 ts. With cw = 512 ts, significant "rippling" occurred in some parameters; with cw = 2048, the parameter calculation time increased significantly, mainly for the Hurst exponent.

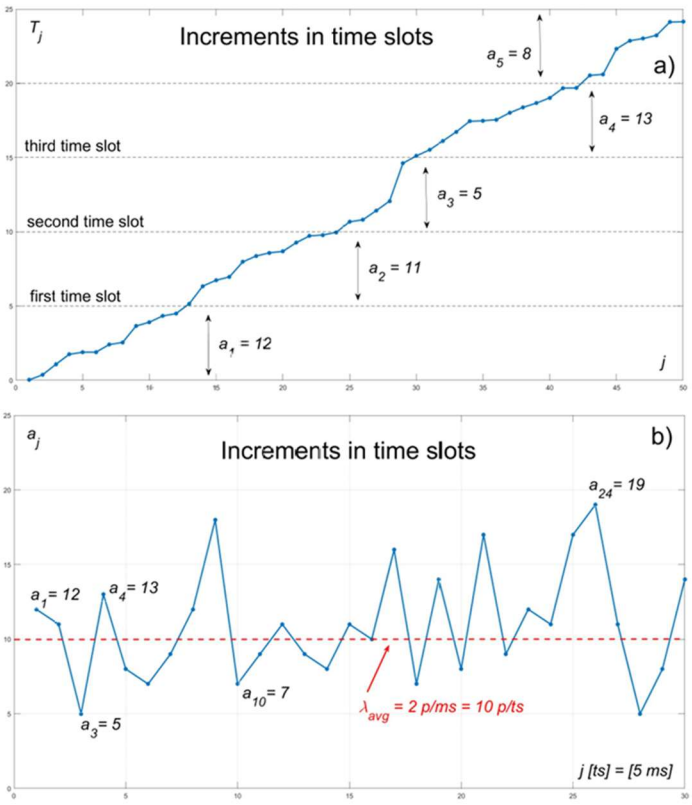


Fig. 2: Description of an IP flow: a) cumulative time, b) increments in time windows

To describe the probabilistic character of the observed IP flow, we selected statistical coefficients that can be divided into two types: simple (single) and double. Single coefficients take their value from only one window (computer window) of measured packet increments, or from one random sample of the given random variable. The next value is obtained by shifting the cw window by a number of time slots (shift). We usually used only one shift. By gradually shifting the cw, we thus obtain the values of moving statistical coefficients. These coefficients include: the mean, sample variance, the coefficients of variability, skewness and kurtosis, the Hurst exponent and entropy.

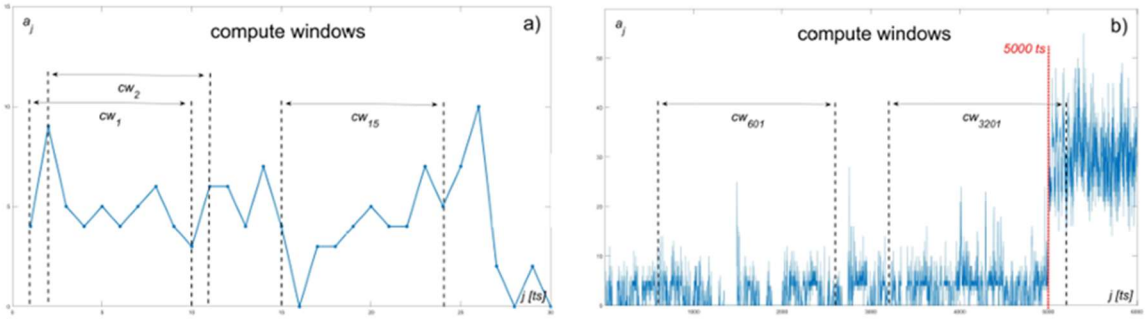


Fig. 3: Windows for calculating parameters: a) |cw| = 10 ts, b) |cw| = 2000 ts.

Double statistical coefficients require two compute windows, or random samples of two random variables, to determine their value. From the point of view of mathematical statistics, these are coefficients that in a certain way compare the probability distribution of two random variables. We used the correlation coefficient, the autoregressive coefficient and divergence.

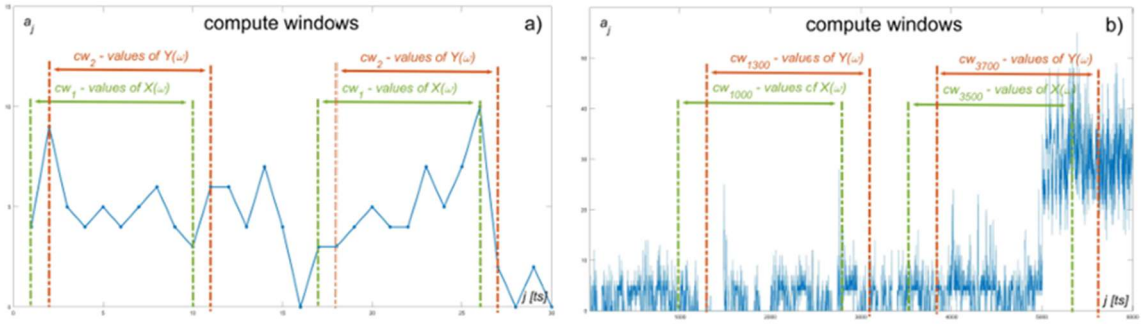


Fig. 4: Pair of windows for calculating parameters: a) |cw| = 10 ts, shift 1 ts, b) |cw| = 2800 ts, shift 300 ts.

We will present the response of selected statistical parameters on various records of DDoS attacks. We have omitted a number of experiments that we carried out on various simulated scenarios. In these scenarios, the selected statistical parameters worked "in an exemplary manner" and we gained an initial idea of the effectiveness of using the individual parameters in recognizing a change and increase in simulated traffic. When deployed to recognize a real attack, the responses of the statistical parameters were no longer so unambiguous. For the presentation, we selected eight different records of DDoS attacks.

The first four records represent standard DDoS attacks (N – normal), in which the parameters behave similarly to the simulated data. The next two records also represent standard attacks at first sight, but the parameters behave in a substantially different, indeterminate manner in them (S – special). The seventh record is a representative of so-called Low Rate (LR) DDoS attacks, i.e. attacks with a low average flow intensity. The last record is an attack which, in our opinion, cannot be detected by the above statistical parameters (P – as in problem). For the records, together with the number of packet increments a(t), we also displayed their moving average m(t) with a compute window cw = 1024 ts.

In the graphs of the DDoS attack records, we also displayed the course of the moving average and the time interval in which the moving average m(t) reached its local maximum from the start of the attack. We will use this interval in the subjective evaluation of the other statistical coefficients. In all DDoS attack records, we can observe a linear growth of the moving average. We are looking for coefficients whose values begin to grow faster than linearly at the start of an attack. More detailed information on the individual attacks can be found in the supporting document Machine Recognition of DDoS Attacks Using Statistical Parameters.pdf

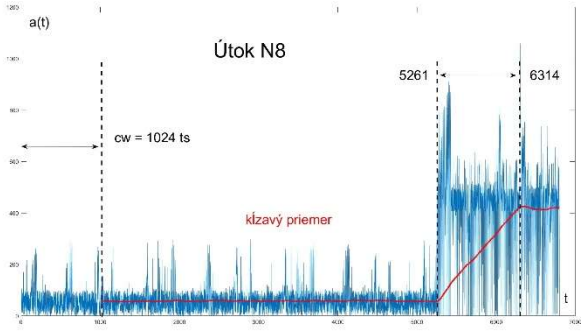


Fig. 5: Attack N8 and moving average

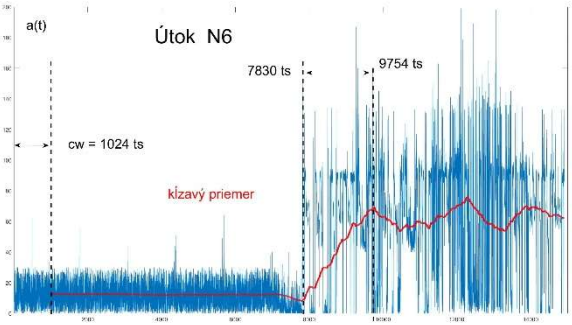


Fig. 6: Attack N6 and moving average

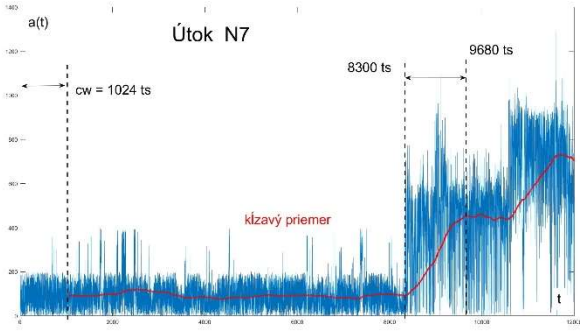


Fig. 7: Attack N7 and moving average

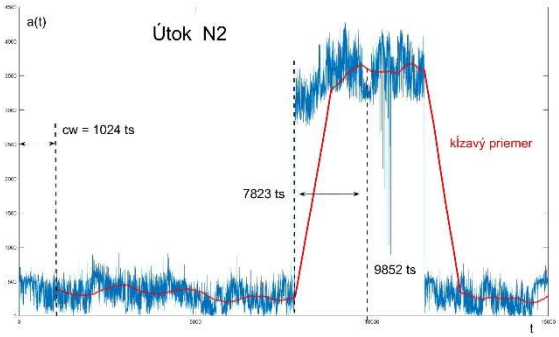


Fig. 8: Attack N2 and moving average

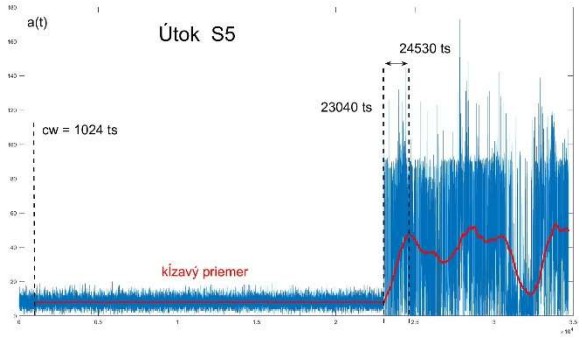


Fig. 9: Attack S5 and moving average

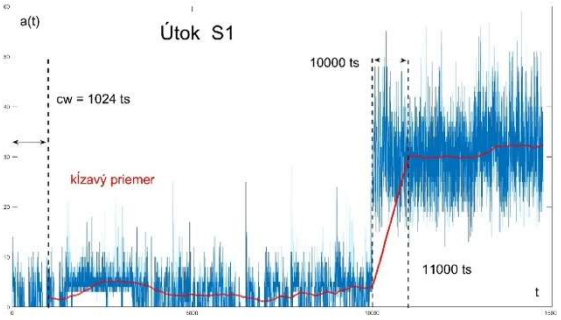


Fig. 10: Attack S1 and moving average

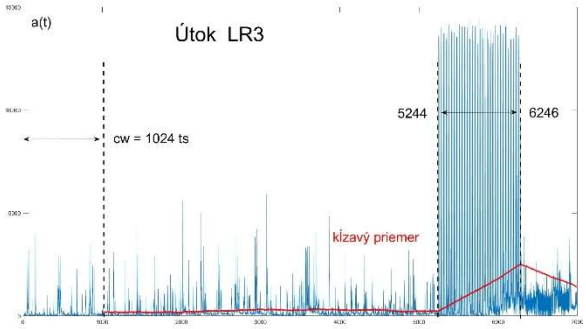


Fig. 11: Attack LR3 and moving average

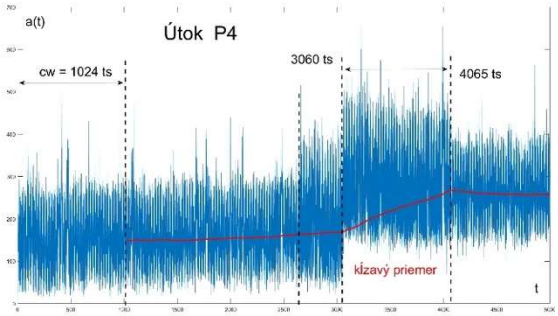


Fig. 12: Attack P4 and moving average

2.1.1.1 Statistical parameters

2.1.1.1.1 1-window parameters

A. Coefficient of variation

On the basis of our experience with modelling real IP traffic, we can assume that a normal standard IP flow, particularly in a backbone network or in cases where several IP services are aggregated, exhibits the character of a stationary stochastic process with

independent occurrence of packets, whose distribution represents a unimodal probability distribution of MMP (Markov modulated process) character. When a DDoS attack begins, such a "natural " probabilistic character of standard aggregated traffic is substantially disrupted. In most cases, there will of course be a marked increase in the average and peak intensity, but at the same time the natural data flow is disrupted by artificially generated packets, which in most cases causes a marked decrease in the dispersion of packets. This change is monitored by the coefficient of variability V:

$$\lambda_{avg} = \frac{1}{N}\sum_{i=1}^{N} a_i, \qquad S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(a_i - \lambda_{avg})^2, \qquad V = \frac{S}{\lambda_{avg}}$$

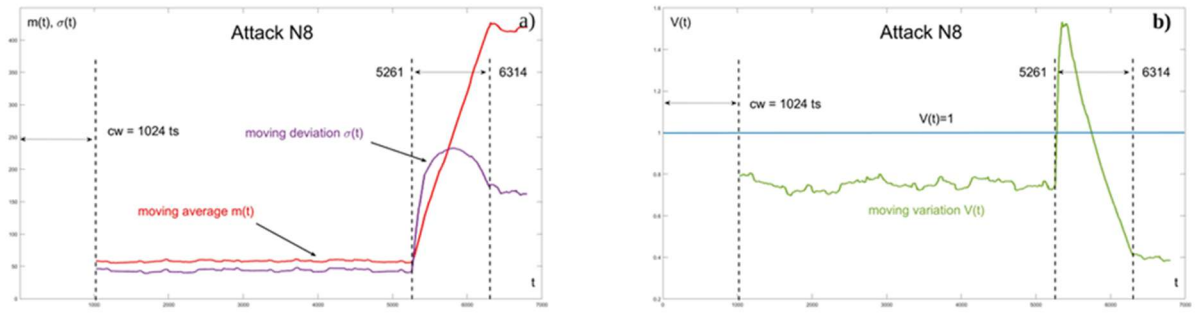


Fig. 13: Attack N8: a) Moving average and sample deviation, b) moving sample variation

For most of the observed records, approx. 75%, the coefficient of variability was able to recognize the onset of a DDoS attack in the first time slots by exceeding the value V=1. The coefficient had a problem with Low-rate attacks, where a threshold with the value 1 was not usable.

B. Skewness and kurtosis

Skewness and kurtosis (Curtosis) are further statistical coefficients that describe the shape of the probability distribution in a given compute window.

$$K = \frac{\frac{1}{N-1}\sum_{i=1}^{N}(a_i - \lambda_{avg})^3}{\left[\frac{1}{N-1}\sum_{i=1}^{N}(a_i - \lambda_{avg})^2\right]^{3/2}}, \qquad S_{kw} = \frac{\frac{1}{N-1}\sum_{i=1}^{N}(a_i - \lambda_{avg})^4}{\left[\frac{1}{N-1}\sum_{i=1}^{N}(a_i - \lambda_{avg})^2\right]^{2}}$$

When even a small amount of attack IP traffic is loaded, we assume a marked change in the unimodal symmetry of the packet distribution in the given compute window.

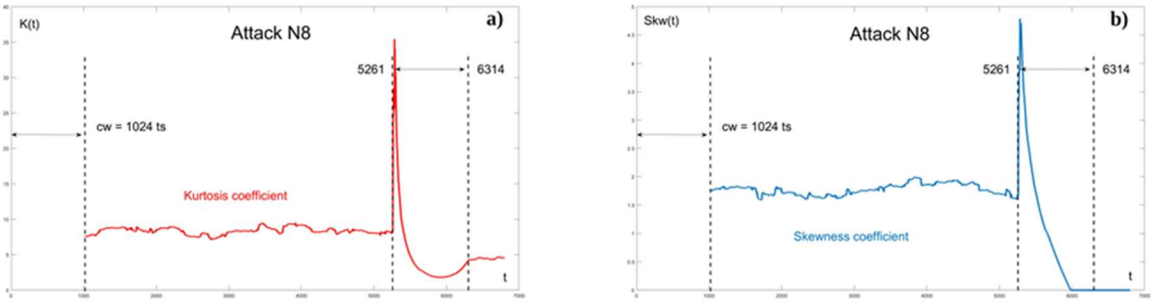


Fig. 14: Attack N8: a) Moving kurtosis coefficient, b) Moving skewness coefficient.

The coefficients *K* and *Skw* are estimates of the third and fourth central moments of the random variable *X*; therefore, the course of their values is very similar. In most of the analysed records, they responded to the start of a DDoS attack with a several-fold increase (peak) in values compared with the preceding course; however, unlike the coefficient of variability, they did not provide any threshold value for detection, so further detection methods will need to be used to recognize a marked peak at the moment of the attack.

C. Hurst exponent

When testing the responses of the Hurst exponent to simulated DDoS scenarios and to the first real recorded data, we obtained interesting results. The exponent tended to "jump" to values close to H = 1 at the start of an attack. At the same time, during the processing of standard traffic, it remained in the range 0.4 to 0.6, which corresponds to the values of a stationary random process. This suggested the possibility of using the Hurst exponent for automatic attack recognition if a value close to H = 1 is exceeded.

The disadvantage of the Hurst exponent is that the coefficient cannot be calculated analytically; it can only be estimated statistically. Several methods are used for the estimation, the main ones including: R/S statistics, the aggregated variance method, the absolute value method, residual variance, Higuchi's method, the modified Allan variance, the window rescaling method, the Whittle estimator and others.

On the basis of our experience, we decided to use estimates by means of R/S analysis and DFA - Detrended Fluctuation Analysis. We do not present the individual estimation procedures because of their complexity – more information can be found in the supporting document Machine Recognition of DDoS Attacks Using Statistical Parameters.pdf. We also used a modified estimate of the Hurst exponent by means of R/S analysis, in which we first removed the linear autoregressive trend from the incremental values.

In the following figures, we present two cases in which, according to our subjective evaluation, the response proved to be very unsatisfactory and completely ideal, respectively, from the point of view of automatic recognition. When processing record LR3, none of the Hurst exponent estimates responded significantly to the start of the DDoS attack. In the case of N8, all three estimates responded markedly, with the RS estimates exceeding the value H = 1.

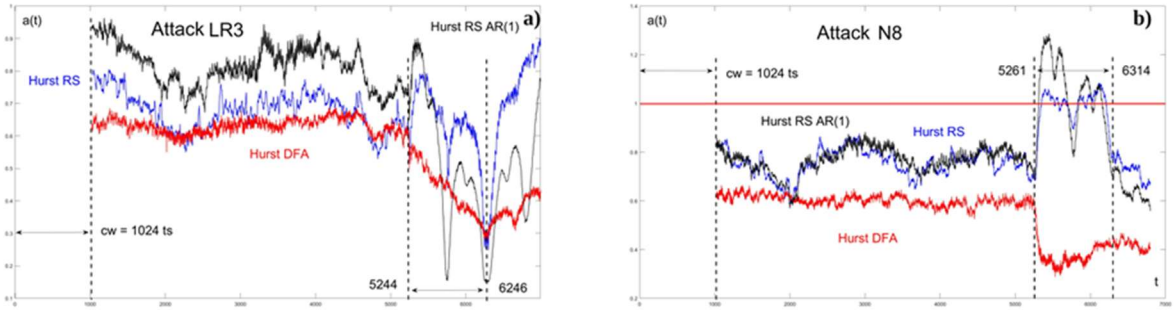


Fig. 15: Hurst exponent: a) Attack LR3, b) Attack N8

Overall, we can say that, with the DFA estimate, the values of the H-exponent mostly decreased markedly at the moment of the attack. With the R/S estimate, the course of the values was often insignificant from the point of view of automatic recognition. However, after the removal of the linear autoregressive dependence, the values of the H-exponent increased markedly. In some cases, the values of the exponent at the moment of the attack even exceeded the value H = 1.

2.1.1.1.2 2-window parameters

A. Autoregressive and correlation coefficient

The autoregressive and correlation coefficients are very similar probabilistic characteristics that express a certain kind of dependence between two random variables, in our case between variables whose realizations represent the IP flow increments a_i in overlapping compute windows.

The correlation coefficient R represents a scaled covariance, using the standard deviation of the individual random variables. The autoregressive coefficient c represents the linear dependence in the autoregressive model AR(1).

$$c = \frac{\sum_{i=1}^{N-1} a_i^2}{\sum_{i=1}^{N-1} a_i a_{i+1}}, \qquad R = \frac{\sum_{i=1}^{N} (a_i - m_i)(a_{i+1} - m_{i+1})}{\left[\sum_{i=1}^{N} (a_i - m_i)^2\right]^{1/2} \left[\sum_{i=1}^{N} (a_{i+1} - m_{i+1})^2\right]^{1/2}}$$

The form of the calculation of the estimates of these two parameters implies their similar course. Again, we selected two extreme records – LR3 and N6 (Figure 16).

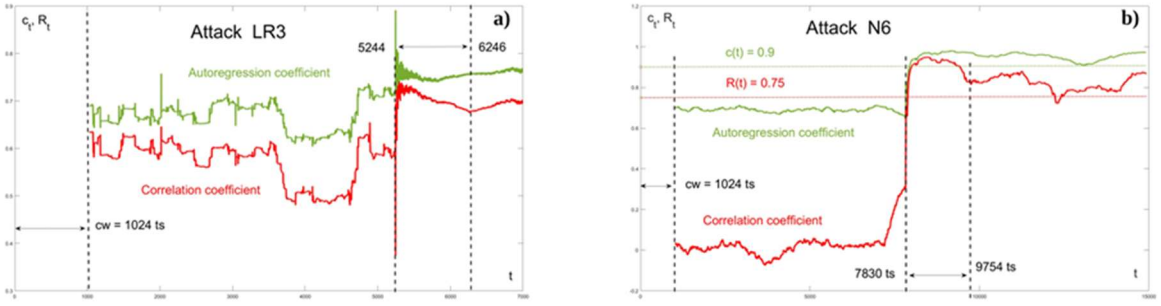


Fig. 16: Autoregressive and correlation coefficient: a) Attack LR3, b) Attack N6

For most records, at the moment of the onset of the DDoS attack, the values of the coefficients began to grow significantly towards the value 1. Thanks to this, it is possible to set a certain limit value (threshold) for both coefficients, the exceeding of which would signal a DDoS attack.

On the basis of the experiments performed, we set the thresholds:

- for the moving autoregressive coefficient: c(t) = 0.9
- for the moving correlation coefficient: ϱ(t) = 0.75

For most of the analysed records, except LR3 and P4, the thresholds set in this way can be used for automatic attack detection.

B. Kullback-Leibler divergence

Another statistical coefficient that compares the probability distribution in two overlapping compute windows is the Kullback–Leibler divergence (KD). Divergence determines how one probability distribution function (P) differs from another distribution function (Q). The following relations apply to the calculation of the divergence KD and its estimate dvg:

$$KD(P\|Q) = \sum_{k=0}^{m} P(k) \ln \frac{P(k)}{Q(k)}, \qquad dvg = \sum_{k=0}^{m} \left[\frac{n_k}{N}\right] \ln \left[\frac{n_k}{m_k}\right]$$

For most records, the divergence responded with a high impulse immediately at the onset of the attack. However, it tended to respond in the same way also to peaks in normal traffic, which

could cause many false alerts during detection. For example, in attack S1 the impulse during the attack is indistinguishable from the impulses during normal traffic. In attack LR3, the course of the divergence is completely ideal for automatic recognition.

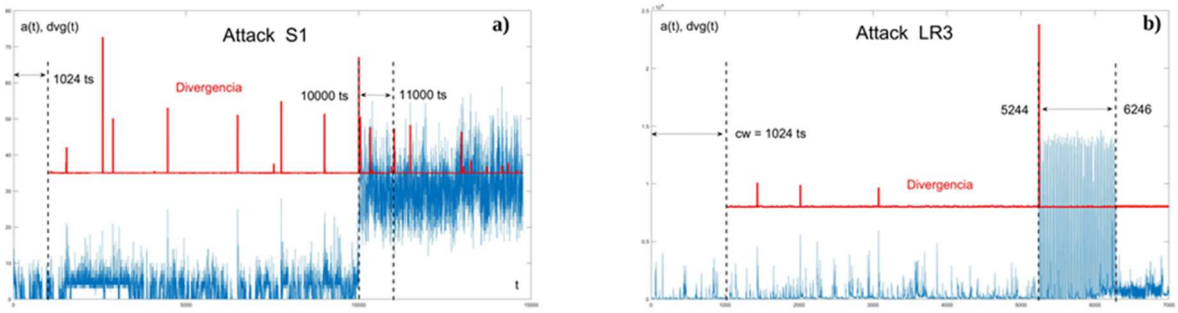


Fig. 17: Kullback-Leibler divergence: a) Attack S1, b) Attack LR3.

A further disadvantage of using divergence is the fact that the high impulse at the start of the attack lasts very briefly compared with other parameters, so we consider its use on its own to be problematic. We have therefore postponed the use of divergence to further research.

2.1.1.1.3 Effectiveness of using statistical coefficients

In the following table, we summarize the evaluation of the response of the investigated statistical coefficients to the analysed DDoS attack records. Of the three investigated estimates of the Hurst exponent, we selected the estimate by means of R/S analysis with removal of the linear autoregressive trend, which we consider the most effective.

Attack	$V(t)$ 1.00	$K(t)$	$Skw(t)$	$H(t)$ 1.00	$c(t)$ 0.90	$R(t)$ 0.75	$dvg(t)$
N2	0	PT	PT	9	PT	PT	4
N6	0	PT	PT	0	PT	PT	5
N7	0	x	x	1	PT	PT	4
N8	0	PT	PT	0	PT	PT	0
S1	5	x	x	6			8
S5	0	PT	PT	0	PT	PT	0
LR3	PT	PT	PT	x	PT	PT	3
P4	x	x	x	PT	x	x	x

Tab. 1: Effectiveness of statistical coefficients, coefficient of variance V(t), kurtosis K(t), skewness Skw(t), Hurst exponent H(t), autoregressive coefficient c(t), correlation coefficient R(t), Kullback-Leibler divergence dvg(t).

Green cells mean that the attack was recognized by exceeding the threshold values of the given parameters. The threshold values are given in the right-hand column of the table. The number in a cell indicates the number of false alerts and the number of times the limit value was exceeded before the actual attack. In the case of divergence, this number represents the number of impulses before the actual attack, or the number of false attacks. The designation "PT" means that we assume that the attack could be recognized using prediction methods (PT – predicting tunnel).

The designation "*x*" means that the given parameter is not usable for the given record.

On the basis of the experiments performed, we can divide the parameters into two groups:

1. First group – parameters for which we can use a threshold value for attack detection: V(t), H(t), c(t), R(t)

2. Second group – parameters for which prediction methods need to be used for attack detection: K(t), Skw(t), c(t) and R(t).

The autoregressive and correlation coefficients belong to both groups. Because of the short duration of its impulse during an attack and, at the same time, its frequent false responses to peaks in normal traffic, we have provisionally excluded divergence from further investigation.

2.1.1.2 3σ-Tunnel

The way to automatically recognize a sudden marked change in the course of the values of a given statistical parameter at the onset of a DDoS attack is based on the idea of creating a so-called prediction tunnel. Originally, we considered using polynomial regression, autoregressive analysis and the Fourier transform to predict new values of the given statistical coefficient, but in the end the simplest method proved its worth: the use of the arithmetic mean and the standard deviation. For a given statistical coefficient θ, we determine a prediction window of size |pw| = N. From the values θ_t of the parameter in this window, we calculate the mean value θ and the standard deviation σ_θ. Around the mean θ, we create the interval

$$I = [\theta - n\sigma_\theta,\ \theta + n\sigma_\theta]$$

We then test whether the new value of the parameter belongs to this prediction interval, i.e. whether θ_{N+1} ∈ I. If it does not belong, the system detects the start of an attack. In the opposite case, we shift the prediction window pw by one value θ and repeat the whole process:

$$\bar{\theta} = \frac{1}{N}\sum_{i=1}^{N}\theta_i, \qquad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(\theta_i - \bar{\theta})^2 \qquad \text{Test:} \quad \theta_{N+1} \in \langle \bar{\theta} - n\sigma, \bar{\theta} + n\sigma \rangle$$

On the basis of experiments with various tunnel widths and prediction window sizes, we can state that the 3σ-tunnel with |pw| = 1000 ts has a sufficiently "long memory" to be able to cover the local variability (jitter) of the statistical coefficient without causing a large number of false alerts. At the same time, a prediction tunnel set up in this way has a sufficiently "long memory" to be able to respond to a significant increase in the values of the given coefficient at the start of a DDoS attack.

In the following figures, we show the behaviour of the 3σ-tunnel on the parameter with the greatest variability among the statistical coefficients considered, namely the Hurst exponent (estimated by means of R/S statistics with removal of the autoregressive trend).

Figure 18a shows the ideal case, where detection occurred when the third time slot with attack traffic was loaded and no false alerts occurred. In Figure 18b, the attack was not detected and one false alert appeared.

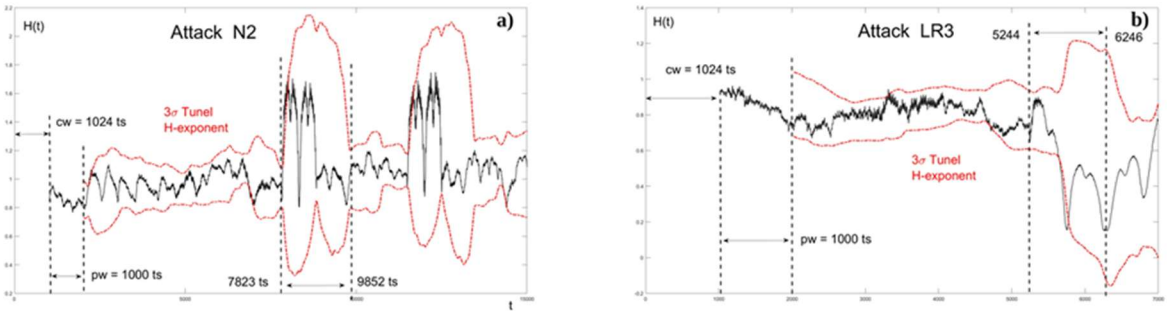


Fig. 18: 3σ-tunnel for the Hurst exponent (without RS): a) Attack N2, b) Attack LR3

Calculating the Hurst exponent is considerably more time-consuming than calculating the other statistical parameters; we have therefore temporarily postponed its use for detection.

In Table 2, we present the results of detection using the 3σ-tunnel applied to selected statistical coefficients. The symbol F/R represents the number of false alerts (F) and the attack detection time (R) in time slots (ts). For example, the first value in the table, 2/30, means that when the 3σ-tunnel applied to the coefficient of variation V(t) was used, the attack in record S1 was recognized within 30 ts of its start, with two false alerts having occurred beforehand.

Attack	$V(t)$	$K(t)$	$Skw(t)$	$c(t)$	$\varrho(t)$
S1	2/30	1/5	2/6	1/9	1/9
N2	1/116	5/24	2/47	0/1	1/1
LR3	0/1	2/1	1/1	1/1	1/8
S5	4/18	4/20	3/42	7/1394	6/5
N6	3/1	2/4	1/10	3/17	5/13
N7	1/51	2/40	2/40	1/90	2/76
N8	0/2	1/1	0/1	5/2	0/3

Tab. 2: Detection using the 3σ-tunnel applied to: variability V(t), skewness K(t), kurtosis Skw(t), autoregression c(t) and correlation ϱ(t).

We consider the parameter R (attack detection time) to be more important than the parameter F (number of false alerts), because the number of false alerts can be eliminated by a suitable combination of statistical coefficients. For this reason, we will use both parameters to compare the effectiveness of the individual coefficients. To visualize the similarity in a 3D view, we use the PCA (Principal Component Analysis) method.

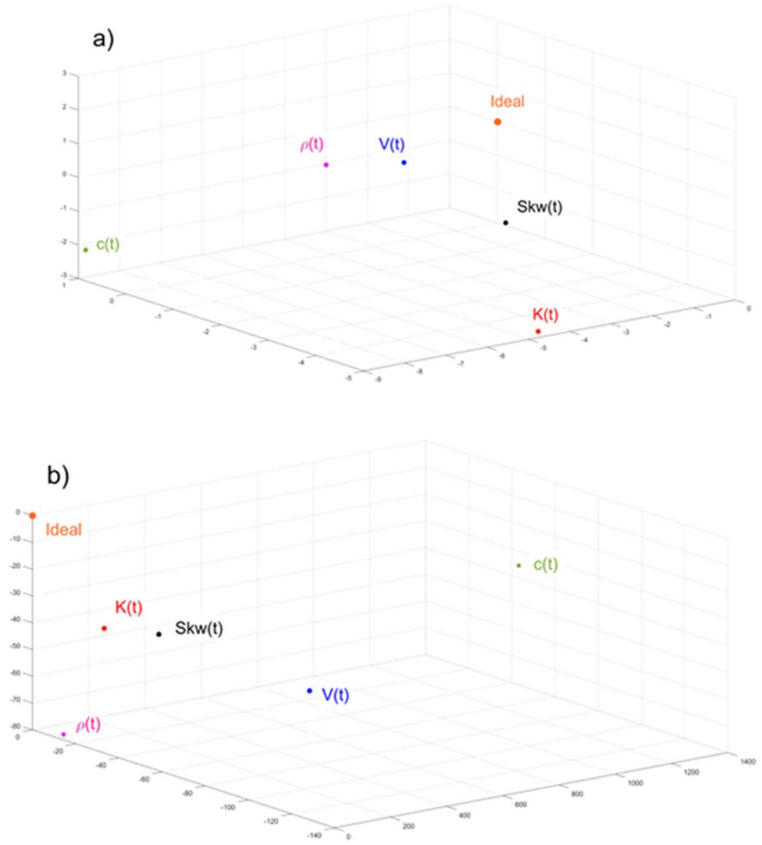


Fig. 19: 3D visualization of the parameter values from Tab. 2: a) F (False) – false alert, b) R (Recognize) – recognized attack

The effectiveness of the individual statistical coefficients is determined on the basis of the distance from the zero vector (see Table 3). For the parameter F (false alerts), the zero vector represents a zero number of false alerts for each traffic record, and for the parameter R (attack recognition) it represents recognition of the attack even before it starts.

Parameters	$V(t)$	$K(t)$	$Skw(t)$	$c(t)$	$\varrho(t)$
F (False)	0.0173	0.0026	0.0056	1.9517	0.0064
R (Recogn.)	28.05	54.92	19.07	85.35	78.92

Tab. 3: Effectiveness of using the coefficients in terms of false alerts and recognized attacks

In the case of false alerts, the order of the coefficients, from the most effective, is as follows:

K(t), Skw(t), ϱ(t)

The order according to the speed of attack detection is as follows:

Skw(t), V(t), K(t)

The autocorrelation coefficient c(t) achieved the worst results.

2.2 Conclusion

In the course of the project, we tested the possibilities of recognizing DDoS attacks using various statistical coefficients. We carried out the tests not only on standard attacks, approx. 75%, but also on problematic records where the coefficients responded surprisingly. In the report, we present both cases for comparison. Further records of the experiments are given in the supporting document Machine Recognition of DDoS Attacks Using Statistical Parameters.pdf.

When standard IP traffic turns into a DDoS attack, not only do the average and peak intensities change markedly, but the overall probabilistic structure of the flooding flow also changes. To capture this change, we used the presented statistical coefficients. These can be divided into two types: coefficients that exceeded a certain detection value at the onset of the attack, and coefficients that responded to the attack with a marked change in their values or with an impulse. Some coefficients even belong to both groups. On the basis of this division, we created the following methods, which are described above:

- The method of using detection limits (MDH)
- The method of using prediction tunnels (MPT)

For some coefficients, an upper limit can be determined, a certain threshold value, the exceeding of which signals the onset of an attack. This is the ideal case for machine recognition of an attack (V(t), H(t), c(t), R(t).). For other parameters, the onset of the attack was detected by a high impulse (K(t), Skw(t), c(t) and R(t).), which is evident to a human; however, the problem is to teach a machine to recognize this impulse and not to confuse it with a common peak value (impulse peak) in standard operation. For these purposes, we designed and tested a so-called prediction tunnel, which, with varying success for different parameters, enables machine recognition of the impulse at the onset of an attack while suppressing false alarms during common operational peaks.

The next step of the research will be the design and testing of detection functions that take into account the knowledge gained in testing the statistical parameters.

From the perspective of the SOCaaS architecture, these results have several practical implications:

- **DDoS detection must be multi-layered.** The combination of statistical parameters, threshold detection and prediction tunnels provides sufficient robustness for the initial detection of anomalies that may indicate an attack.
- **Statistical detection is lightweight and suitable for real-time monitoring.** Calculating the monitored parameters is not computationally demanding, which allows them to be deployed in real time even in a multi-tenant SOCaaS environment, where scaling requirements are critical.
- **These parameters are suitable as an "early warning" layer in front of ML models.** A SOC solution can use a combination of:
    - initial statistical detection ⇒ rapid identification of a sudden change,
    - subsequent ML/AI models ⇒ more accurate classification, reduction of false alarms.
      Statistical coefficients thus represent a cheap and fast trigger for deeper analysis.
- **They take multi-tenant behaviour into account.** Since individual tenant networks have a different normal, the prediction tunnel makes it possible to learn a "baseline" for each customer independently. This is an essential prerequisite for any centralized SOC solution.
- **Limitations for low-rate attacks remain.** Some attacks are deliberately designed to imitate legitimate traffic. In these cases, it is necessary to add further detection layers – flow analyses, behavioural models or event correlation.

Overall, it can be stated that the statistical parameters of IP traffic constitute a robust basis for the rapid detection of DoS/DDoS attacks in the SOCaaS ecosystem. Their combination with prediction methods provides a multi-level early warning system that is easily scalable, has low data processing requirements and enables reliable recognition even with high variability of network traffic. These findings are beneficial for the design of autonomous detection modules in a modern SOC, which need to respond quickly and reliably to changes in network behaviour without operator intervention.

3 DETECTION OF PHISHING E-MAILS AND SPOOFING

Phishing is a deceptive cyber tactic used by attackers to obtain sensitive information from unsuspecting victims. This method usually involves sending fraudulent e-mails or messages that imitate legitimate sources such as banks, government agencies or well-known companies. The aim is to trick recipients into revealing personal data such as login credentials, credit card numbers or other sensitive data such as a social security number.

Phishing is a typical example of a social engineering attack. Social engineering attacks are various techniques that exploit human psychology for malicious purposes. E-mail phishing is perhaps the best-known example; it involves attackers sending e-mails that appear to come from trusted sources in order to deceive recipients and induce them to provide sensitive information or click on malicious links. These e-mails often imitate the appearance and language of legitimate correspondence from banks, service providers or even friends and colleagues, which makes them difficult to distinguish from genuine communications.

Phishing has long been among the most frequent and most successful cyberattacks. According to regular security reports (e.g. Verizon DBIR, ENISA Threat Landscape), phishing is every year one of the main initial compromise vectors and is behind the majority of incidents in which attackers gain initial access to accounts, systems or internal networks. The success rate of phishing campaigns is high mainly because attackers target the human factor, which is the weakest link in the security chain.

Phishing also extends to text messages, known as SMS phishing or ***smishing***. In these attacks, the perpetrator sends text messages that lure the recipient into clicking on a malicious link or providing personal data. These messages may appear to come from a trusted organization and often create a sense of urgency to react immediately to the received message. ***Vishing***, or voice phishing, is another variant in which attackers use phone calls to obtain sensitive information. In these situations, the attacker often poses as a legitimate representative of an organization in order to obtain information or a direct action from the victim of the attack.

A modern form of voice vishing is ***deepfake vishing***, in which the attacker uses artificial intelligence-based voice synthesis to imitate the voice of a specific person – for example a manager, colleague or customer. Thanks to advanced voice cloning models, a recording of just a few seconds is enough for the attacker to create a convincing fake voice. Such a call can persuade the victim to perform high-risk actions such as money transfers, changing passwords or providing sensitive data. Deepfake vishing is extremely dangerous because it outwardly appears to be a legitimate phone call and bypasses traditional technical detection mechanisms.

Impersonation and pretexting are sophisticated tactics used in social engineering attacks, in which the attacker pretends to have a certain authority or creates a convincing scenario to exploit the trust of the victim of the attack. By assuming the identity of an authoritative person such as a police officer, IT technician or company manager, attackers can manipulate the targets of attacks into revealing confidential information or performing actions that compromise security. Creating convincing scenarios requires detailed knowledge of the victim or their organization, which makes these attacks particularly insidious and effective.

Baiting (baiting = to lure, to entice) and ***tailgating*** (tailgate = rear gate) are other widespread social engineering methods. *Baiting* attacks involve luring victims of the attack with the promise of goods or services in order to steal their personal data or infect their systems with malicious software. This may take the form of downloading free software or access to exclusive content. *Tailgating*, on the other hand, is a tactic of physical security breach in which an unauthorized person gains access to a restricted area by following someone who is authorized. This attack exploits the social norm of politeness and the natural tendency to help

others, which allows attackers to penetrate secured environments.

Elicitation and manipulation are subtle but powerful social engineering techniques. These methods involve obtaining information through a seemingly innocent conversation in which the attacker engages the target victim in a discussion and discreetly steers them towards obtaining the desired information. Attackers often manipulate targets emotionally, exploiting human tendencies such as the desire to be liked or the fear of confrontation. Gaining trust through rapport and empathy is crucial in these interactions, as it lowers the defences of the target of the attack and makes them more susceptible to revealing sensitive information.

Each social engineering attack shows the complexity and diversity of the tactics that attackers use to exploit human psychology. They highlight the need for constant vigilance in both personal and work environments in order to prevent these manipulative techniques. Understanding the nature of these attacks is the first step towards developing effective strategies to deal with them, while emphasizing the importance of awareness and education in the ongoing fight against cyber threats.

To combat phishing, individuals and organizations must be vigilant and sceptical of unsolicited communications. Key preventive measures include not clicking on links or downloading attachments from unknown sources, verifying the authenticity of requests for personal data and using the anti-phishing tools available in most modern web browsers and e-mail services. Education and training in identifying phishing attempts are key to building the first line of defence against these cyberattacks.

3.1 Experiments and results

The aim of the pilot testing was to verify and compare the suitability of the K-Means and HDBSCAN methods for identifying phishing in data in the form of logs. The testing was to determine whether these methods are able to:

- identify recurring patterns and dense regions of logs
- reveal anomalies and outliers that may indicate phishing or suspicious events
- compare the differences in the approach of both algorithms, in particular the ability to detect outliers and the cluster structure
- provide a comprehensible interpretation of the clusters.

For the pilot testing, we worked with a set of 57,000 logs obtained from operation at FRI UNIZA, originally containing approximately 160 variables, of which 41 were identified as relevant for identifying phishing. Several of these variables were categorical, and it was therefore necessary to pre-process them using a combination of one-hot encoding and the hashing trick, which expanded the dataset to 2,009 variables suitable for clustering. Before clustering, all variables were standardized using StandardScaler to remove differences in value ranges. Subsequently, PCA dimensionality reduction to 50 components was applied, which enabled efficient work with the clustering algorithms while preserving the variability of the data. This step was necessary and time-consuming, and it is of fundamental importance for the preparation of the machine learning model. Logs originating from different systems contain different variables, and therefore, when creating the final model for identifying phishing, it will be a major challenge to develop a robust script capable of processing unexpected or new variables as well. A detailed description of the experiments can be found in the supporting document Analýza ML metód a algoritmov pre detekciu phishing.docx. Based on the experiments performed, the following conclusion can be drawn.

The *K-Means* algorithm is suitable if a fast and deterministic decomposition is required; however, it does not detect outliers and distributes points unevenly in data of uneven density.

The *HDBSCAN* algorithm is suitable for robust log analysis because it can detect both dense regions and outliers, which is a significant advantage when identifying phishing and other

anomalies in network logs. Unlike algorithms that assign every point to a cluster, HDBSCAN makes it possible to label atypical or outlying logs (outliers) that do not meet the density criteria required for a cluster to form. In this way, the algorithm effectively filters out noise and focuses on logs that represent repeatable and significant behavioural patterns.

In the pilot testing, **HDBSCAN** proved more suitable for capturing finer differences between logs that could be indicators of suspicious or non-standard events. The algorithm can divide the data into clusters according to the density of points while preserving the hierarchical structure, and makes it possible to identify smaller but significant groups of logs that could be "lost" in massive clusters when using classic K-Means.

The main advantage of HDBSCAN over K-Means is therefore its ability to:

- automatically determine the number of clusters according to data density
- robustly identify outliers that may indicate anomalies or phishing activities
- preserve fine differences between logs, which improves the interpretation and analytical value of the results.

Overall, HDBSCAN provides a flexible and robust approach to log analysis that takes into account the natural variability of the data and enables effective detection of suspicious events without the need for a predetermined number of clusters.

3.2 Description of the selected algorithm

HDBSCAN is a hierarchical clustering algorithm based on the well-known and proven-in-practice DBSCAN algorithm. DBSCAN is a density-based clustering algorithm that does not require a predetermined number of clusters. DBSCAN depends on two parameters: the minimum number of points required to form a cluster from a group of points, and ε (which determines the minimum distance between two points for them to be called neighbouring points).

Internally, the algorithm classifies each data point into one of the following three categories:

- core points – are those that have at least the minimum number of points in the neighbourhood defined by the radius ε
- border points – are points that are not core points but are located in the neighbouring region or cluster of a core point described above
- anomaly points – are points that are neither a core point nor reachable from one (i.e. not a border point either)

Briefly summarized, the clustering process works as follows:

1. Setting the values of the parameters, the minimum number of points and ε.
2. Randomly selecting a starting point (say A) and finding all points at a distance of ε or less from this point.
3. If the number of points meets the threshold for the minimum number of points, then A is a core point and a cluster can be formed around it. All points at a distance of ε or less are border points and are added to the cluster centred on point A.
4. Iteratively repeating these steps for each point. If a point (say B) added to the cluster turns out also to be a core point, we first create its own cluster and then merge it with the original cluster around A.

DBSCAN is thus able to identify high-density regions in the feature space and group points into clusters. A point that does not fall into a cluster is designated as anomalous or an outlier.

The DBSCAN algorithm expects 2 user-defined inputs: the density threshold ε and m_{pts}, which acts as a smoothing factor for the density estimate of individual objects. The original DBSCAN algorithm introduced the notion of so-called *core objects*, which are objects with at least m_{pts} neighbours within a defined distance ε. These neighbours form the neighbourhood $N_{\in}$ of the object ε. A cluster C is defined as a non-empty maximal subset of the input data set such that every pair of objects in C is density-connected. 2 core objects x_p and x_q are density-connected if they are directly or transitively reachable within the threshold ε, where direct reachability means that $x_p \in N_\varepsilon(x_q)$ and $x_q \in N_\varepsilon(x_p)$. In the case of the DBSCAN algorithm, the global density threshold is difficult to set if there are clusters of different density, because each cluster must exceed the specified density level. The HDBSCAN algorithm solves this problem by creating a hierarchy of clusters with different densities. To limit the depth of the hierarchy, an optional parameter m_{cl} was introduced as the minimum number of objects within a cluster. The literature recommends setting $m_{cl} = m_{pts}$ and keeping m_{pts} as the only user-defined parameter.

The HDBSCAN hierarchy is created by the following steps:

1. Creating a minimum spanning tree MST (Minimum Spanning Tree) of the mutual reachability graph, which consists of all objects as vertices and the mutual reachability distances for each pair of objects as edge weights. Mutual reachability is defined as:

   $d_{mreach}(x_p, x_q) = max\{d_{core}(x_p), d_{core}(x_q), d(x_p, x_q)\}$,

   where $d_{core}(x)$ is the distance of object x from its nearest m_{pts} neighbours and $d(x_p, x_q)$ is the distance between objects x_p and x_q.
2. The entire data set is contained in a cluster that forms the root of the hierarchy. Subsequent clusters and random or noise elements arise by removing edges from the MST in descending order of their weights. Removing an edge leads either to two new clusters with more objects than m_{cl} in each cluster, to a shrunken cluster with at least m_{cl} objects and random elements, or only to noise elements if the cluster disappears.
3. From the entire clustering hierarchy containing all splits, a compact hierarchy is created containing only the most important levels, namely when clusters appear for the first time or disappear completely.

   In the special case, setting the parameter $m_{pts} = 1$ leads to a single-linkage hierarchy.

Thanks to the different density thresholds represented by the hierarchy (the closer to the root of the tree, the less dense the density threshold), the user no longer has to specify the parameter ε. The original DBSCAN algorithm creates flat partitions whose clusters are maximal sets of connected points with a density of points in their neighbourhood above a given density threshold, set by the parameter ε. The result corresponds to the extraction of clusters from the HDBSCAN hierarchy at density level ε, which represents a horizontal cut through the hierarchy.

In order to extract clusters with variable densities (of different levels) from such a hierarchy, a stability measure S (Stability) was introduced. The stability S of a cluster measures how stable a given cluster is in the hierarchy. This depends on the density range in which the cluster exists, i.e. on the minimum density at which the cluster begins to exist and the maximum density at which the cluster either splits or disappears, as well as on the number of objects the cluster contains compared to its descendants and parent cluster. The stability values can be calculated while the hierarchy is being created.

The algorithm for extracting a flat partition from such a hierarchy is an iterative procedure and starts at the leaf level as the current clustering solution. For each parent–child pair in the current solution: If the parent C_p is more stable than its children, say C_{d1} and C_{d2}, i.e. $S_{Cp} > S_{Cd1} + S_{Cd2}$, C_{d1} and C_{d2} are replaced in the current solution by C_{d1} and C_{d2}. Otherwise, the parent C_p is assigned to C_{d1} and C_{d2} as a new parent cluster, while C_p is from now on practically ignored. This procedure is repeated until the root of the hierarchy is the parent of all

clusters in the current solution.

Extracting a clustering solution in this way is efficient because local decisions are made at each stage, i.e. comparing only parents and children (and no other clusters) is sufficient to obtain a globally optimal solution with respect to *S*, thereby returning the most stable set of clusters.

More information on the description of the above-mentioned clustering algorithms can be found in the supporting document Clustering.pdf.

3.3 Conclusion

This chapter focused on phishing attacks and related social engineering techniques, which have long been among the most frequent and most successful cyberattack vectors. The input data for the machine learning algorithms used consist of security logs obtained from SIEM tools, which aggregate and normalize events from mail servers, security gateways and network sensors. These logs contain structured features describing the technical, behavioural and reputational characteristics of e-mail communication, on the basis of which anomalies and suspicious behavioural patterns can be identified without the need for explicit data labelling. The analysis showed that phishing, including its modern forms such as smishing, vishing and deepfake vishing, represents a significant security threat, mainly due to the exploitation of the human factor and its ability to bypass traditional technical protection mechanisms.

In the practical part of the chapter, the possibilities of using machine learning methods to identify phishing in data in the form of security logs were verified. The pilot testing compared the K-Means and HDBSCAN clustering algorithms in order to assess their ability to identify recurring behavioural patterns, reveal anomalies and provide interpretable results suitable for analytical use in the SOC environment. The results of the experiments confirmed that while the K-Means algorithm enables fast and deterministic decomposition of the data, its use is limited by the assumption of uniform cluster density and its inability to explicitly identify outliers.

Conversely, in the pilot testing, the HDBSCAN algorithm proved to be a more suitable tool for analysing heterogeneous security logs. Its ability to work with clusters of different density, automatically determine their number and identify outliers is a significant advantage in detecting phishing and other non-standard events. These properties make it possible to capture more effectively subtle differences in behaviour that may indicate suspicious or malicious activities, while reducing noise in the data.

Based on the above findings, it can be stated that density-based clustering methods, represented by the HDBSCAN algorithm, constitute a promising approach to log analysis in the context of phishing and spoofing detection, especially in cases where precisely labelled training data are not available. This chapter thus creates a methodological basis for the further parts of the work, which focus on the application of machine learning in detecting other categories of security threats in the SOC environment.

4 SPAM/HAM CLASSIFICATION

Most security threats use e-mail as an attack vector. Since the volume of traffic transmitted in this way is particularly large, it is necessary to use automated detection procedures that employ machine learning (ML) and artificial intelligence (AI) algorithms. In order to fully understand the power of ML/AI algorithms applied to spam detection, it is first necessary to clarify the tasks on which the filtering of spam e-mail messages should be performed.

For a better understanding of the tasks performed by a spam filter, the following example can be given. Imagine that you are separating the e-mails we receive and categorizing them based on the presence or absence of specific keywords that occur in the text of the e-mails with a certain frequency. For this purpose, we could perhaps list all the messages we receive in the inbox in a table. But how should we proceed in classifying the received messages into the categories HAM (legitimate message) or SPAM?

As already mentioned, the number of occurrences of suspicious keywords in the text of e-mail messages will be searched for. Individual messages identified as spam will then be assigned a score based on the number of occurrences of the identified keywords. This score will further provide a reference for classifying subsequent e-mail messages.

A threshold value is determined that makes it possible to separate spam e-mail messages. If the calculated score exceeds the set threshold, the e-mail will automatically be classified as spam – otherwise the message will be accepted as legitimate and thus classified as ham. This threshold value (as well as the assigned scores) will be continuously determined to take into account a new series of spam messages that may be encountered in the future.

Even from the abstract description of the spam detection algorithm, some important properties that need to be kept in mind can be noticed: it is necessary to proceed with the identification of a certain number of suspicious keywords that make it possible to classify messages as potential spam e-mails, with each e-mail being assigned a score based on the number of occurrences of the identified keywords.

It is necessary to set a certain threshold value for the score assigned to individual e-mails, above which e-mails will automatically be classified as spam. It is also important to properly weigh the significance of the keywords present in the text of the e-mails in order to adequately represent the degree of probability that a message containing the given keywords is spam (the keywords could in essence even be harmless, but together they are more likely to represent junk mail).

It must be taken into account that spammers are well aware of the efforts to filter unwanted messages and will therefore try their best to adopt new strategies to deceive spam filters. This translates into a process of continuous and iterative learning, which can be implemented well using ML/AI algorithms.

From the above, it is clear that it is no coincidence that spam detection represents the first test in the introduction of ML/AI in the field of cybersecurity. The first spam detection solutions in practice employed static rules that used regular expressions to identify predefined patterns of suspicious words in the e-mail text.

These static rules quickly proved ineffective due to the ever-new deception strategies implemented by spammers to deceive anti-spam filters. It was therefore necessary to adopt a dynamic approach that would allow the spam filter to learn on the basis of the constant innovations introduced by spammers while also making use of the user's decisions in classifying their e-mails. In this way, it was possible to effectively cope with the explosive spread of the spam phenomenon.

4.1 Experiments and results

The experiments were conducted on the 2 most frequently used datasets in the field of automatic detection of SPAM e-mail messages – "SpamAssassin" and "SMS Spam", which represent the "gold standard" in this field. Further details on these databases as well as on the experiments themselves are given in the supporting document Analýza ML metód a algoritmov pre detekciu spam emailových správ.docx. In summary, the following can be stated. Based on the experiments performed, it can be concluded that the most suitable method for detecting spam e-mail messages in terms of efficiency is the XGBoost method. For the "SpamAssassin" dataset, it achieves a spam detection success rate of 99%. This value is almost identical for both the Precision and Recall metrics, so the detection of both false positive and false negative spam detections is very well covered. For this dataset, the SVM method also achieves almost identical results; however, the evaluation time in the case of the SVM algorithm is approximately 4x longer compared to XGBoost. Nevertheless, it is still a process of a few seconds, so both methods can be deployed for real-time applications. For the "SMS Spam" dataset, the spam detection success rate dropped significantly to ~78%. The best results were achieved by the Logistic regression method, although the difference between the XGBoost and Logistic regression methods is only tenths of a percent. In terms of the time required for data prediction, the SVM algorithm again requires the longest interval.

Based on the experiments performed, it can therefore be summarized that the most suitable method for detecting spam messages is the **XGBoost** algorithm.

4.2 Description of the selected algorithm

The **XGBoost** (Extreme Gradient Boosting) algorithm is one of the most efficient implementations of gradient-boosted decision trees and has been chosen as one of the best machine learning algorithms used in Kaggle competitions. The XGBoost algorithm, specifically designed to optimize memory usage and exploit hardware computing power, reduces execution time while increasing efficiency compared to many machine learning algorithms and even deep learning models. The main idea of boosting is to gradually build sub-trees from the original tree so that each subsequent tree reduces the errors (residuals) of the previous tree. In this way, the new sub-trees will update the previous residuals in order to reduce the error of the cost function.

Consider a data set expressed as follows:

$$D = \{(x_i, y_i) \text{ where } x_i \in R_n;\ y_i \in R)\},$$

where n represents the dimension of the features x_i and y_i is the corresponding response or class. For the i-th element of the data set, we define the predicted value $\hat{y}_i$ as follows:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F,$$

where f_k denotes an independent tree in the space of regression trees F and $f_k(x_i)$ expresses the predicted score for the i-th sample and the k-th tree. The objective function of the XGBoost algorithm, denoted L, is given as follows:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k).$$

By minimizing the objective function L, the regression tree functions f_k can be trained. The error function $l(y_i, \hat{y}_i)$ evaluates the difference between the prediction $\hat{y}_i$ and the actual value y_i. The

term Ω is used to prevent the problem of overfitting by penalizing the complexity of the model as follows:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

,

where *T* is the number of leaves in the tree, *w* represents the score (weight) of a leaf (e.g. the predicted value for samples that end up in this leaf), γ is a regularization factor expressing the cost of complexity, which discourages creating a tree with too many leaves, and λ is an L2 regularization term on the leaf weights, which tries to shrink them so that the model is more conservative. Direct minimization of the objective function is complex. XGBoost uses a second-order Taylor expansion to approximate it. This is a key reason for its performance, as it uses both the first and second derivatives (gradient and Hessian). Let $I_j = \{i|q(x_i) = j\}$ be the instance set of leaf *j* with a fixed structure $q(x)$. The optimal weights w_j of leaf *j* and the corresponding optimal value can be obtained by the following equations:

$$w_j^* = -\frac{g_j}{h_j + \lambda}$$

,

$$\mathcal{L}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \lambda T$$

,

where g_i and h_i are the first and second derivatives of the loss function *L*. The loss function *L* can be used as a quality score of the tree structure *q*. The smaller the score, the better the model. Since it is not possible to enumerate all tree structures, a *greedy* algorithm can solve the problem by starting from a single leaf and iteratively adding branches to the tree. Let I_R and I_L be the instance sets of the right and left nodes after the split. Assuming $I = I_R \cup I_L$, the loss reduction after the split is then expressed as follows:

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma.$$

This relation is usually used in practice to evaluate split candidates. The XGBoost model uses many simple trees and the scores of leaf nodes during splitting. The first three terms of the equation represent the scores of the left, right and original leaf. In addition, the term γ represents the regularization on the additional leaf and will be used in the training process.

4.3 Conclusion

This chapter focused on the automatic classification of spam and ham e-mail messages, which is one of the longest-researched and most widely deployed in practice areas of machine learning use in cybersecurity. The analysis pointed out the historical development of spam filters from simple rule-based and heuristic approaches to modern machine learning methods capable of adaptively responding to the changing strategies of junk mail senders.

The experimental part of the chapter focused on comparing selected classification algorithms on the SpamAssassin and SMS Spam reference datasets, which represent an established standard in evaluating the effectiveness of spam filters. The results achieved confirmed that ensemble learning-based algorithms, above all XGBoost, provide very high classification accuracy while maintaining favourable evaluation time requirements. The high values of the

Precision and Recall metrics indicate that the chosen approach is suitable not only for detecting junk mail but also for minimizing false positive and false negative classifications, which have a fundamental impact on the usability of the system in real operation.

The comparison of results on different datasets also showed that the effectiveness of classification models is significantly influenced by the nature of the input data, their structure and quality. The drop in accuracy when processing the SMS Spam dataset points to the need to adapt models to the specific type of communication and highlights the importance of the correct choice of features and training data. For this reason, it is essential that classification models in a production environment undergo continuous updating and adaptation to new forms of spam campaigns.

Based on the experiments performed, it can be stated that the XGBoost algorithm ***represents a suitable solution*** for spam/ham classification in the context of security systems, mainly thanks to its robustness, high accuracy and ability to work efficiently with large and heterogeneous data. This chapter thus provides a solid methodological basis for the use of machine learning classification algorithms in protecting e-mail communication and creates a starting point for further chapters dealing with the detection of more complex and sophisticated security threats.

5 DETECTION OF ADVANCED APT TECHNIQUES

An advanced persistent threat APT (Advanced Persistent Threat) is a form of cyber threat posed by well-funded organizations, often with the aim of obtaining key information from the target organization. An APT threat differs from conventional and multi-stage attacks and is defined by a combination of three words:

- Advanced – An advanced threat actor is technically proficient, experienced and capable of discovering or introducing new vulnerabilities into a system and developing new ways to exploit these vulnerabilities. Attackers even resort to various attacks such as zero-day exploits, planted insiders or supply chain exploitation when the target's defences require it.
- Persistent – A persistent threat is anyone who shows determination to attack until the objective is achieved, is skilled and experienced in adapting to hostile territory in order to remain undetected, and has the resources to develop their TTPs to defeat defensive strategic or tactical countermeasures. These resources include various assets and are not specifically limited to financial resources. Human capital, for example, is a highly relevant resource. A persistent threat will have access to personnel capacity for every operation it initiates. They work predominantly in teams, as the scope and scale of their activities are sufficiently extensive. Other assets are infrastructure, time and financial resources.
- Threat – Threat refers to individuals (e.g. persons, groups and organizations) who have the motivation, capability and opportunity to engage in malicious acts. The key defining feature of a threat is its malicious intent, which is the driving force behind its motivation.

When considering APTs, it is important to realize that they are people, as this term was specifically designed to distinguish them from simple malware. As Richard Bejtlich states: "Threat means that the adversary is not a piece of mindless code." Technically, there is no way to capture human creativity or imagination. Therefore, we cannot assume that we can predict what attack an APT will carry out. Research often regards APT attacks as targeted multi-stage attacks. However, the restriction that a multi-stage attack must consist of at least two atomic or minor events is not mandatory for an APT attack. If the attacker needs only one action to achieve the objective, they will use it. The path of least resistance, as part of the term Advanced in the definition, describes their ability to exploit the full spectrum of intrusions. For example, if there is a known vulnerability with corresponding exploit material, an APT can and will use it to achieve its objectives.

A multi-stage attack may also hit one specific target. Detection systems therefore have difficulty distinguishing the threats behind an attack.

APTs try to remain undetected and therefore use concealment techniques, one of the most frequently used being the living-off-the-land technique. These techniques involve disguising malicious activity as seemingly harmless programs that use tools already present on users' computers. These tools are often part of the operating system or other software installed by the user. This type of attack is particularly difficult to detect because attackers usually do not create new malicious files on the target computers, and common antivirus scans may not identify them.

These techniques are not taken into account in conventional cyber threat intelligence CTI (Cyber Threat Iintelligence), because they require a higher level of sophistication from the attacker and cannot be compared with known procedures. This fact highlights the importance of taking these advanced tactics into account in security measures and of continuously updating CTI in order to stay one step ahead of the changing threat landscape.

Nevertheless, an APT can and will use known attack patterns, such as replay attacks or code injection, through known vulnerabilities. Established rule-based systems and data-driven approaches for specific TTPs are a partial solution.

In its worst manifestation, an APT can be described as a complex adversary with a broad spectrum of attacks, characterized by the DoD (Department of Defense) and sometimes called integrated threats. This characteristic means that they do not look for opportunities to strike – they use political, military, intelligence or economic resources to enable their operations.

Historical examples of APT attacks that are particularly interesting are the Stuxnet attack, which was used to destroy uranium enrichment centrifuges in Natanz in Iran, Industroyer, which caused a power outage in Ukraine in 2016, and TRITON, which attacked safety instrumented systems in the oil industry. Among other APT attacks, a significant pillar in the evolution of APTs is the shift from predominantly espionage to destructive attacks.

In addition to stealing information and manipulating or even destroying targets, an APT may also seek to install backdoor software in order to gain easy access at strategically advantageous times. Apart from the use of various tactics, techniques and procedures TTP (Tactics, Techniques, and Procedures), types of attacks also differ according to the attacker's domains, target, purpose and approach.

5.1 Experiments and results

Based on the experiments performed, it can be concluded that the most suitable method for detecting advanced APT attacks is the Random forest method. On the DAPT2020 dataset, which is one of the most comprehensive and most frequently used data sets for detecting APT attacks, we achieved a detection success rate of 98%. This value is almost identical for both the Precision and Recall metrics, so the detection of both false positive and false negative detections is very well covered. The second most successful method, with a detection success rate of 92%, is the SVM method, while the logistic regression method achieves a comparable detection success rate of ~90%. In terms of the time required for testing the data, the Random forest and logistic regression methods achieve very comparable results at the level of tens of milliseconds, so both methods can be deployed for real-time applications. The SVM method requires a significantly longer detection time during testing, at the level of tens of seconds, which represents more than a 100-fold longer time. Further details on the experiments themselves are given in the supporting document Analýza ML metód a algoritmov pre detekciu APT.docx.

Based on the experiments performed, it can therefore be summarized that the most suitable method for detecting advanced APT attacks is the **Random Forest** method.

5.2 Description of the selected algorithm

A decision tree is a non-parametric algorithm that is used for both classification and regression tasks. It has a hierarchical tree structure consisting of a root node, branches, internal nodes and leaf nodes. As can be seen from the diagram below, a decision tree starts with a root node, which has no incoming branches.

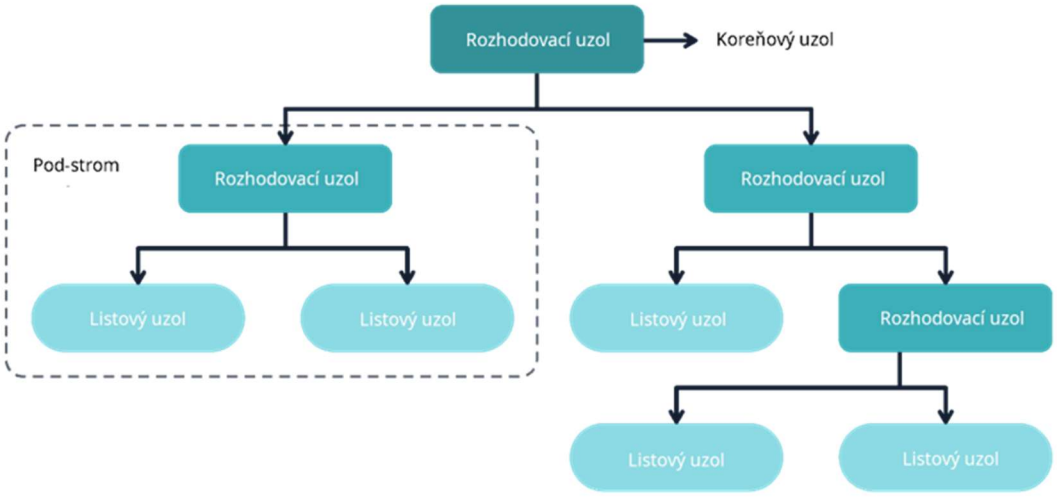


Fig. 20: Decision tree

The outgoing branches from the root node then feed into the internal nodes, also known as decision nodes. Based on the available features, both types of nodes perform evaluations in order to form homogeneous subsets, which are denoted by leaf nodes or terminal nodes. The leaf nodes represent all possible outcomes within the dataset. As an example, imagine that we are trying to assess whether or not we should go and play golf; we can use the following decision rules to make the choice:

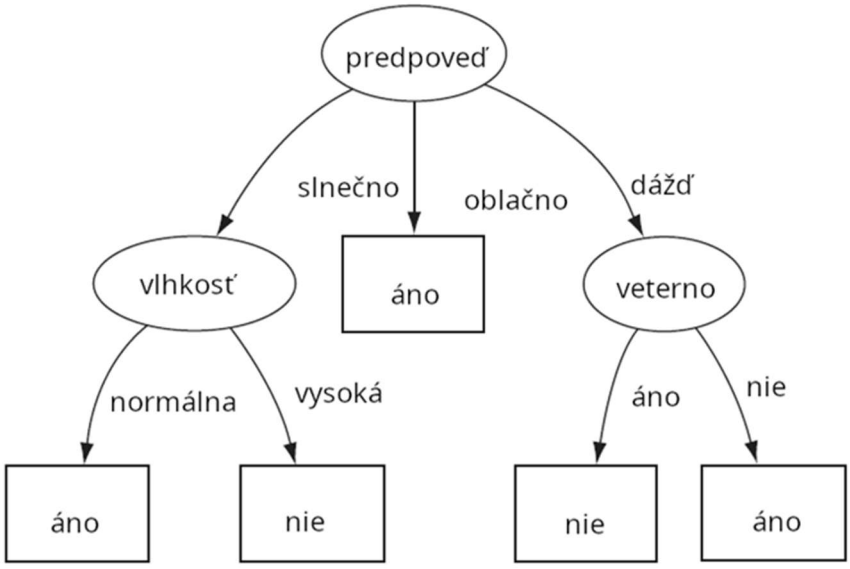


Fig. 21: Decision tree for the Golf dataset

This type of flowchart structure also creates an easily digestible representation of decision-making, which allows different groups across the organization to better understand why a decision was made. Decision tree learning employs a divide-and-conquer strategy by performing a greedy search to identify the optimal split points in the tree.

This splitting process is then repeated in a top-down, recursive manner until all or most records have been classified under specific class labels. Whether all data points are classified as homogeneous sets depends largely on the complexity of the decision tree. Smaller trees are more easily able to attain pure leaf nodes – i.e. data points in a single class. However, as a tree grows,

it becomes increasingly difficult to maintain this purity, and it usually results in too little data falling within a given sub-tree. When this occurs, it is known as data fragmentation, and it can often lead to overfitting. As a result, decision trees prefer small trees, which is consistent with the principle of parsimony according to the “Occam's razor” principle, meaning that "entities should not be multiplied beyond necessity". In other words, decision trees should increase their complexity only when necessary, as the simplest explanation is often the best. To reduce complexity and prevent overfitting, pruning is usually used; this is a process that removes branches that split on features of low importance. The fit of the model can then be evaluated through the process of cross-validation.

Random forest is an ensemble learning method that is used for either classification or regression tasks. Random forests consist of multiple decision trees that are combined into a unanimous decision or classification (figure below).

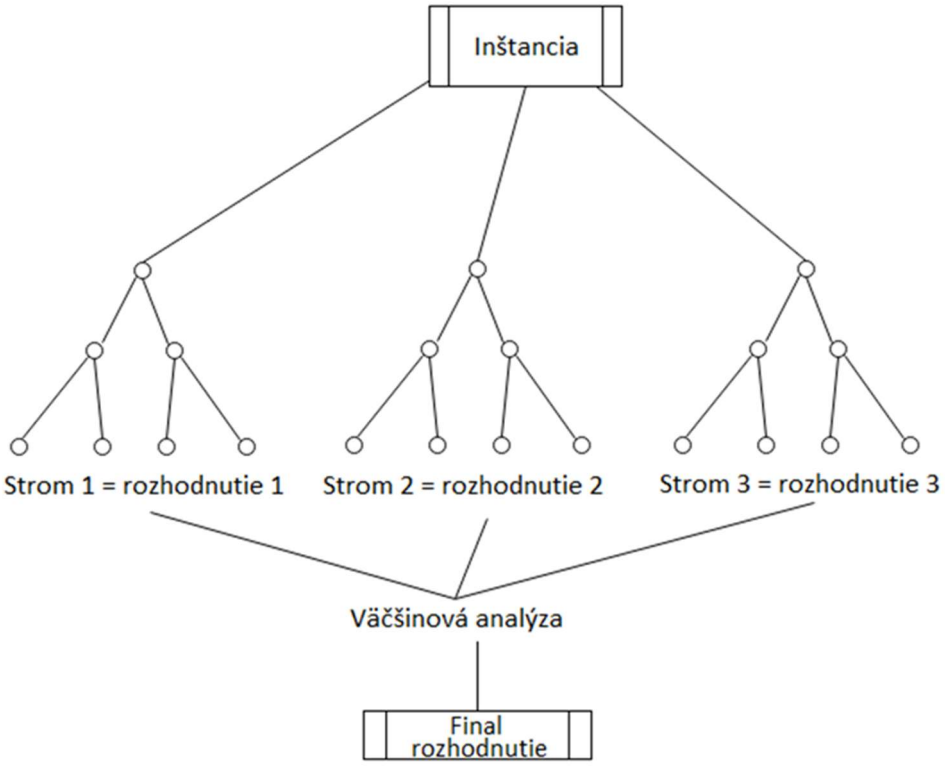


Fig. 22: Random forest – principle of operation

Random forests are better than ordinary decision trees because they do not cause overfitting of the data and subsequent overtraining of the model. This is achieved through the use of methods that employ random subspaces (for example *bagging*) to reduce the correlation between individual trees. The elementary algorithms/trees are trained on different subsets of the feature space, which are also randomly selected. An ensemble of decision tree models using the random subspace method can be built using the following algorithm.

For N training objects and a number of features D , we proceed as follows:

1. The parameter L is chosen as the number of individual trees in the ensemble (*ensemble*).
2. For each individual l-th tree, $d_l < D$ is selected as the number of features for l. Typically, only one value is used for all trees.
3. For each tree, a training subset X_n is created using the bootstrap method.

The decision trees are created from the X_n samples as follows:

1. Random features d_l are selected from the source; the optimal split of the training set then restricts its search to them.
2. According to the given criterion, the best attribute is selected and a split in the tree is made according to it.
3. The tree is grown until no more than n_{min} objects remain in each leaf, until a certain tree height is reached, or until the training set is exhausted.

In order to apply the model ensemble to a new data object, it is necessary to combine the results of the individual models by majority voting or by combining *a posteriori* probabilities. An example of the final classifier is as follows:

$$a(x) = \frac{1}{L}\sum_{i=1}^{L} l_i(x)$$

The basic parameters of the *random forest* method include:

- number of trees
- number of features for the tree split selection
- maximum tree depth
- impurity criterion

5.3 Conclusion

This chapter dealt with the detection of advanced persistent threats (APTs), which represent one of the most complex and most demanding categories of cyberattacks. APT attacks are characterized by a high degree of sophistication, long-term persistence and the ability to adapt to the defence mechanisms of the target organization, often using a combination of technical vulnerabilities, legitimate tools and the human factor. These properties significantly reduce the effectiveness of traditional detection mechanisms based exclusively on signatures or static rules.

In the experimental part of the chapter, the possibilities of using machine learning algorithms to detect APT attacks were analysed on the basis of data from the DAPT2020 set, which is among the most extensive and most frequently used datasets in this field. A comparison of selected classification methods showed that the Random Forest algorithm achieves the best results in terms of detection accuracy, with the achieved values of the Precision and Recall metrics confirming its ability to reliably identify APT activities while maintaining a low rate of false alarms. Its computational efficiency, which enables its use in a near-real-time environment, is also a significant factor.

A comparison with other algorithms, such as SVM and logistic regression, pointed to a trade-off between accuracy and processing time. While SVM achieves a relatively high detection success rate, its significantly higher time requirements limit its practical usability in the operational SOC environment. Conversely, Random Forest provides a balanced ratio between detection quality, robustness against overtraining and computational demands, which makes it a suitable candidate for deployment in production security systems.

Based on the results achieved, it can be stated that methods represented by the Random Forest algorithm constitute an effective tool for supporting the detection of advanced APT techniques, especially in environments with a large volume of heterogeneous security data. This

chapter also points to the need to combine data-driven approaches with knowledge of attackers' tactics, techniques and procedures, since the detection of APT attacks requires a multi-layered approach that goes beyond the capabilities of individual detection mechanisms.

6 CLASSIFICATION AND IDENTIFICATION OF MALWARE PATTERNS

Malware, also known as malicious software, remains one of the main threats related to cybersecurity to this day. Attackers often install malicious software on the victim's system by tricking users into performing certain actions, for example clicking on a URL from which the malware is downloaded, opening e-mail attachments or downloading certain files. In order to gain a foothold in the victim's system, malware may also exploit the victim's vulnerabilities, such as bugs, open ports and plugins. Once downloaded, malware often spreads to other systems/networks.

For protection against malware, an important security function is to determine whether a software sample is malicious or not and, if a software sample is malicious, to determine its type. Malware detection techniques have therefore been continuously improved to cope with the rapid evolution of malware.

Many proposed approaches are based on ML-based classification techniques. Classification can be binary, i.e. dividing software samples into two classes, namely malicious and non-malicious (benign/normal), or multi-class, i.e. classifying a sample according to different malware categories such as backdoor, Trojan horse or virus.

The ability to correctly classify malware helps to gain insight into its behaviour and thus to defend against it better. However, malware detection is challenging due to the obfuscation and behaviour modification techniques used by malware authors.

There are several malware families, which usually differ according to their objective and approach:

- Adware
- Fileless malware
- Viruses
- Worms
- Trojan horses
- Botnets
- Ransomware
- Downloader

Malware analysis determines the impact, origin and objective of malware. Such analysis may have different goals, namely:

- Malware detection
- Malware similarity analysis
- Malware category detection: to detect a specific type of malware with respect to a given malware taxonomy.

Malware analysis is usually divided into static, dynamic or hybrid. Static analysis focuses on the executable code and structure of malware without executing it. Static analysis is useful for identifying malicious software samples and distinguishing them from benign ones by generating fingerprints of malicious samples. Fingerprints of malicious samples help to uniquely identify known malware. Features analysed during static analysis include strings, file hashes, DLL imports, opcodes, API calls and metadata associated with the PE (Portable Executable) file. However, static analysis is not very effective when malware authors use dynamically downloaded data, malware packing, obfuscation of sources and anti-disassembly methods.

Unlike static analysis, dynamic analysis monitors the behaviour of malware during its execution. It is usually performed in a controlled virtual environment, for example in a sandbox. The actions

performed by the malware are dynamically tracked and extended to monitoring traffic, API calls, registry modifications, memory analysis and system file modifications. Hybrid analysis combines static and dynamic analysis. Initially, the source code is analysed without executing it, and later the analysis is performed by executing the code in a controlled virtual environment. In other words, hybrid analysis deals with analysing malware signatures and improving the analysis by combining it with dynamic parameters. As far as detection is concerned, there are two main techniques, namely the signature-based technique and the behaviour-based technique. Signature-based detection is based on fingerprints that uniquely characterize different types of malware. It has a low number of false alarms, but it can be circumvented by modifications to the malware code. Its main limitation, however, lies in the case of unknown malware. Behaviour-based detection dynamically monitors the actions of malware and is effective in detecting unknown malware. However, it is prone to a high number of false positive results. In addition to signature-based and behaviour-based techniques, there are specification-based techniques that store certain rules specifying valid behaviour in order to distinguish it from malicious behaviour. The rules are based on extracted parameters or features and may be applicable to networks or systems depending on where the malicious software is detected. However, when malware uses code obfuscation techniques, the rules need to be updated frequently, which increases the complexity of systems monitoring malicious behaviour. To eliminate the shortcomings of existing techniques, machine learning-based techniques have been proposed to improve the detection and classification of malware and also to process big data generated by commercial threat intelligence feeds. More detailed information on malware, the individual experiments and the results can be found in the supporting document Analýza ML metód a algoritmov pre detekciu malware.docx.

6.1 Experiments and results

From the broader spectrum of malicious software, in our experiments we focused on malware of the **Trojan horse** type. The Trojan detection dataset was used for the experiments; it contains records of flows in network traffic, including records of benign and malicious traffic. More information can be found in the supporting document Analýza ML metód a algoritmov pre detekciu malware 2.docx. In summary, the following can be stated. Based on the experiments performed, it can be concluded that the most suitable methods for detecting “Trojan horse” malware from network traffic are the following:

- Random Forest: success rate 99.99%, evaluation time ~9 s
- AdaBoost: success rate 99.96%, evaluation time ~19 s
- DL FFNN: success rate 98.39%, evaluation time ~18 s

Of the above algorithms, Random Forest performed the calculations twice as fast. The DL FFNN (Deep Learning Feed Forward Neural Network) method has a longer model training time compared to the other methods, which is, however, a standard property of neural networks (especially deep networks). The fastest method was the Gaussian Naive Bayes classifier – however, it achieved the worst results (91.49%).

We consider the **Random Forest** algorithm to be the most suitable method for detecting Trojan horse malware by analysing network traffic.

6.2 Description of the selected algorithm

Given that the description of our selected algorithm can be found in section 5.2, we will not duplicate it here.

6.3 Conclusion

This chapter focused on the detection, classification and identification of malware, which, despite the long-standing existence of security mechanisms, remains one of the most

significant threats in the field of cybersecurity. The analysis pointed out the diversity of malware families, propagation techniques and masking methods that attackers use to bypass traditional protection mechanisms based on signatures and static rules.

In the theoretical part of the chapter, the basic approaches to malware analysis, namely static, dynamic and hybrid analysis, were compared together with their advantages and limitations. At the same time, it was emphasized that modern obfuscation and behaviour modification techniques significantly reduce the effectiveness of isolated analytical approaches, which creates room for the use of data-driven machine learning methods capable of working with large volumes of heterogeneous data.

The experimental part of the chapter focused on the detection of Trojan horse malware based on network traffic analysis. The results of the experiments showed that ensemble learning algorithms achieve a very high detection success rate, with the Random Forest algorithm showing the best ratio between classification accuracy and processing time. The success rates achieved confirm the ability of this approach to effectively distinguish between malicious and legitimate traffic even in an environment with a high degree of data variability.

A comparison with other methods, such as AdaBoost and FFNN deep neural networks, pointed to trade-offs between accuracy, computational demands and practical usability in a real environment. While deep neural networks provide high accuracy, their higher requirements for training and model interpretation may represent a limitation in operational security systems. In this context, Random Forest proved to be a robust and well-interpretable solution suitable for deployment in the SOC environment.

Based on the results achieved, it can be stated that machine learning classification algorithms, especially ensemble learning methods, constitute an effective tool for supporting the detection and identification of malware at the network traffic level. This chapter thus complements the previous parts of the work with another important aspect of the use of ML in cybersecurity and provides a methodological basis for integrating malware detection mechanisms into a comprehensive SOC solution.

Funded by the European Union
NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

7 CONCLUSION – PROPOSAL

This document analyses selected categories of cyber threats and, on the basis of this analysis, proposes practically feasible solutions using machine learning and artificial intelligence algorithms in the security operations centre (SOC) environment. The document focuses on threats that represent a significant burden for SOC analysts in real operation, in particular in terms of data volume, processing time and the need for a rapid and accurate response.

On the basis of theoretical analysis and experimental verification, the use of several classes of ML algorithms suitable for different phases of security incident processing was proposed, since using only one is impossible. In the area of detecting phishing attacks and social engineering, unsupervised learning methods proved promising, above all the HDBSCAN algorithm, which is capable of identifying non-standard and unknown behavioural patterns in security logs. For mass classification tasks, such as distinguishing spam/ham communication, XGBoost-type classification algorithms were proposed, which provide high accuracy while maintaining low time requirements. In the case of detecting advanced APT techniques and malware in network traffic, ensemble learning methods, above all Random Forest, were identified as the most suitable, offering a balanced ratio between accuracy, robustness and interpretability of results.

The proposed solutions are conceived as an analytical layer integrated into the existing SOC workflow, building on traditional security mechanisms such as SIEM, IDS/IPS or EDR systems. In this context, machine learning algorithms do not provide autonomous decision-making but serve to support the detection, triage and prioritization of security events, thereby significantly reducing the manual workload of analysts and shortening the time needed to identify relevant incidents.

The result of the proposed approach is an ***AI/ML-powered SOC***, in which data-driven analytical methods are used for the systematic processing of large volumes of security data and to support the decision-making of security teams. The document also shows that the effective deployment of ML in a SOC requires a combination of technical measures, high-quality input data and clearly defined processes, with the human factor remaining an inseparable part of security operations.

The proposed solutions create a methodological and technological basis for the further development of the SOC towards a higher level of automation and analytical maturity, while maintaining control over decision-making processes and respecting the practical constraints of real operation. This approach enables the gradual transformation of the SOC into a modern, data-oriented and AI/ML-supported security environment capable of effectively responding to current and future cyber threats.

ANNEXES

File: V7 Príloha (KPB3) – Analýza ML metód a algoritmov pre detekciu spam emailových správ.docx

Supporting documents:

Analýza ML metód a algoritmov pre detekciu APT.docx

Analýza ML metód a algoritmov pre detekciu malware 2.docx

Analýza ML metód a algoritmov pre detekciu phishing.docx

Analýza ML metód a algoritmov pre detekciu sieťových anomálií.docx

Analýza ML metód a algoritmov pre detekciu spam.docx

Clustering.pdf

DAPT_at_MLHat2020.pdf

DECISION TREES - CLASSIFICATION AND REGRESSION TREES - CART.pdf

Ensamble metody obecně Bagging - náhodné lesy - Boost.pdf

KREDITNÝ SKÓRING POMOCOU NÁHODNÝCH LESOV.pdf

Machine Recognition of DDoS Attacks Using Statistical Parameters.pdf

podklad pre V7 v rámci KPB3_Návrh ML algoritmov.docx