PROJECT

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

V7 result of the analysis of suitable algorithms

KPB3 – Design of ML algorithms

type – documentation

submission month – M10

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Contents

1 Categorization of attacks in an IP network 4
1.1 Network layer attacks 4
1.2 Transport layer attacks (OSI layer 4) 5
1.3 Application layer attacks 5
2 Analysis of DoS and DDoS attacks 7
2.1 DoS (Denial of Service) 7
2.2 DDoS (Distributed Denial of Service) 9
3 Attack tactics of IP attacks 10
3.1 Exploitation 10
3.2 Lateral movement 10
3.3 Privilege escalation 11
4 Machine learning methods and algorithms 13
4.1 Classification of algorithms and methods by type of learning 13
4.1.1 Supervised learning 13
4.1.1.1 Classification 13
4.1.1.2 Regression 14
4.1.1.3 K-Nearest Neighbor 16
4.1.1.4 Support Vector Machine 18
4.1.1.5 Decision Tree 19
4.1.2 Unsupervised learning 20
4.1.2.1 Clustering 21
4.1.2.2 K-means 22
4.1.2.3 Gaussian Mixture Model 23
4.1.2.4 DBSCAN 24
4.1.3 Dimensionality reduction 25
4.1.4 Anomaly detection 26
4.1.5 Association Rule-Mining 27
4.1.6 Semi-supervised learning 28
4.1.7 Reinforcement learning (reward-based learning) 28
4.2 Classification of algorithms and methods by type of attack 29
4.2.1 Network intrusion detection (Network Intrusion Detection (NIDS)) 29
4.2.2 Malware detection 30
4.2.3 Spam/Ham & Phishing detection 30
4.2.4 Spoofing detection 31
4.2.5 Botnet detection 31
4.3 Classification of algorithms and methods according to the use of a neural network 31
4.3.1 Classical machine learning algorithms for anomaly detection 32

4.3.1.1 Classical supervised, unsupervised and semi-supervised algorithms 33
4.3.1.2 Input data for algorithms – data sources and preparation 33
4.3.1.3 Algorithm parameters and the training process 34
4.3.1.4 Recommendations of classical ML methods for recognizing various attacks 34
4.3.1.5 Criteria for comparing algorithms 35
4.3.1.6 Recommended algorithm 35
4.3.2 Machine learning algorithms using neural networks.....35
4.3.2.1 Principle of deep learning algorithms 36
4.3.2.2 Principle of deep learning algorithms 36
4.3.2.3 Input data and their preparation 36
4.3.2.4 Algorithm parameters and training 36
4.3.2.5 Recommendations of neural network methods for attack detection..37
4.3.2.6 Criteria for comparing algorithms 37
4.3.2.7 Recommended algorithm 37
5 Analysis of statistical parameters on selected DDoS attacks 38
5.1 Description of the DDoS attack records used for testing the response of statistical coefficients 38
5.2 Use of statistical coefficients.....40
5.2.1 Variability, skewness and kurtosis.....40
5.2.2 Detection values for Autoregression and Correlation 43
5.2.3 Response of Divergence to a DDoS attack.....45
5.3 Use of the 3σ tunnel for recognizing DDOS attacks.....47
5.4 Detection functions for machine recognition of DDoS attacks.....49
5.5 Conclusion 51
Annexes 52

1 CATEGORIZATION OF ATTACKS IN AN IP NETWORK

Individual types of hacker attacks can be categorized according to various criteria. For example, according to the method of attack, into **passive attacks**, which attempt to obtain information without interfering with the communication (e.g. eavesdropping (sniffing), traffic analysis, etc.), and **active attacks**, which interfere with network transmission or modify data (e.g. MITM, DoS, spoofing, etc.). Network attacks can be further divided according to their various goals:

- **DoS/DDoS attacks –** The goal is to put a service out of operation by flooding it with requests.
- **Sniffing (eavesdropping)** – Collecting packets from the network in order to obtain sensitive information.
- **MITM (Man-in-the-Middle)** – The attacker places themselves between two communicating parties and eavesdrops on/modifies the communication.
- **Port scanning and reconnaissance attacks** – Scanning the network to find open ports and vulnerabilities.
- **Attacks on DNS** – E.g. DNS cache poisoning, DNS spoofing.

One of the most significant classifications of network attacks is by the part of the network infrastructure in which the attack took place. In the following overview we only briefly describe the individual types of attacks; some selected serious attacks will be discussed later in the following chapters.

1.1 Network layer attacks

Network layer attacks (Layer 3 in the OSI model) target the network infrastructure and protocols such as IP, ICMP, or the addressing and routing itself. The goals of network layer attacks are to disrupt service availability, bypass security measures such as firewalls, gain access to data by manipulating routing, and conceal the attacker's identity.

IP Spoofing

The attacker falsifies (spoofs) the sender's IP address. The goal is to hide their identity or to pose as a trusted source. It is used, for example, in **DoS/DDoS** attacks or **Man-in-the-Middle** attacks.

ICMP Flood/Ping Flood

The attacker floods the target with a large number of ICMP requests (ping). This results in overloading of the system or network.

Smurf Attack

A specific type of ICMP attack. The attacker sends an ICMP echo request to the broadcast address of a network with the victim's spoofed IP address. All replies are directed back to the victim, which overloads it.

Fragmentation Attacks

Manipulation of IP packet fragmentation. The attacker sends non-standard fragmented packets that the target system cannot reassemble correctly. This can lead to system overload or evasion of firewall detection.

Routing Table Poisoning

Abuse of routing protocols (e.g. RIP, BGP) to insert false information. Consequence: incorrect packet routing, redirection through the attacker's device (MITM), or disabling of the network.

TTL Expired Attacks

They exploit the TTL (Time To Live) setting in the IP packet header. They can be used for **network tracing** (e.g. traceroute), but also as part of more complex attacks on infrastructure.

1.2 Transport layer attacks (OSI layer 4)

The transport layer (Layer 4 in the OSI model) provides reliable or unreliable delivery of data between communication endpoints, mainly by means of the TCP (Transmission Control Protocol – reliable transmission) and UDP (User Datagram Protocol – unreliable transmission) protocols. Attacks at this layer focus on abusing the way these protocols work. The goals of the attacks are to disable services or overload systems (DoS, DDoS), take control of a session (session hijacking), bypass security mechanisms and obtain information about open ports and services

TCP SYN Flood

The best-known attack on TCP. The attacker sends a large number of requests to establish a TCP connection (SYN packets) but never completes it. The server keeps "half-open connections" open and eventually exhausts its memory or the number of available connections. The server stops responding to legitimate clients.

TCP RST Attack (Reset Attack)

The attacker sends spoofed TCP RST packets, thereby forcibly terminating connections between two parties. It can be used to disrupt communication between two systems (e.g. between a client and a server).

TCP Session Hijacking

The attacker takes over an already established TCP connection between two parties. If they can guess the TCP sequence numbers, they can "insert" themselves into the communication and take control of the session. It is used for Man-in-the-Middle attacks or data injection.

UDP Flood

The attacker sends a large number of UDP packets to random ports of the target. The target service usually responds with ICMP "Destination Unreachable". The excessive traffic loads the CPU and the network.

UDP-based Amplification Attacks

The attacker sends small requests to open services (e.g. DNS, NTP), setting the victim's IP as the sender. The service responds to the victim with a many times larger packet – so-called amplification occurs. It is used in DDoS attacks.

Port Scanning

Although it is not directly an attack, it is a form of reconnaissance. The attacker scans ports using tools such as Nmap to find out which services are running. It often precedes further attacks.

1.3 Application layer attacks

The application layer (Layer 7 in the OSI model) is closest to the user and includes various services such as web applications, e-mail, DNS, FTP, etc. Attacks at this layer directly target vulnerabilities of specific applications and services. They are often sophisticated,

targeted and difficult to detect because they resemble normal user activity. The goals of application layer attacks are data compromise (theft, modification, deletion), taking control of the system, obtaining credentials, bypassing authentication or authorization, and disabling the service (via an application-layer DDoS)

HTTP Flood

The attack is carried out in the form of DoS/DDoS. The attacker generates a large number of legitimate HTTP requests (e.g. GET/POST). The intention is to overwhelm the web server with computationally intensive requests. Unlike network attacks, it is harder to recognize that it is an attack, because the requests are valid.

SQL Injection

The attacker inserts malicious SQL code into input fields (e.g. the login name). If the application does not filter the input, the attacker can: read, modify or delete data from the database, and obtain administrator privileges.

Cross-Site Scripting (XSS)

The attacker inserts a malicious script (often JavaScript) into a web page, which is executed in the victim's browser

Cross-Site Request Forgery (CSRF)

The attacker manipulates the user into performing an action they did not intend (e.g. a money transfer) while they are logged in to another application. Example: a malicious link that "silently" sends a password change request on behalf of the victim.

Command Injection

Insertion of a system command into an application's input field. If the application runs shell commands with unvalidated input, the attacker can execute commands on the system.

Directory Traversal (Path Traversal)

The attacker attempts to access files outside the permitted directory.

DNS Amplification / DNS Hijacking

Attacks on application services such as DNS:

- Amplification – abuse of a DNS server for DDoS.
- Hijacking – manipulation of DNS records to redirect victims.

Email-based attacks (Phishing, Spoofing)

The attacker sends e-mails that impersonate trusted senders, for example in order to obtain login credentials, spread malware or gain access to company systems.

Attack type	Description	Attack examples
1. DoS/DDoS attacks	Overloading a service in order to prevent access by legitimate users.	SYN Flood, UDP Flood, HTTP Flood

2. Sniffing (eavesdropping)	Unauthorized interception of network communication.	Packet sniffing (e.g. Wireshark)
3. Spoofing	False identity in network communication (IP, MAC, DNS, e-mail).	IP Spoofing, ARP Spoofing, DNS Spoofing
4. Exploitation	Exploitation of a vulnerability in a system or application.	Buffer Overflow, SQL Injection, RCE
5. MITM (Man-in-the-Middle)	The attacker inserts themselves between the two communicating parties.	SSL Stripping, Session Hijacking
6. Malware attacks	Infecting a system with malicious software.	Trojan horse, ransomware, spyware
7. Port scanning	Scanning ports to discover open services.	Nmap, Zenmap
8. Social engineering	Manipulating the user to gain access or information.	Phishing, pretexting, baiting
9. ARP/DNS attacks	Exploitation of weaknesses in the ARP or DNS protocol.	ARP Poisoning, DNS Cache Poisoning
10. Privilege escalation	Obtaining higher privileges in the system.	Kernel exploit, misconfigured permissions

Table 1: Types of IP attacks

2 ANALYSIS OF DOS AND DDOS ATTACKS

DoS (*Denial of Service*) and DDoS (*Distributed Denial of Service*) techniques occur in various types of security compromise in an IP network and in incidents at various levels of the network (see the previous chapter).

2.1 DoS (Denial of Service)

The abbreviation DoS translates as an attack denying a service. This type of attack is used to overwhelm a server. In this case, the attacker uses a device with which they attempt

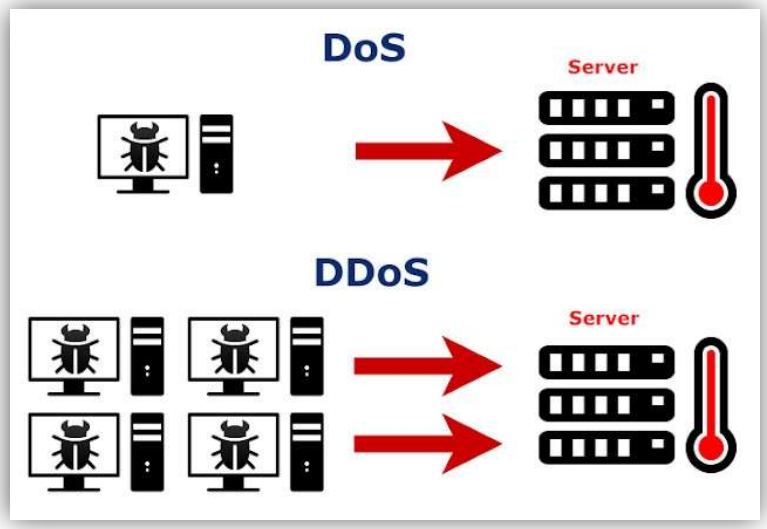
to disrupt the normal operation of the target server, service or network. They very quickly became a cybersecurity threat. They are simple, yet their impact can be devastating for individuals as well as companies.

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

- **Buffer overflow** (*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 overflows, and some of it may leak into other buffers. This then results in the corruption or overwriting of other data. Based on this principle, an attacker can corrupt files and modify or disclose various information.
- **ICMP attack** (*ICMP attack* or *Ping of Death*): This type of attack uses the ICMP protocol, exploiting ICMP echo requests, so-called pings. In general, echo requests and echo replies are used to ping a network device in order to verify connectivity between the device and the sender. If an attacker were to flood the target with request packets, the network must respond with the same number of packets, which makes the device unavailable for other incoming requests.
- **UDP attack** (*UDP attack*): The attack is based on sending UDP packets 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 packet on a port that is not used by any application, it informs the sender. If the attacker sends a sufficient number of packets to such a port, it can be overloaded.
- **SYN attack** (SYN flood): This is an attack that arises when a TCP connection is being created between 2 devices. TCP uses a three-way handshake (*3-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 reserved for TCP connections, and all further communication attempts will then be ignored.
- **Teardrop attack** (*Teardrop attack*): During this attack, the attacker attempts to send fragmented packets and the network then tries to compile this data back into its 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 versions of operating systems such as Linux or Windows contain a bug in reassembling a fragmented packet, the server can be attacked precisely because of this vulnerability.

2.2 DDoS (Distributed Denial of Service)

Unlike a simple DoS attack, a DDoS attack (*Distributed Denial of Service*) gains in strength because the attacker uses multiple systems at once, thereby increasing 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. These thus become part of a so-called botnet and, following the attacker's commands via a command-and-control server, these bots then attack the targeted server in unison. DDoS attacks are very difficult to detect because the attacking bots can be located in various places around the world. The main and key difference between a DoS and a DDoS attack is that in DDoS, using bots, the attacker can send huge volumes of



traffic.

Fig. 1: Difference between DoS and DDoS

DDoS attacks are divided into 3 main categories:

1. **Volumetric attack** (*Volumetric attack*): This type exploits vulnerable services such as NTP, DNS or SSDP. The attacker tries to flood the target with a large volume of packets to which the target tries to respond, until the targeted network infrastructure collapses. It is usually a high-bandwidth attack that can exceed 100 GB or even several TB per second and is immediately obvious to connectivity providers. Network monitoring, flow analysis and flow telemetry analysis have proved effective for detecting volumetric attacks.
2. **Protocol attack** (*Protocol attack*): This type of attack attempts to exhaust the resources of the server or its systems and relies on vulnerabilities of internet communication protocols. Protocol attacks include, for example: SYN attack, UDP attack, ICMP attack. These attacks are considered sufficiently crippling to the targeted infrastructure owing to the frequency of their duration. In 2018, attackers used the BGP protocol to redirect traffic intended for the “MyEtherWallet” service, which, however, was only a cover for the theft of cryptocurrency wallets. It lasted approximately 2 hours.
3. **Application layer attack** (*Application layer attack*): This attack targets a specific application and its vulnerabilities so that it is unable to process, send or receive data. The attacks are weak and slow and focus on GET and POST requests. The goal is then to cripple the web server, and it is measured in requests per second (*Requests per second - Rps*). They are low and weak so as to comply with the protocol.

There are several signs that can help us detect whether our network, application or entire infrastructure is under a DDoS attack. This can be a sudden increase in web activity on the server, reduced network performance or a sudden inability to communicate with the application.

3 ATTACK TACTICS OF IP ATTACKS

3.1 Exploitation

Exploitation is the phase of an attack during which the attacker deliberately exploits vulnerabilities, misconfigurations or weak points in software (OS, applications, network services) or in the human factor (social engineering) in order to gain unauthorized access or execute malicious code. We encounter, for example, techniques that include the exploitation of remote services, client applications, web interfaces or defence systems (sandbox, antivirus).

Exploitation of Remote Services (T1210)

Exploitation of vulnerable network services running on remote servers (e.g. unpatched versions of SMB, RDP, HTTP, FTP, database services). The attacker tries to gain access over the network by exploiting publicly known vulnerabilities (CVE) or 0-day vulnerabilities.

1. **Drive-by Compromise** (T1189)

A compromised or malicious website contains an exploit kit which, when the victim visits, automatically attempts to exploit vulnerabilities in the browser (Internet Explorer, Chrome, Firefox) or its plugins (Flash, Java, Silverlight). The intention is to download and run malware without user interaction.

2. **Exploitation for Client Execution** (T1203)

Opening a document (Office, PDF), running a malicious attachment script or installing a macro that exploits a bug in the victim's software. Examples include the abuse of macros in Word or bugs in PDF readers.

3. **Supply Chain Compromise** (T1195)

The attacker targets a vulnerable part of the supply chain (e.g. a developer, a software distributor) in order to infect legitimate applications and subsequently exploit them at end users. A classic case is a “trojanized” installation package containing malicious code with an exploit.

4. **Hardware/Firmware Exploits**

Less common but serious techniques in which the attacker exploits weaknesses in the firmware implementation (BIOS, UEFI) or bugs in the CPU (e.g. Meltdown, Spectre). This enables access at a very low level (so-called ring-0) and can lead to persistent control of the system without being noticed by antivirus software.

5. **Zero-day Exploits**

Exploitation of vulnerabilities hitherto unknown to the software vendor or the security community. Because no patch exists, the detection and mitigation of such exploits is extremely difficult and often requires behavioural, heuristic or AI/ML techniques.

6. **Exploitation for Defense Evasion** (T1211)

The attacker exploits vulnerabilities in security software (e.g. antivirus, EDR, sandbox) or in the operating system itself in order to bypass protection mechanisms. A typical example is the exploitation of a bug in drivers signed with a trusted certificate.

3.2 Lateral movement

Lateral movement (*Lateral Movement*) is a tactic in which the attacker, after the initial security

breach, e.g. by gaining access to a given device, attempts to move within the network and extend their access and control over other systems. The goal is often to escalate privileges (e.g. obtain higher rights), manage key services (e.g. domain controllers, SMB shares) or access sensitive data.

1. **Remote Services** (T1021) – extended view

In addition to the usual RDP, SMB, SSH and WinRM, more recent cases also show the abuse of cloud services or third-party remote administration tools (e.g. VNC, TeamViewer).

The importance of monitoring unusual times and anomalous geolocation access is emphasized.

2. **Kerberoasting** (T1208)

The attacker obtains Kerberos Service Tickets and then attempts to crack the password offline (hashing using RC4_HMAC_MD5).

Combination with overpass-the-hash or pass-the-ticket further simplifies propagation within Windows domains.

3. **Golden Ticket/Silver Ticket Attacks**

They are specific to Kerberos – a Golden Ticket allows the attacker to create their own TGT (Ticket-Granting Ticket) tickets, thereby gaining almost unlimited access.

A Silver Ticket is a TGS (Ticket Granting Service) ticket specific to a particular service, which also allows unauthorized access, but with a narrower scope.

4. **Credential Stuffing and Overpass-the-Hash**

Credential Stuffing uses bulk login credentials from data leaks and tests their validity within the target infrastructure.

Overpass-the-Hash works similarly to pass-the-hash, but instead of moving the NTLM hash into LSASS, a Kerberos ticket is generated directly from the NTLM/RC4 hash.

5. **Privileged Access Management (PAM) Bypass**

Procedures that bypass systems intended for managing privileged accounts (CyberArk, central password vaults, etc.).

This often involves exploiting shortcomings in the integration interface, inconsistent logging or the handling of temporary tokens that PAM does not notice.

6. **Lateral Tool Transfer** (T1570)

In addition to traditional tools (PsExec, Mimikatz), transfers of custom exploit frameworks (e.g. Impacket, Cobalt Strike Beacon) are emerging.

Network segmentation and checking of EDR/AV network integration are important in order to detect attempts to download large binaries or scripts between segments.

7. **Scripts and in-memory techniques**

The abuse of PowerShell (T1059.001), WMI (T1047) or Dynamic Data Exchange (T1173) makes it possible to execute code in memory without saving it to disk, which makes detection more difficult.

The latest approaches use combinations of obfuscation and legitimate API calls to disguise malicious content.

3.3 Privilege escalation

Privilege escalation is a tactic in which the attacker, after gaining access to a system or network,

looks for ways to obtain a higher level of permissions or administrator rights. The goal is to extend the possibilities of manipulating the system, bypass security restrictions or make changes in key parts of the infrastructure (e.g. a domain controller, database servers, company applications).

1. **Exploitation for Privilege Escalation** (T1068)

Exploitation of vulnerabilities in the OS (Windows, Linux, macOS) or in drivers so that a process with low privileges becomes a process with higher privileges. Common examples include exploits of local escalation vulnerabilities (e.g. CVEs in kernels, printer drivers, etc.).

2. **Bypass User Account Control** (UAC) (T1548.002)

UAC in Windows prevents an ordinary user from performing administrator operations without additional confirmation. Attackers often use various methods of bypassing UAC, e.g. via fake binaries, registry modifications, or abuse of legitimate Windows utilities (CMSTP, schtasks, Event Viewer, etc.).

3. **Access Token Manipulation** (T1134)

In the Windows environment, access tokens are used for the identification and permissions of processes. The attacker can clone, create or modify a token so as to impersonate another user (e.g. an administrator) and gain access to further resources.

4. **Credential Dumping** (T1003)

Obtaining credentials (hashes, passwords, tickets) from memory (LSASS in Windows), the registry, the SAM database or other stores. Tools such as Mimikatz, Impacket or crackmapexec then make it possible to escalate privileges or gain access to other accounts in the domain.

5. **Scheduled Task/Job** (T1053)

Creating or modifying scheduled tasks (e.g. via Windows Task Scheduler or cron in Linux) in order to run malicious code with higher privileges. The attacker uses legitimate system functions, but schedules the task so that it runs with administrator rights.

6. **DLL Search Order Hijacking** (T1574.001)

The Windows operating system first searches for DLL libraries in directories with higher priority. If the attacker knows where and how DLLs are loaded, they can place a malicious DLL with the same name in a higher-priority location in the path, thereby succeeding in running code with higher privileges.

7. **SetUID and SetGID in Unix/Linux** (T1548.001)

The attacker can exploit binaries with SUID bits set (e.g. /bin/su or others) or misconfigured scripts in order to obtain root privileges. Similarly, attackers can exploit errors in the sudo configuration (e.g. if undesirable exceptions are defined in sudoers).

8. **Abusing Application Shims** (T1138)

In Windows there is an Application Compatibility Shims feature that allows applications to run in compatibility mode. Attackers can abuse this functionality to run code with higher privileges if the shim is incorrectly configured.

4 MACHINE LEARNING METHODS AND ALGORITHMS

The subject of this chapter is an analysis of machine learning methods – ML (Machine Learning) methods and algorithms that are used for anomaly detection in network traffic.

4.1 Classification of algorithms and methods by type of learning

4.1.1 Supervised learning

Supervised learning methods or algorithms include learning algorithms that take data samples (known as training data) and the associated outputs (known as a label or response) with each data sample during the model training process. The main goal is to learn the mapping or association between input data samples x and their corresponding outputs y based on multiple instances or samples of training data. This learned knowledge can then be used in the future to predict the output y for any new input data sample x that was previously unknown or unavailable during the model training process (Fig. 2).

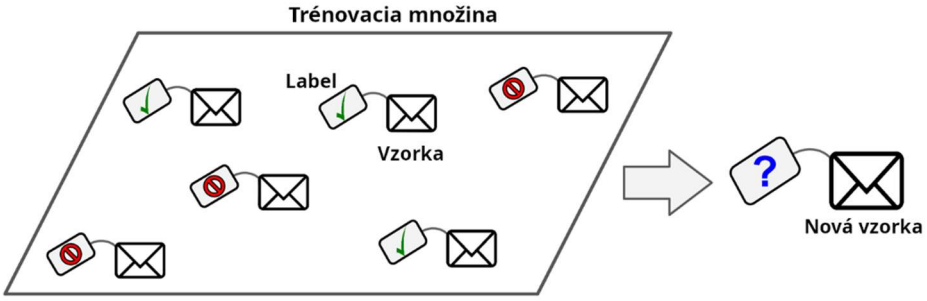


Fig. 2: Example of supervised learning – a labelled training set for a spam detection task.

These learning methods are called supervised learning because the model learns on data samples where the desired output labels are already known in advance in the training phase. Supervised learning essentially tries to model the relationship between inputs and their corresponding outputs from the training data so that we are able to predict output responses for new data inputs based on the knowledge it has previously gained with regard to the relationships and mappings between inputs and their target outputs. This is precisely why supervised learning methods are widely used in predictive analytics, where the main goal is to predict a certain response to some input data that is usually fed into a trained supervised ML model. Supervised learning methods are divided into two main classes according to the type of ML tasks they seek to solve:

- Classification
- Regression

4.1.1.1 Classification

Classification-based tasks are a sub-field within supervised machine learning where the key goal is to predict output labels, which are categorical in nature, for input data based on what the model learned in the training phase. The output labels here are also known as classes or class labels, because they are categorical in nature, which means that they are unordered and discrete values. Each output response therefore belongs to a specific discrete class or category. Suppose we take weather prediction as a real-world example. For simplicity, let us say that we are trying to predict whether the weather is sunny or rainy based on multiple input data samples consisting of attributes or features such as humidity, temperature, pressure and precipitation. Since the prediction can be sunny or rainy, there are two distinct classes in total; therefore this problem can also be referred to as a binary classification problem. Figure 3

shows a binary weather classification task predicting the weather as sunny or rainy based on training a supervised model on input data samples with feature vectors (precipitation, humidity, pressure and temperature) for each data sample/observation and their corresponding class labels as sunny or rainy.

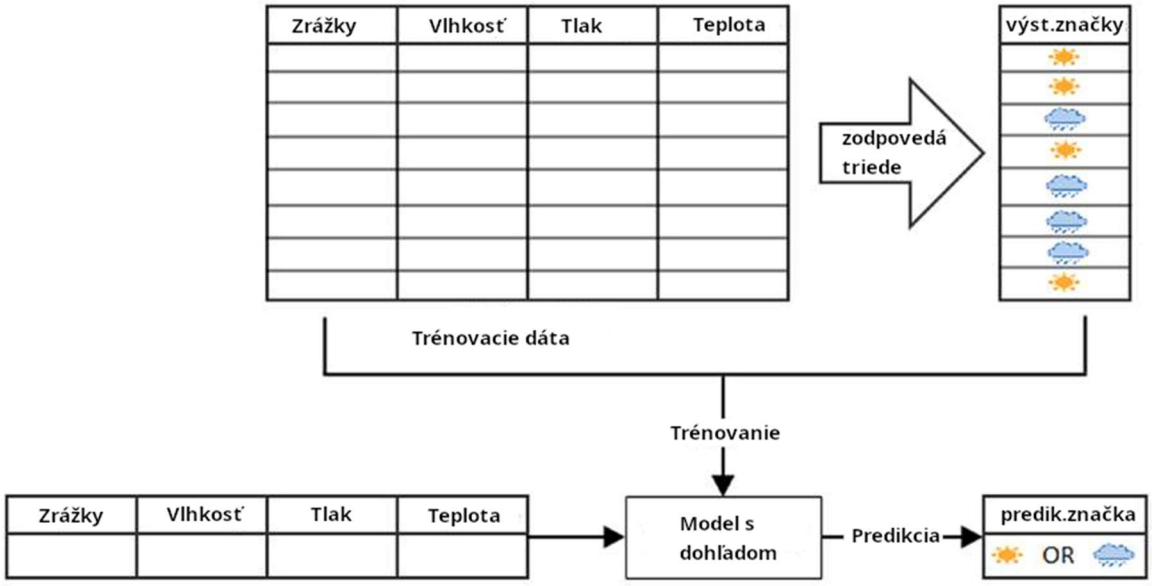


Fig. 3: Binary classification for weather prediction

A task where the total number of distinct classes is more than two becomes a multi-class classification problem, where each prediction response can be any of the probable classes from this set. A simple example would be trying to predict numeric digits from scanned handwritten images. In this case it becomes a 10-class classification problem, because the output class label for any image can be any digit from 0 to 9. In both cases, the output class is a scalar value pointing to one specific class. Multi-label classification tasks are such that, based on any input data sample, the output response is usually a vector with one or more output class labels. A simple real-world problem would be trying to predict the category of a news article, which could have multiple output classes such as news, finance, politics, etc.

4.1.1.2 Regression

Machine learning tasks whose main goal is the estimation of a value can be called regression tasks. Regression-based methods are trained on input data samples that have output responses which are continuous numeric values, as opposed to classification, where we have discrete categories or classes. Regression models use the attributes or features of the input data (also called explanatory or independent variables) and their corresponding continuous numeric output values (also called the response, dependent or outcome or output variable) to learn specific relationships and associations between the inputs and their corresponding outputs. With this knowledge, the model can predict output responses for new, unknown data instances similarly to classification, but with continuous numeric outputs. One of the most common real-world examples of regression is the prediction of house prices. A simple regression model can be built to predict house prices based on data relating to land areas in square metres.

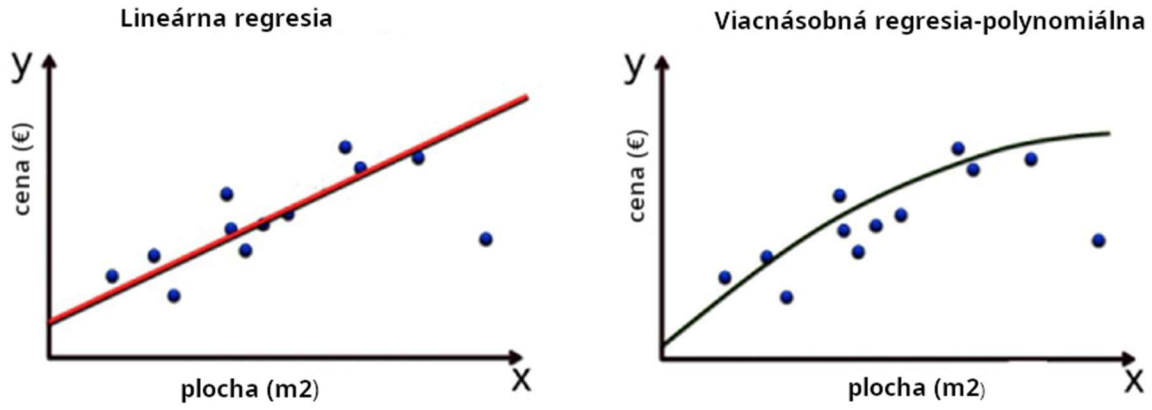


Fig. 4: Regression model for house price prediction

Figure 4 shows two possible regression models based on different methods of predicting house prices on the basis of land area. The basic idea is that we try to determine whether there is any relationship or association between the area of the data feature plot and the outcome variable, which is the house price, and which we want to predict. So once we learn this trend or relationship shown in Figure 3, we can predict house prices in the future for any given plot of land. If you have looked at this figure carefully, we have deliberately shown two types of models to demonstrate that there are multiple ways of building a model based on the training data. The main goal is to minimize errors during model training and validation so that it generalizes well, does not overfit or is not biased only towards the training data, and performs well in future predictions.

Simple *linear regression models* attempt to model relationships in data with one feature or explanatory variable x and one response-output variable y, where the goal is to predict y. Methods such as the method of least squares (ordinary least squares OLS) are usually used to obtain the best linear fit during model training. Multiple regression is also known as multivariable regression. These methods attempt to model data where we have one output response variable y in each observation, but multiple explanatory variables in the form of a vector X instead of one explanatory variable. The goal is to predict y based on the various features present in vector X. A real-world example could be extending our house prediction model by building a more sophisticated model in which we predict the house price based on multiple features instead of only the land area in each data sample. The features can be represented in a vector containing the land area, the number of bedrooms, the number of bathrooms, the total number of floors, and whether the house is furnished or unfurnished. Based on all these attributes, the model tries to learn the relationship between the individual feature vectors and its corresponding house price so that it can predict them in the future.

Polynomial regression is a special case of multiple regression, where the output variable y is modelled as an nth-degree polynomial of the input feature x. It is essentially multiple regression where each element in the input feature vector is a multiple of x. The model on the right in Figure 3 for predicting house prices is a polynomial model of degree 2. Non-linear regression methods attempt to model the relationships between input features and outputs based on a combination of non-linear features applied to the input attributes and the necessary model parameters.

Lasso regression is a special form of regression that performs normal regression and generalizes the model well by performing regularization as well as feature or variable selection. Lasso stands for least absolute shrinkage and selection operator. The L1 norm is usually used as the regularization term in lasso regression.

Ridge regression is another special form of regression that performs normal regression and generalizes the model by performing regularization in order to prevent overfitting of the model. The L2 norm is usually used as the regularization term in ridge regression.

Generalized linear models are general frameworks that can be used to model data predicting different types of output responses, including continuous, discrete and ordinal data. Algorithms such as logistic regression are used for categorical data and ordered probit regression for ordinal data.

The most important supervised learning algorithms include:

- K-nearest neighbours method - K-Nearest Neighbors (KNN)
- Linear regression – Linear Regression
- Logistic regression – Logistic Regression
- Support vector method - Support Vector Machines (SVMs)
- Decision trees – Decision Tree (DT) and random forests – Random Forest (RF)
- Neural networks – Neural Networks (NN). Some neural network architectures, such as the autoencoder and restricted Boltzmann machines (RBM), represent unsupervised learning. Some architectures can represent semi-supervised learning, as for example in the case of using a Deep Belief Network with pre-training based on unsupervised learning.

4.1.1.3 K-Nearest Neighbor

KNN can be defined as a non-linear non-parametric classification method. This algorithm is based on the very simple principle that similar data lie close to each other in the search space, or the data feature space. For each element of the test data, the kNN algorithm searches, based on distance, for a certain vicinity – a neighbourhood – containing k elements of the training, or reference, data and, based on a certain criterion – most often the majority rule – assigns a class label to the given element. The kNN method belongs to the group of so-called lazy learning techniques. These algorithms are specific in that, unlike the model-based methods already mentioned, no generalized model describing the training data and subsequently used in the classification itself is created from the training data. The classification process therefore does not consist of 2 phases – model training and the actual classification of the test data – but takes place in a single step. The kNN algorithm has 2 key elements:

- the distance metric
- the value k of neighbours

It is obvious that in the case of the kNN algorithm, distance expresses the degree of similarity of the data – the closer 2 elements are to each other in the feature space, the more similar they are. From this point of view, the choice of a suitable distance metric plays an important role and depends to a large extent on the given classification problem. Perhaps the best-known measure is the Euclidean distance, defined by the following relationship:

$$d(x,y)=\sqrt{\sum_{k=1}^{n}\left(x_k-y_k\right)^2}\ ,$$

or the standardized Euclidean distance

$$d(x,y)=\sqrt{\sum_{k=1}^{n}\left(x_k-y_k\right).V^{-1}.\left(x_k-y_k\right)}$$

where *x* is the test data vector, *y* is a test data element, ***V*** is a diagonal matrix whose j-th diagonal element is $S(j)^2$, where *S* is the vector of standard deviations. Other metrics include the Mahalanobis distance

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k).C^{-1}.(x_k-y_k)'}\ ,$$

where $\boldsymbol{C}^{-1}$ is the inverse covariance matrix, or the Manhattan metric

$$d(x,y)=\sqrt{\sum_{k=1}^{n}\left|x_k-y_k\right|}\ .$$

The value *k* determines the "neighbourhood" or vicinity, that is, the number of training data elements that are taken into account when determining whether a certain test data element belongs to a given class. If the value of k is too small, the classification result may be sensitive to so-called noisy points. On the other hand, if the value of k is too large, the "neighbourhood" may contain too many elements from other classes. The example is illustrated in the following figure.

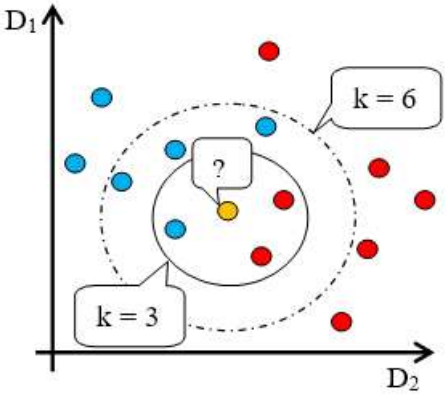


Fig. 5: Demonstration of the effect of the value of k on the accuracy of the kNN algorithm

The task is to determine which class the orange circle belongs to. In the first case, for *k* = 3, the given element will belong to the class of red circles; in the second case, for *k* = 6, it will belong to the class of blue circles. K is usually chosen as an odd number.

The final determination of the class to which a given test data element should belong, from the set of possible classes determined by the "neighbourhood", is usually subject to the majority rule (majority voting) described by the following relationship:

$$l=\arg\max_{v}\sum_{(x_i,y_i)\in D_Z}I(v=y_i)\ ,$$

where v is the class label, y_i is the class label of the i-th nearest neighbour and *I*(.) is a function that returns the value 1 if the argument is true. Another example is the selection of the class based on distance weighting

$$l=\arg\max_{v}\sum_{(x_i,y_i)\in D_Z}w_i*I(v=y_i)\ ,$$

where the weighting factor w_i is the reciprocal of the distance between the test and training , or reference, element

$$wi = \frac{1}{d(x_i, y_i)^2} \ .$$

If this decision rule is used, the kNN algorithm is not so sensitive to the size of the value *k*.

Despite its simplicity, the kNN method is very successful in the field of data retrieval and ranks among the ten best algorithms in that field. In some cases it can even outperform much more sophisticated and complex classification methods. Thanks to its simplicity, this method is very easy to implement, is suitable for multimodal classes and is very flexible. Compared to the classification methods mentioned above, the very time-consuming (especially with SVM) model-building phase is eliminated, although the classification itself usually requires a longer time interval compared to those methods. The disadvantage is therefore the growing computational complexity with large databases. Several different later modifications and variations of the original algorithm have been published over time (fuzzy kNN).

4.1.1.4 Support Vector Machine

The support vector method is a classification and regression method proposed in the 1990s by V. Vapnik, which essentially determines a boundary curve separating 2 classes. The main idea is to find a hyperplane that would optimally divide the input data in the feature space. For a given set of training data belonging to 2 classes (x_1, y_1), ...,(x_l, y_l), where $x_i \in \boldsymbol{R}_n$ and $yi \in$ {-1, 1}, we therefore try to find a hyperplane $wx + b = 0$ so that we can separate the data. Of all the hyperplanes determined by *w* and *b*, there is only one that maximizes the distance between the closest points of these classes and the hyperplane – the optimal hyperplane, and the following holds:

$$d(w,b) = \min_{\{x_i | y_i = 1\}} \frac{wx_i + b}{|w|} - \max_{\{x_i | y_i = -1\}} \frac{wx_i + b}{|w|}$$

and the corresponding minimum and maximum values are -1 and +1, respectively

$$d(w,b) = \left| \frac{1}{\mathbf{w}} - \frac{-1}{\mathbf{w}} \right| = \frac{2}{|\mathbf{w}|} \ .$$

By minimizing the expression $|w|^2 / 2$ subject to the condition $y_i (wx_i + b) \geq 1$, we obtain the maximum distance between the 2 classes for the given pair of hyperplanes. The solution of the SVM optimization problem is given by the saddle point of the Lagrange function:

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i \left[y_i (wx_i + b) - 1 \right] \ ,$$

where α_i is the Lagrange multiplier. While $L(w, b, \alpha)$ decreases with *w* and *b*, the derivatives of $L(w, b, \alpha)$ are equal to zero for all $\alpha_i \geq 0$. The solution is given by

$$\overline{w} = \sum_{i=1}^{l} \overline{\alpha}_i y_i x_i \qquad \overline{b} = -\frac{1}{2} \overline{w} \left[x_r + x_s \right] \ ,$$

where x_r and x_s are support vectors belonging to the class $y_r = 1$ and $y_s = -1$, respectively. The data in the original feature space are not separable by any linear hyperplane, but they may be linearly separable in a non-linearly mapped feature space, which differs from the original one and usually has a higher dimension. This non-linear mapping $K(.,.)$ is carried out by means of kernel functions:

$$f(x) = \text{sgn}\left(\sum_{i=1}^{l} \overline{\alpha}_i y_i K(x_i, x) + b\right)\sum_{i=1}^{N} \alpha_i t_i K(x, x_i) + b\,,$$

where y_i are the target values, $\sum_{i=1}^{l} \overline{\alpha}_i y_i = 0$ and $\overline{\alpha}_i > 0$. The kernel $K(.,.)$ is constructed so as to satisfy specific properties depending on the given problem, and we can write:

$$K(x, y) = \Phi(x)\Phi(y)\ ,$$

where $\Phi(x)$ is the mapping of the input feature space into a new space with a potentially infinite dimension. There are 3 kernel functions providing non-linear mapping:

- *polynomial* $K(x, y) = (xy + 1)\ z$ – where z represents the degree of the polynomial
- *RBF Gaussian radial basis functions* $K(x, y) = \exp\{-[(x - y)^2 / 2\sigma^2]\}$ – where σ is the standard deviation of the Gaussian function
- *MLP function* $K(x, y) = \tanh(\text{scale}(xy) - \text{offset})$ – scale and offset represent known and given parameters.

4.1.1.5 Decision Tree

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.

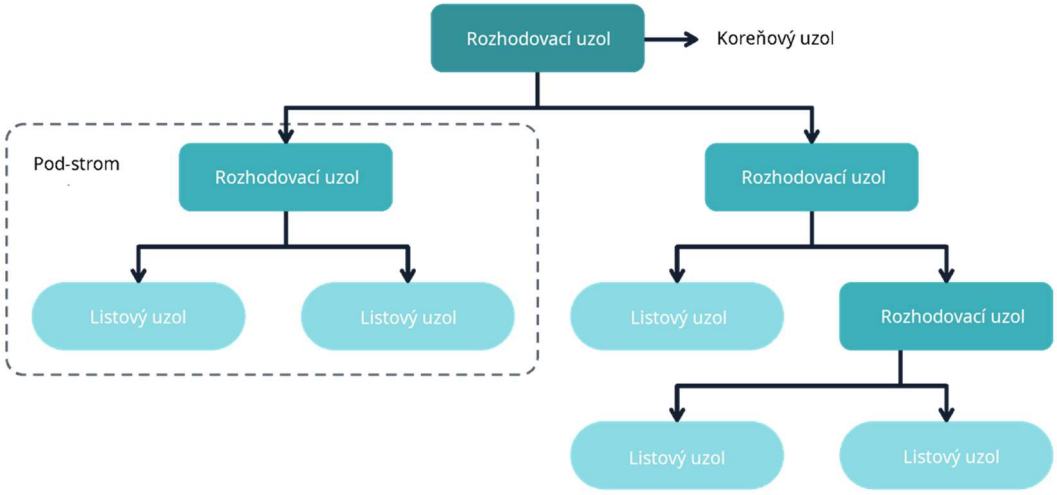


Fig. 6: Decision tree

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:

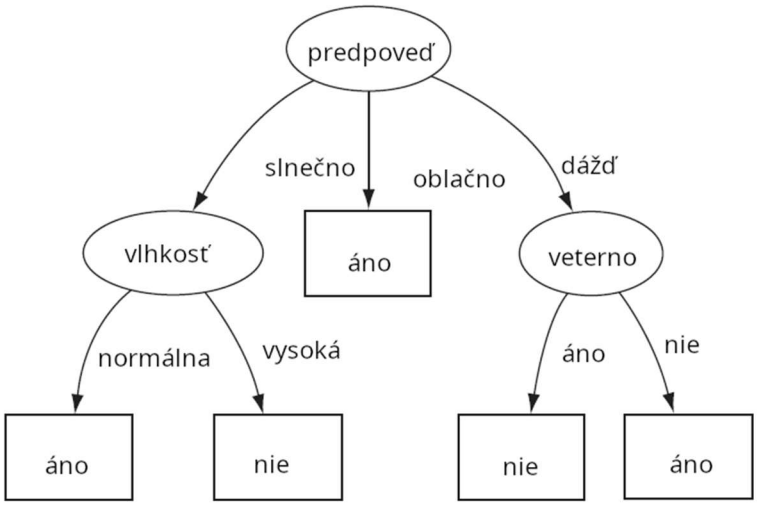


Fig. 7: Decision tree for the Golf dataset

This type of flowchart structure also creates an easily digestible representation of decision-making, which enables different groups across the whole organization to better understand why a decision was made. Decision tree learning uses 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 of the 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 subtree. When this occurs, it is known as data fragmentation, and it can often lead to overfitting. As a result, decision trees have a preference for small trees, which is consistent with the principle of parsimony in Occam's razor; that is, "entities should not be multiplied beyond necessity". In other words, decision trees should add complexity only if necessary, as the simplest explanation is often the best. To reduce complexity and prevent overfitting, pruning is usually employed; this is a process that removes branches that split on features with low importance. The fit of the model can then be evaluated through the process of cross-validation.

Another way in which decision trees can maintain their accuracy is by forming an ensemble via a random forest algorithm; this classifier predicts more accurate results, particularly when the individual trees are uncorrelated with each other. Among the multiple variants, the following most frequently used ones can be mentioned:

- ID3
- C4.5
- CART
-

4.1.2 Unsupervised learning

Supervised learning methods usually require certain training data where the outcomes we are trying to predict are already available in the form of discrete labels or continuous values. In practice, however, we often do not have pre-labelled training data available, and we still want to extract useful knowledge or patterns from our data. In this scenario, unsupervised learning methods are extremely effective. These methods are called unsupervised because the model or algorithm tries to learn inherent latent structures, patterns and relationships from

the given data without any help or supervision, such as the provision of annotations in the form of labelled outputs or outcomes. Unsupervised learning is more concerned with trying to extract meaningful insights or information from data than with trying to predict some outcome based on previously available supervised training data. There is greater uncertainty in the results of unsupervised learning, but a lot of information can also be gained from these models that was previously not available when looking at the raw data. Unsupervised learning can often be one of the tasks involved in building a huge intelligence system. For example, we could use unsupervised learning to obtain possible outcome labels for the sentiment of tweets using knowledge of the English vocabulary, and then train a supervised model on similar data points and their outcomes that we previously obtained through unsupervised learning. There is no fixed rule as regards using only one specific technique. Multiple methods can always be combined as long as they are relevant to solving the problem. Unsupervised learning methods can be categorized into the following broad areas of ML tasks relevant to unsupervised learning.

- clustering – clustering
- dimensionality reduction – dimensionality reduction
- anomaly detection – anomaly detection
- association rule mining – association rule-mining

4.1.2.1 Clustering

Clustering methods are machine learning methods that try to find patterns of similarity and relationships between data samples in our dataset and then group these samples into different groups, so that each group or cluster of data samples has a certain similarity based on inherent attributes or features. These methods are completely unsupervised because they attempt to cluster data by looking at the features of the data without prior training, supervision or knowledge of the data attributes, associations and relationships. A real-world example is the problem of running multiple servers in a data centre and trying to analyse logs for typical problems or errors. Our main task is to determine the different kinds of log messages that usually occur frequently each week. Simply put, we want to group the log messages into different clusters based on some inherent characteristics. A simple approach would be to extract features from the log messages, which would be in text format, apply clustering to them and group similar log messages based on content similarity. Figure 4 shows how clustering would solve this problem. Essentially, we start with raw log messages. Our clustering system would use feature extraction to extract features from the text, such as word occurrences, phrase occurrences, etc. Finally, a clustering algorithm such as K-means or hierarchical clustering would be used to group or cluster the messages based on the similarity of their inherent features.

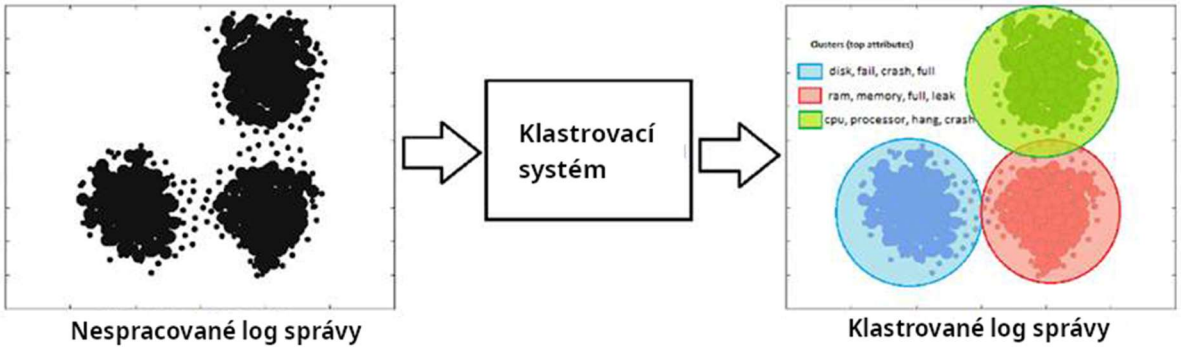


Fig. 8: Clustering

From Figure 7 it is quite obvious that our system has three distinct clusters of log messages, where the first cluster shows disk problems, the second cluster is about memory problems and the third cluster is about processor problems. The figure also shows the main words that helped distinguish the clusters and group similar data samples (logs). Of course, sometimes some features may be present in multiple data samples, and therefore slight overlapping of clusters may also occur, since this is unsupervised learning. However, the main goal is always to create clusters so that the elements of each cluster are close to each other and far from the elements of other clusters. There are various types of clustering methods, which can be classified according to the following main approaches.

- Centroid-based methods – K-means, K-medoids
- Hierarchical clustering methods – Hierarchical Cluster Analysis (HCA)
- Distribution-based clustering methods – Gaussian Mixture Model (GMM)
- Density-based methods – DBSCAN, OPTICS

4.1.2.2 K-means

K-means, or the method of K mean values, is one of the most popular clustering methods. The pseudo-code given below shows the K-means clustering procedure.

input: K - number of clusters, D - dataset of N points

output: set of K clusters

- initialization
- **repeat**

  for each point p in D **do**

  find the nearest centroid and assign p to the corresponding cluster

  end for

  update the clusters by computing new centres using the mean of the members

- **until** the stop criteria are not met
- **return** the clustering result.

The basic idea is: Given an initial but not optimal clustering, move each point to its new nearest centre, update the cluster centres by computing the mean of the member points, and repeat the relocation and updating process until the convergence criteria are met (such as a predefined number of iterations, a difference in the value of the criterion function). The task of initialization is to create the initial K clusters. Many initialization techniques have been proposed, ranging from simple methods such as selecting the first K data points,

Forgy initialization (random selection of K data points in the dataset) and *Random partitions* (random division of the data points into K subsets), to more sophisticated methods such as density-based initialization, *Intelligent initialization*, *Furthest First initialization* (FF, in short, works by randomly selecting the first centre point), then adding further centre points that are furthest from the existing ones) and *subset furthest-first* (SFF) initialization. The figures below show an example of K-means clustering on a set of points, with K = 2. The clusters are initialized by randomly selecting two points as centres.

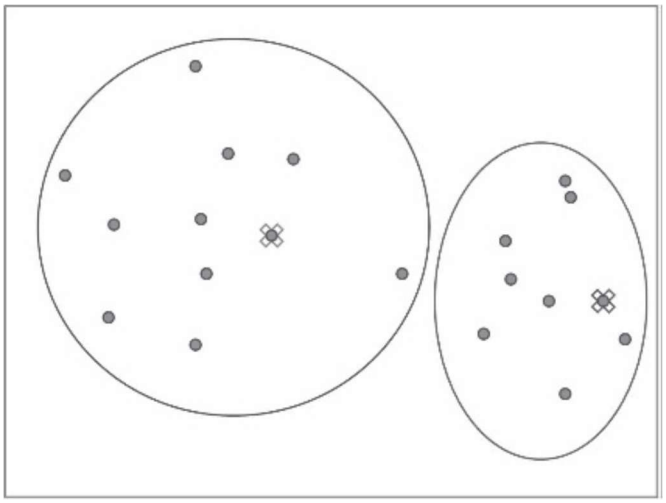


Fig. 9-1: k-means initialization

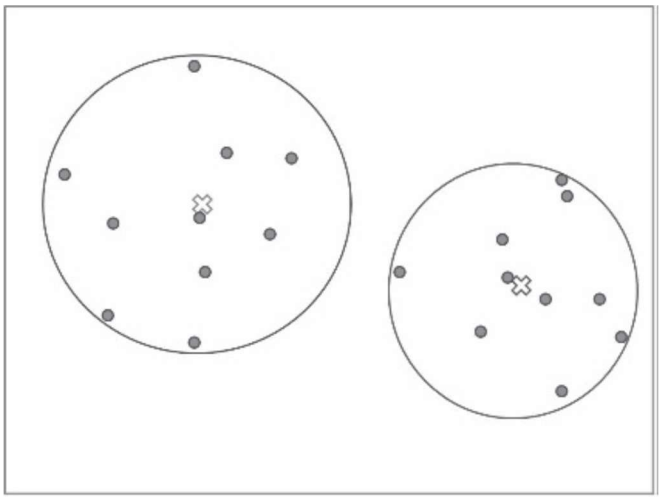


Fig. 9-2: k-means update

4.1.2.3 Gaussian Mixture Model

The GMM method is based on the statistical properties of signals and assumes that the observed process and its distribution can be modelled by the sum of a certain group of components and their probability density distributions:

$$P(x|\lambda) = \sum_{m=1}^{M} c_m b_m(x) ,$$

where x is a random vector of dimension d, M is the number of model components, c_m is the weight vector and $b_m(x)$ is the Gaussian distribution defined by the mean vector μ_m and the covariance matrix Σ_m, where the following holds:

$$b_m(x) = \frac{1}{(2\pi)^{d/2}\left|\Sigma_m\right|^{1/2}} \exp\left\{-\frac{1}{2}\left[(x-\mu_m)\Sigma_m^{-1}(x-\mu_m)^T\right]\right\} .$$

GMM is a parametric model and its parameters are usually denoted $\lambda = \{c_m, \mu_m, \Sigma_m\}$ for $m = 1,2,\ldots M$. These parameters are estimated so that the probability $P(x \mid \lambda)$ of the input data for all classes is maximal. One useful algorithm is the iterative *Expectation Maximization* (EM) algorithm, which adjusts the model parameters based on the following equations:

$$\mu_m = \frac{\sum_{i=1}^{n} x_i P(m|x_i, \lambda)}{\sum_{i=1}^{n} P(m|x_i, \lambda)}$$

$$\Sigma_m = \frac{\sum_{i=1}^{n} P(m|x_i, \lambda)(x_i - \mu_m)(x_i - \mu_m)^T}{\sum_{i=1}^{n} P(m|x_i, \lambda)}$$

$$c_m = \frac{1}{n} \sum_{i=1}^{n} P(m|x_i, \lambda)$$

$$P(m|xi, \lambda) = \frac{c_m b_m(x_i)}{\sum_{j=1}^{M} c_j b_j(x_i)}$$

For a set of K sound classes represented by GMM models λ_1 λ_2, ... , λ_K, the goal of recognition (classification) is to find the model that has the maximum a posteriori probability $P(\lambda \mid X)$ (MAP criterion) for a given sequence of observations $X = x_1, x_2, \ldots, x_L$:

$$\hat{K} = \arg\max_{1 \le k \le K} P(\lambda_k | X) = \arg\max_{1 \le k \le K} \frac{P(X|\lambda_k) P(\lambda_k)}{P(X)} \ ,$$

and since the probability $P(X)$ is the same for all models and does not change the result of the maximization, the expression in the denominator can be neglected, so that the following then holds:

$$\hat{K} = \arg\max_{1 \le k \le K} P(X|\lambda_k) P(\lambda_k) \ .$$

By applying the logarithm operation and considering the statistical independence between individual observations, we can then write:

$$\hat{K} = \arg\max_{1 \le k \le K} \sum_{l=1}^{L} \log P(x_l|\lambda_k) P(\lambda_k) \ ,$$

and if we assume that the probability $P(\lambda_k)$ is the same for all classes of the dataset, we can omit this probability from the previous relationship.

4.1.2.4 DBSCAN

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 needed to form a cluster from a group of points, and ξ (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 a circle of radius ξ
- border points – are points that are not core points but are located in the neighbouring area 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 parameter values, the minimum number of points and ξ.
2. randomly selecting a starting point (say A) and finding all points at a distance of ξ 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 ξ or less are border points and are added to the cluster centred at point A.
4. repeating these steps for each point. If a point (say B) added to the cluster turns out to also be a core point, we first form 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.

DBSCAN offers two strong advantages over K-Means clustering:

- it is not necessary to specify the number of clusters. When analysing data, especially in unsupervised learning settings, we may often not be aware of the number of classes in the data.

Therefore, it is not known what the number of clusters for anomaly detection should be. DBSCAN eliminates this problem and forms clusters appropriately based on density.

DBSCAN is able to find clusters that have arbitrary shapes. Because it is based on the density of the regions around the core points, the shape of the clusters does not have to be circular. On the other hand, K-Means clustering cannot detect overlapping or arbitrarily shaped clusters.

4.1.3 Dimensionality reduction

Once we start extracting attributes or features from raw data samples, the feature space can sometimes become bloated with a huge number of features. This poses multiple challenges, including the analysis and visualization of data with thousands or millions of features, which makes this feature space extremely complex and poses problems with regard to training models, memory and space constraints. In fact, this is referred to as the "curse of dimensionality". In these scenarios, unsupervised learning methods can also be used, in which we reduce the number of features or attributes for each data sample. These methods reduce the number of feature variables by extracting or selecting a set of principal or representative features. Multiple popular algorithms are available for dimensionality reduction:

- principal component analysis PCA (Principal Component Analysis PCA)
- independent component analysis ICA (Independent Component Analysis ICA)
- discriminant analysis (Discriminant Analysis)

Figure 10 shows the output of a typical feature reduction process applied to the 3D Swiss Roll structure, which has three dimensions, to obtain a two-dimensional feature space for each data sample using PCA.

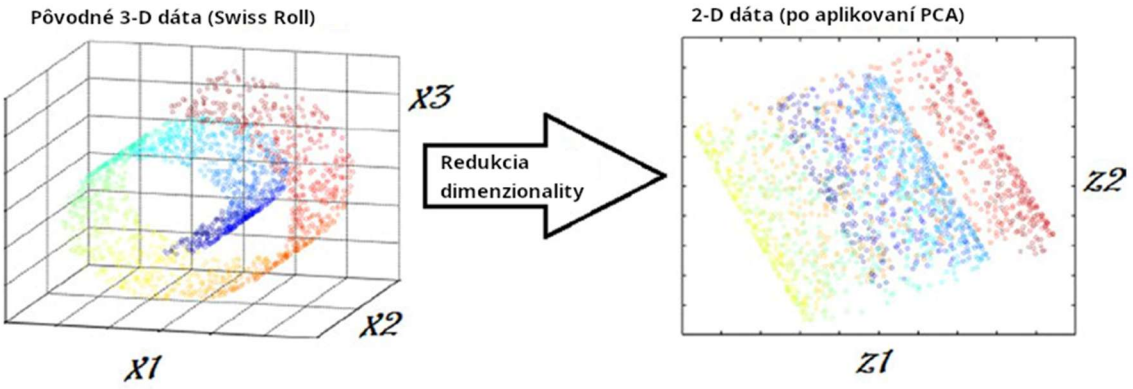


Fig. 10: Dimensionality reduction using PCA

From Figure 10 it is quite clear that each data sample originally had three features or dimensions, namely D(x1, x2, x3), and after applying PCA we reduce each data sample in our dataset to two dimensions, namely D'(z1, z2). Dimensionality reduction techniques can be classified into two main approaches as follows.

- Feature **selection** methods: Specific features are selected for each data sample from the original list of features and the other features are discarded. No new features are generated in this process.
- Feature **extraction** methods: We design or extract new features from the original list of features in the data. The reduced subset of features will therefore contain newly generated features that were not part of the original feature set. PCA belongs to this category.

4.1.4 Anomaly detection

The process of anomaly detection is also called outlier detection, where we are interested in detecting the occurrence of rare events or observations that do not usually occur, based on historical data samples. Sometimes anomalies occur infrequently and are therefore rare, and in other cases anomalies may not be rare but may occur in very short bursts over time, so they have specific patterns. Unsupervised learning methods can be used for anomaly detection, so that we train the algorithm on a training dataset that contains normal, non-anomalous data samples. Once it has learned the necessary data representations, patterns and relationships between attributes in normal samples, for any new data sample it would be able to identify it as an anomalous or normal data point using its learned knowledge. Figure 11 shows some typical scenarios based on anomaly detection. The most common methods include:

- One-class SVM – One-class SVM
- Isolation forest – Isolation Forrest

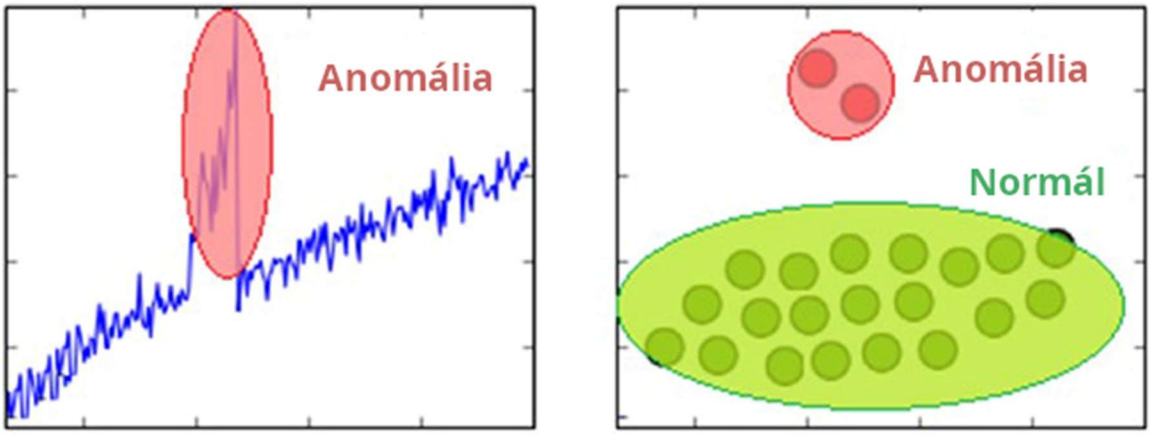


Fig. 11: Anomaly detection

Methods based on anomaly detection are extremely useful in various real-world cases, such as the detection of security attacks or intrusions, credit card fraud, manufacturing anomalies, network problems and many others.

4.1.5 Association Rule-Mining

Association rule mining is usually a data mining method used to explore and analyse large transactional datasets in order to find patterns and rules of interest. These patterns represent interesting relationships and associations between various items across transactions. Association rule mining is often also called market basket analysis, which is used to analyse customers' purchasing patterns. Association rules help in detecting and predicting transactional patterns based on the knowledge obtained from training transactions. Using this technique, we can answer questions about which items people tend to buy together, thereby indicating frequent itemsets. We can also associate or correlate products and items, i.e. insights such as that people who buy beer tend to buy chicken wings in a restaurant. Figure 12 shows how a typical association rule mining method should ideally work on a transactional dataset.

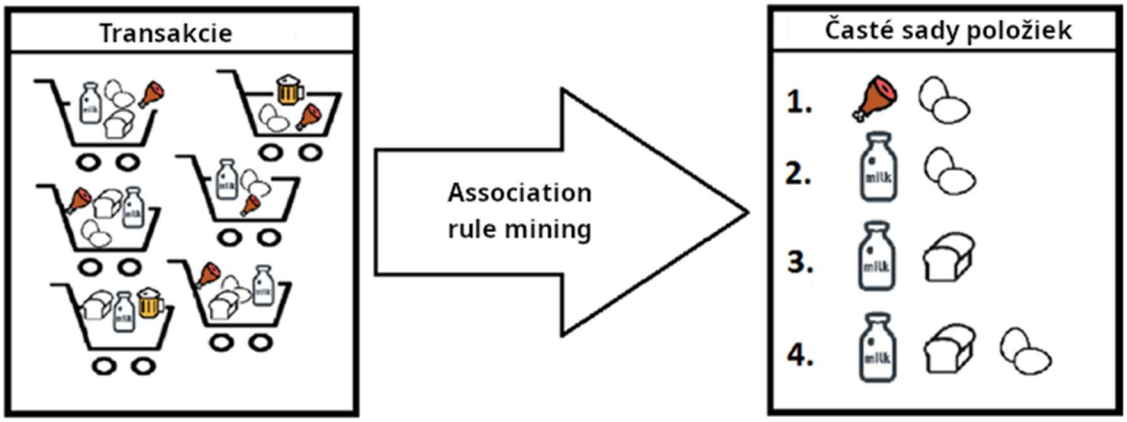


Fig. 12: Association rule mining

From Figure 12 it can clearly be seen that, based on various customer transactions over a certain period of time, we have obtained items that are closely associated and that customers tend to buy together. Some of these frequent itemsets are shown as {meat, eggs}, {milk, eggs} and so on. The criterion for determining good-quality association rules or frequent itemsets

is usually applied using metrics such as support, confidence and lift. This is an unsupervised method, because we do not know in advance what the frequent itemsets are or which items are more strongly associated with which items. Only after applying decision algorithms can we discover and predict products or items that are closely associated with each other and find conditional probabilistic dependencies. The most frequently used decision algorithms include:

- Apriori
- FP-growth
- Eclat

4.1.6 Semi-supervised learning

Since labelling data is usually time-consuming and costly, there are often many unlabelled instances or examples available and few labelled instances. Some algorithms can work with data that is partially labelled. This is called semi-supervised learning, an example of which with two classes is shown in Figure 13. The unlabelled examples (grey circles) help classify the new instance (cross) into the “triangle” class and not into the “square” class, even though it is closer to the labelled examples of the “square” class.

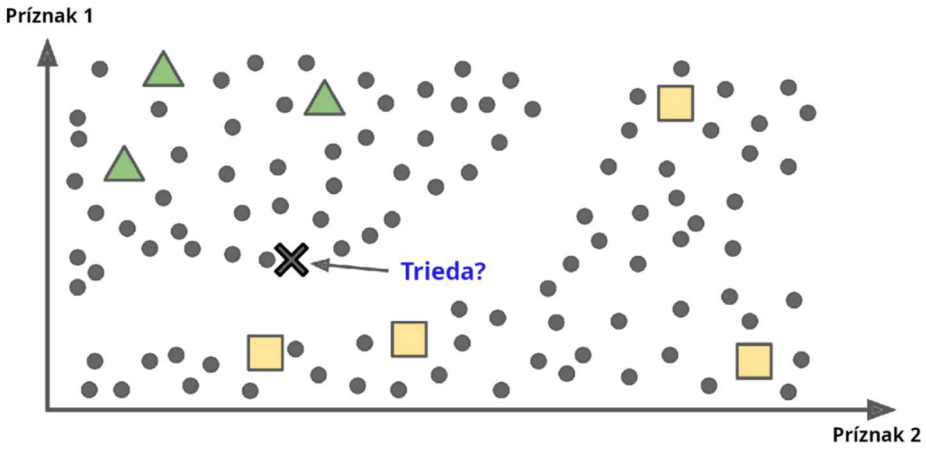


Fig. 13: Example of semi-supervised learning

Some photo hosting services, such as Google Photos, are a good example of this. When you upload all your family photos to the service, it automatically recognizes that the same person A appears in photos 1, 5 and 11, while another person B appears in photos 2, 5 and 7. This is the unsupervised part of the algorithm (clustering). Now all the system needs is for you to tell it who these people are. Just add one label per person 4 and it can name everyone in every photo, which is useful when searching for photos. Most semi-supervised learning algorithms are a combination of unsupervised and supervised algorithms. For example, *Deep Belief Networks* (DBN) are based on unsupervised components called *Restricted Boltzmann machines* (RBM) stacked on top of each other. The RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques.

4.1.7 Reinforcement learning (reward-based learning)

Reinforcement learning methods differ somewhat from conventional supervised or unsupervised methods. The learning system, which in this context is called an agent, can observe the environment, select and perform actions and, based on these decisions, receive rewards (or penalties in the form of negative rewards, as shown in Figure 14). It must then learn by itself what the best strategy, called a policy, is in order to obtain the greatest reward over time. A

policy defines which action the agent should choose when it is in a given situation. The main steps of the reinforcement learning method are given as follows.

1. preparing the agent with a set of initial policies and strategies
2. observing the environment and the current state
3. selecting the optimal policy and performing an action
4. obtaining the corresponding reward/penalty
5. updating the policy if necessary
6. iteratively repeating steps 2-5 until the agent learns the most optimal policies

An example of a real problem might be a situation where you are trying to get a robot or machine to learn to play chess. In this case, the agent would be the robot, and the environment and states would be the chessboard and the positions of the chess pieces. A suitable reinforcement learning methodology is shown in the figure

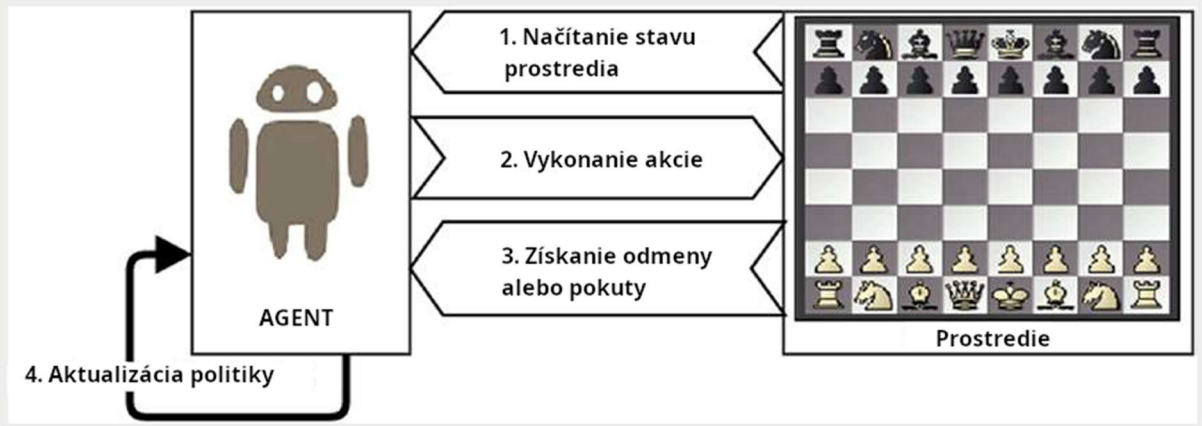


Fig. 14: Reinforcement learning – training an agent to play chess

For example, many robots implement reinforcement learning algorithms in order to learn to walk. DeepMind's AlphaGo program is also a good example of this type of learning: it made newspaper headlines in May 2017 when it beat the world champion Ke Jie at the game of Go. It learned its winning policy by analysing millions of games and then playing many games against itself. Note that learning was switched off during the matches against the champion; AlphaGo was merely applying the policy it had learned.

4.2 Classification of algorithms and methods by type of attack

4.2.1 Network intrusion detection (Network Intrusion Detection (NIDS))

The most common methods:

- Statistical methods:
  - Anomaly detection: modelling of basic network parameters (mean/variance of packet size and frequency) using the Z-score or “Tukey fences”
  - Entropy analysis: detection of DDoS (low entropy in destination IP addresses) or scanning (high entropy in ports)
  - Time series models: ARMA, ARIMA ARFIMA for predicting network traffic volume
- ML methods:

- Supervised learning:
  - Random Forest/GBT: high accuracy in classifying attacks (e.g. DoS, Probe)
  - SVM: effective for data with a large number of features (e.g. the NSL-KDD dataset)
- Unsupervised learning:
  - Clustering: k-means, DBSCAN – grouping of similar traffic patterns
  - PCA: uses dimensionality reduction to identify outlier flows (points lying outside the “standard” positions)

- Deep Learning methods:
  - LSTM/GRU: detection of sequential attacks (e.g. brute-force attacks over time)
  - Autoencoder: flags traffic reconstruction with a high error as anomalies
    - Graph Neural Networks (GNNs): models the network topology for the purpose of detecting lateral movement
    - Reinforcement Learning (RL): Adaptive policy tuning for dynamic defence

4.2.2 Malware detection

The most widely used methods:

- Statistical methods:
  - Signature Hashing: Fuzzy hashing (ssdeep) for detecting variants
  - PE header analysis: Section entropy (.text, .data) for detecting encapsulation
- ML methods:
  - Feature-based:
    - Static: N-grams from bytecode, import tables and strings
    - Dynamic: API call sequences, registry changes (XGBoost, LightGBM)
  - Deep Learning methods:
    - CNN: image-based malware classification (binaries represented as an RGB image)
    - RNN/Transformer: analysis of API call sequences

4.2.3 Spam/Ham & Phishing detection

The most widely used methods:

- Statistical methods:
  - Bayesian filters: Naive Bayes for assessing token probability (word/URL).
  - TF-IDF: determines the weight of keywords (e.g. "urgent", "invoice").
- ML methods:
  - SVM/RF: classification using lexical features (header, body).
  - Logistic regression: baseline for URL/domain blacklists

- Deep Learning methods:
  - BERT/Transformers: contextual classification of phishing e-mails
  - CNN: detects malicious logos in e-mails

4.2.4 Spoofing detection

The most widely used methods:

- Statistical methods:
  - SPF/DKIM/DMARC: uses e-mail authentication protocols (statistical policy checks).
  - Geolocation analysis: exploits IP-geo mismatch (e.g. a login from an unexpected country).
- ML methods:
  - IP spoofing:
    - Ensemble methods: detection of TCP SYN anomalies (e.g. random source IP addresses)
  - Email spoofing:
    - RNN: analyses header inconsistencies (e.g. "Reply-To" vs. "From").

4.2.5 Botnet detection

The most widely used methods:

- Statistical methods:
  - Flow correlation: detect periodic warning signs – beaconing (e.g. the entropy of data flow intervals)
  - DNS analysis: unusual TTL values or NXDOMAIN rates
- ML methods:
  - Supervised learning:
    - Random Forrest, SVM: classify traffic based on packet size, timing, ports
  - Unsupervised learning:
    - Clustering, k-means: uses grouping of similar C&C communication patterns
- Deep learning methods:
    - LSTM: uses temporal modelling of C&C signals (heartbeat signals)
    - GNNs: detection of P2P botnet topologies

4.3 Classification of algorithms and methods according to the use of neural networks

This chapter describes machine learning methods and algorithms for anomaly detection in computer networks, dividing them into classical ML methods and methods using neural networks. For each class of methods, it states its advantages and disadvantages, as well as the most widely used methods. For the individual methods, it gives a recommendation as to which

type of attack they are suitable for detecting. It also states evaluation criteria for the methods and gives an overall recommendation of methods.

To ensure the faultless and reliable operation of present-day computer networks and information systems, it is essential to protect these systems against attacks of various types. These attacks may manifest themselves through the occurrence of unusual events and situations, or of an unusual pattern of network traffic. Such information is obtained continuously by monitoring network traffic and recording log messages from various devices. In present-day networks, this activity is generally performed automatically by means of software tools – so-called anomalies are detected. An anomaly may, but need not, conceal an attack. The system alerts the network administrator to the anomaly, or responds to the situation itself.

Machine learning (ML) methods are used to detect such anomalies in the network (unusual events and situations in the network, as well as to detect unusual user behaviour). We divide these methods into two large groups – classical ML methods and methods that use neural networks, or deep neural networks (DNN). Both categories have their advantages and disadvantages and cases in which it is appropriate to use them. In simpler tasks it is often sufficient to use fast classical ML methods, whereas methods based on neural networks, which achieve better results here, are used to detect complex patterns.

In the following sections, we describe machine learning algorithms and methods suitable for the detection and prediction of security and operational anomalies. We describe how security anomalies and operational anomalies are defined, and what the difference is between anomaly detection and anomaly prediction. We divide the methods into two groups – classical machine learning algorithms and machine learning algorithms based on neural networks. We describe the principle of the algorithms, the required input data and their preparation, the parameters of the algorithms and the training process. And examples of their use for attack detection.

4.3.1 Classical machine learning algorithms for anomaly detection

We consider classical machine learning algorithms to be algorithms that do not use neural networks. We analyse only algorithms used for anomaly detection in computer networks

Machine learning (ML) plays a key role in the field of cybersecurity and IT systems management. Although deep neural networks (DNN) have achieved significant results in many tasks, their deployment often requires considerable computational resources, large amounts of labelled data and expertise. In many cases, however, classical machine learning algorithms such as decision trees, SVM, KNN or clustering methods are sufficient, and they can be more efficient, more transparent and easier to interpret. We focus on methods used for anomaly detection. First, we introduce the basic concepts.

Definition of security anomalies and operational anomalies

- **Security anomalies** are unusual events or patterns in network or system activity that may indicate a cyberattack, such as an intrusion, malware, DoS/DDoS, data exfiltration or unauthorized access.
- **Operational anomalies** concern deviations in the normal running of IT systems, such as service outages, exceptional resource consumption, software errors or unplanned device restarts. They may be of technical origin, without a direct security basis.

Anomaly detection vs. anomaly prediction

- Anomaly **detection** means the identification of deviations in real time or retrospectively in historical data. It is a reactive approach that focuses on current or recent incidents.

- Anomaly **prediction** is a proactive method whose aim is to predict the occurrence of an anomaly on the basis of trends, correlations and the evolution of data. It is used mainly in the prevention of outages or cyberattacks.

4.3.1.1 Classical supervised, unsupervised and semi-supervised algorithms

In this section, we divide classical machine learning methods into categories according to whether the respective methods require the training data to be labelled.

When solving practical tasks, it is often difficult to obtain labelled data. An expert or an auxiliary system must assign labels to the data – whether they are data obtained during an attack, or data about a normal situation when no attack was taking place.

We divide machine learning methods into three basic types:

- Supervised learning – requires a dataset with labelled examples of normal and anomalous behaviour.
- Unsupervised learning – uses unlabelled data to identify patterns and outliers.
- Semi-supervised learning – combines the advantages of both approaches, which is suitable when labelled data are scarce.

The following algorithms can be assigned to the individual classes of machine learning:

1. Supervised algorithms (supervised learning – all training data are labelled):

- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- k-Nearest Neighbors (KNN)
- Logistic regression

2. Unsupervised algorithms (unsupervised learning – the data are not labelled):

- K-means clustering
- DBSCAN
- Principal Component Analysis (PCA)
- Isolation Forest

3. Semi-supervised approaches:

- They combine known cases with unlabelled data.

4.3.1.2 Input data for the algorithms – data sources and preparation

Input data obtained from various sources are used for ML algorithms:

- Network logs (e.g. NetFlow, firewall, IDS/IPS)
- System logs (syslog, Windows Event Log)
- System performance data (CPU, memory, I/O)
- Access and authentication records

Before the data are used to train an ML model, they need to be prepared and pre-processed in the following steps:

- Collection and aggregation of data from various sources.
- Data cleaning – removal of errors, duplicates and irrelevant data.
- Feature extraction (feature engineering) – e.g. the number of connections per minute, the volume of transferred data, the number of failed logins.
- Normalization or scaling of values for scale-sensitive algorithms.

Data labelling – in the case of supervised learning.

4.3.1.3 Algorithm parameters and the training process

To achieve the highest possible anomaly recognition success rate, it is recommended to perform a larger number of experiments for the individual methods while appropriately setting the following parameters:

- **Decision Trees / Random Forest:** tree depth, number of trees, minimum number of samples per split.
- **SVM:** choice of kernel, parameter C (regularization), γ.
- **KNN:** number of neighbours *k*, distance metric (e.g. Euclidean).
- **K-means:** number of clusters *k*, initial initialization of centroids.
- **Isolation Forest:** number of trees, number of samples per tree.

The training process consists of the following steps:

- Splitting the data into a training set and a test set.
- Optimization of the parameters of the respective algorithm using cross-validation.

Performance evaluation using metrics such as anomaly detection accuracy, recall, F1 score or AUC.

4.3.1.4 Recommendations of classical ML methods for recognizing various attacks

We present selected types of attacks and classical ML methods suitable for recognizing these attacks; we also state the type of data that need to be prepared for these methods:

- DoS attacks – classification based on the number of requests per unit of time (Random Forest).
- Port scanning – detection of an unusual number of short TCP connections (K-means, DBSCAN).
- Insider threats – anomalous employee behaviour (Isolation Forest, PCA).
- Malware in the network – C2 server communication patterns (SVM, Decision Trees).
- Outage prediction – regression models or time series (e.g. ARIMA or simple ML regressions).

4.3.1.5 Criteria for comparing algorithms

The criteria in the following table are used to compare ML algorithms:

Table 2 Criteria for comparing ML algorithms

Criterion	Description
Detection accuracy	Percentage of correctly classified anomalies.
False alarms (false positives)	Number of normal events labelled as threats.
Time and computational demands	Speed of learning and detection, HW requirements.
Scalability	Possibility of deployment on large datasets.
Interpretability	Possibility of explaining the model's decisions.
Resistance to noise in data	Robustness to incomplete or erroneous data.
Requirements for labelled data	Degree of dependence on labelled datasets.

Table 2: Types of IP attacks

4.3.1.6 Recommended algorithm

Isolation Forest – is one of the most suitable algorithms for anomaly detection in both security and operational data, particularly because of its:

- ability to work without labelled data (unsupervised),
- low computational demands,
- excellent detection of outliers,
- robustness to noise.

For classification tasks with labelled data, we recommend Random Forest owing to its high accuracy, good generalization and ability to work even with incomplete data.

4.3.2 Machine learning algorithms using neural networks

This section analyses machine learning (ML) algorithms built on deep neural networks that are suitable for solving the task of detecting and predicting security and operational anomalies.

In an environment of dynamic and complex information system infrastructure, the identification and prediction of anomalies is key to ensuring its stability and security. Deep neural networks (DNN) represent an effective approach to solving these tasks owing to their ability to model non-linear dependencies and to learn representations from raw data. This document focuses on explaining the principles of using DNN in the field of detecting and predicting security and operational anomalies. It presents an analysis and design of suitable ML/AI algorithms, a description of the principle, input data sources, their preparation and the datasets used, and it states the model parameters and a description of the model training process.

4.3.2.1 Principle of deep learning algorithms

Algorithms based on deep learning rely on multi-layer neural network architectures (e.g. MLP, LSTM, autoencoders, CNN, RNN), which enable:

- Automatic learning of representations (models, patterns) from input data.
- Generalization from examples – the system learns from historical cases of normal and anomalous behaviour.
- Capturing temporal and spatial patterns – which is important particularly for prediction.

4.3.2.2 Principle of deep learning algorithms

The best-known models using DNN neural networks include:

- **Autoencoders**: They are trained on "normal" data and identify anomalies by a large reconstruction error.
- **LSTM** (Long Short-Term Memory): Effective in processing time series, useful in predicting operational outages.
- **CNN** (Convolutional Neural Networks): Can be used to analyse network flows represented in the form of images.
- **GANs** (Generative Adversarial Networks): For the synthesis and detection of complex types of anomalies.

4.3.2.3 Input data and their preparation

Typical sources of input data for training the above networks are:

- Network logs (NetFlow, PCAP, firewall logs)
- System logs (CPU, memory, I/O operations)
- Application logs (web server, databases)
- Metrics from monitoring systems (e.g. Prometheus, Nagios, Zabbix, Graphite)

Preparation of the input data before they are used for training includes:

- **Cleaning**: Removal of missing or irrelevant data.
- **Normalization/scaling**: Min-Max or Z-score transformation.
- **Feature extraction**: Creating aggregated metric windows and statistical characteristics.
- **Sequencing**: For models working with time series.
- **Labelling**: Supervised learning requires annotated data (normal vs. anomalous).

4.3.2.4 Algorithm parameters and training

During training, the model parameters can be selected and changed to achieve the best possible model results:

- **Number of layers and neurons**
- **Activation functions** (ReLU, Sigmoid, Tanh)
- **Optimization algorithm** (Adam, RMSprop)
- **Batch size and learning rate**
- **Dropout/masking** for regularization

The training process consists of the following steps:

- **Splitting the data into training, validation and test sets**
- **Model training** using error backpropagation
- **Hyperparameter tuning** (e.g. via Grid Search, Bayesian Optimization)
- **Model evaluation** using the metrics: accuracy, recall, F1-score, ROC-AUC

4.3.2.5 Recommendations of neural network methods for attack detection

We present selected types of attacks in computer networks and DNN-based ML methods that are suitable for recognizing these attacks:

- **DDoS attacks**: LSTM detects sudden spikes in requests.
- **Botnet activity**: An autoencoder identifies non-standard patterns in network communication.
- **Covert intrusions (APT)**: A combination of CNN and RNN for detecting long-term patterns.
- **SQL injection**: NLP models applied to HTTP requests can recognize syntax abuse.
- **Malware in the network**: GAN-based detection of malicious flows on the basis of behavioural patterns.

4.3.2.6 Criteria for comparing algorithms

The following criteria are used to compare ML algorithms:

- **Detection/prediction accuracy** (Precision, Recall, F1-score, Accuracy)
- **Latency and computational demands**
- **Possibility of online learning (real-time inference)**
- **Robustness to unlabelled or incomplete data**
- **Ability to work with imbalanced data**
- **Interpretability of results**
- **Resistance to attacks on the model (e.g. adversarial examples)**

4.3.2.7 Recommended algorithm

Depending on the use case, the following algorithms can be recommended:

- **Autoencoders** are an excellent choice for **unlabelled data** (unsupervised learning) and for simple anomaly detection.
- **LSTM networks** are suitable for **predicting anomalies in time series** and for processing sequences of events.
- **Hybrid models** (e.g. CNN-LSTM, Autoencoder-LSTM) combine the advantages of both approaches and are often the most effective today.

The most suitable algorithm:

Autoencoder-LSTM hybrid – owing to its ability to model normal behaviour, detect anomalies on the basis of the reconstruction error and at the same time track the temporal context. It has proven itself excellently in both network security and operational diagnostics.

5 ANALYSIS OF STATISTICAL PARAMETERS ON SELECTED DDOS ATTACKS

5.1 Description of the DDoS attack records used to test the response of statistical coefficients

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.

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.

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.

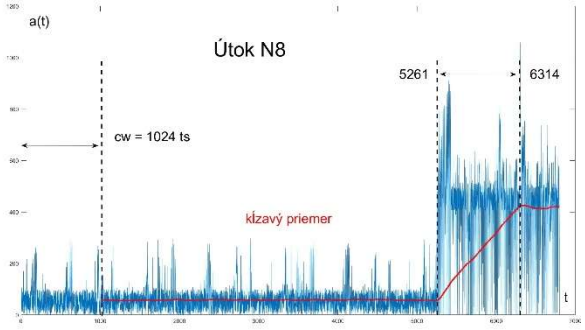


Fig. 15: Attack N8 and moving average

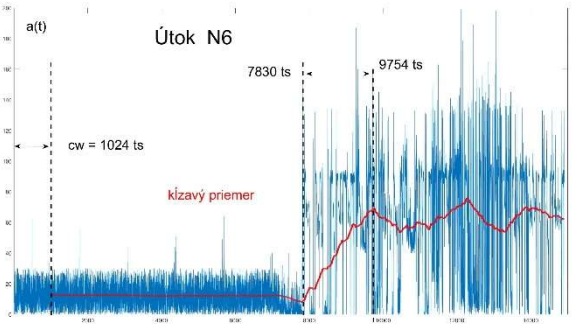


Fig. 16: Attack N6 and moving average

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

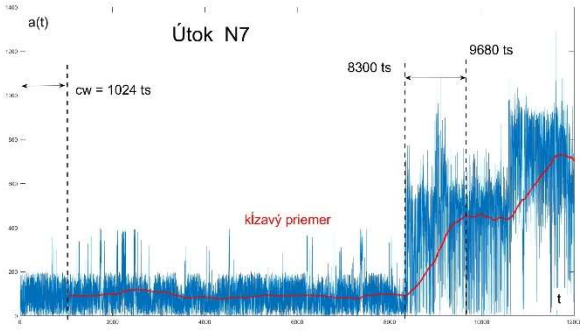


Fig. 17: Attack N7 and moving average

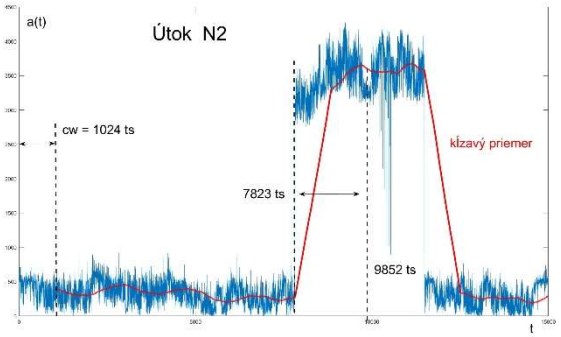


Fig. 18: Attack N2 and moving average

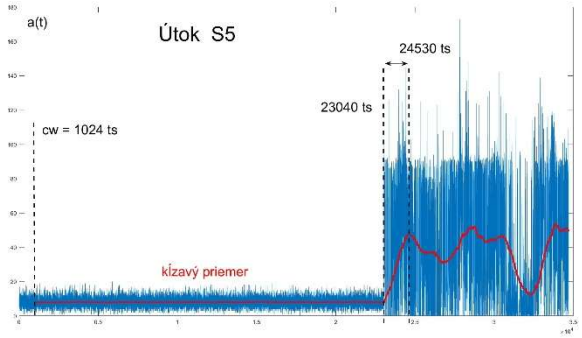


Fig. 19: Attack S5 and moving average

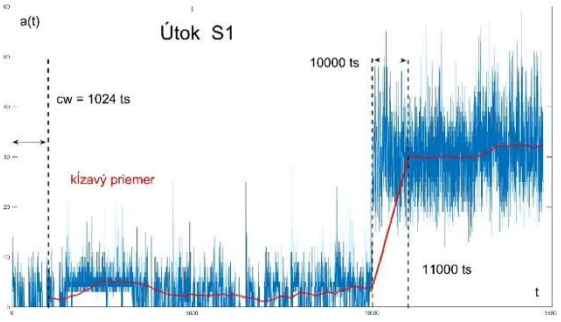


Fig. 20: Attack S1 and moving average

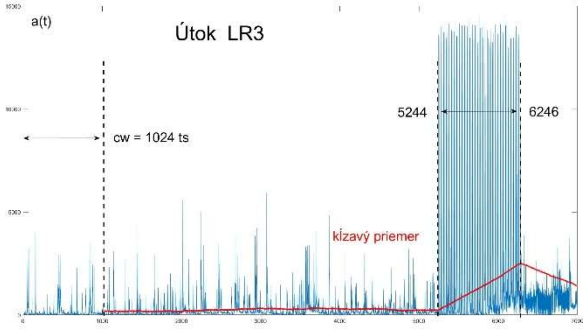


Fig. 21: Attack LR3 and moving average

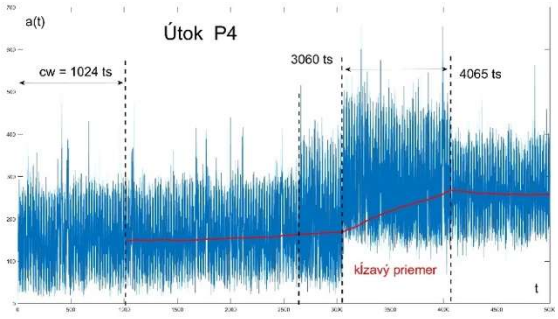


Fig. 22: Attack P4 and moving average

5.2 Use of statistical coefficients

5.2.1 Variability, skewness and kurtosis

First, we will deal with the basic statistical moments, or the coefficients of variability V(t), skewness K(t) and kurtosis Skw(t). The coefficient of variability represents the ratio of the standard deviation to the mean value of the increments in the given time window cw. Variability expresses the degree of dispersion of the increments a(i) around the mean value. In the first graph we display the course of the moving average and the standard deviation; the second graph shows the values of the coefficient of variability:

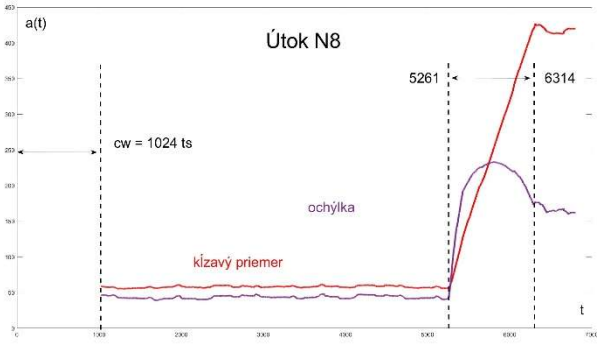


Fig. 23: Attack N8, mean and deviation

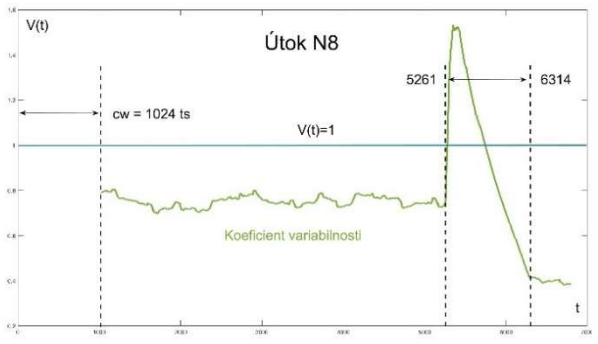


Fig. 24: Attack N8, coefficient of variability

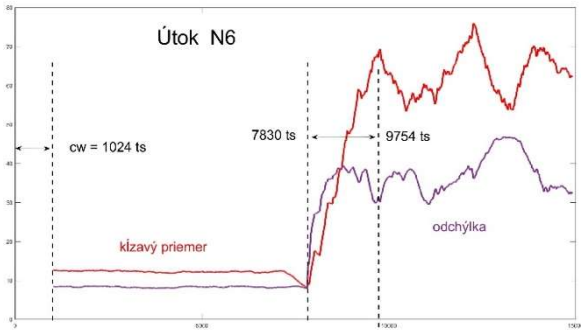


Fig. 25: Attack N6, mean and deviation

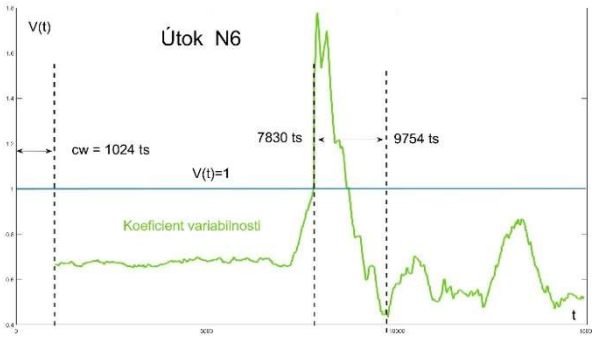


Fig. 26: Attack N6, coefficient of variability

In the case of standard DDoS attacks (N2,N6,N7,N8), the coefficient of variability responded in the same way. Standard traffic represented a stationary flow with a deviation smaller than the mean flow intensity. At the moment of the onset of the attack, the deviation responded with a faster increase in values than the linear growth of the moving average; therefore, the ratio of the deviation to the mean reached the value $V(t_A) = 1$ at the moment of the attack. This fact could be used for fast and simple detection of this type of attack. However, this is not a rule; in Fig. 27, for attack S1, we can see that stationarity is disrupted during standard traffic and the coefficient exceeded the value $V(t_A) = 1$ several times before the attack.

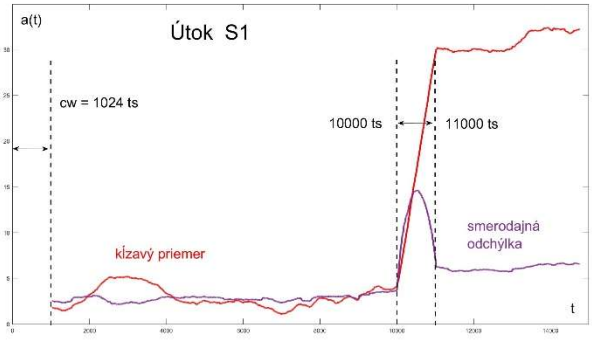


Fig. 27: Attack S1, mean and deviation

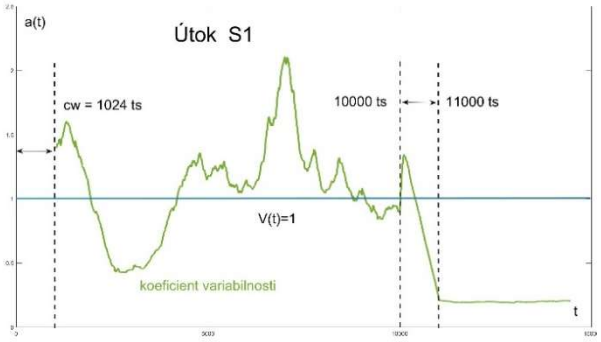


Fig. 28: Attack S1, coefficient of variability

Figs. 29 and 30 show graphs of the mean, deviation and coefficient of variability for the so-called Low Rate DDoS attack LR3:

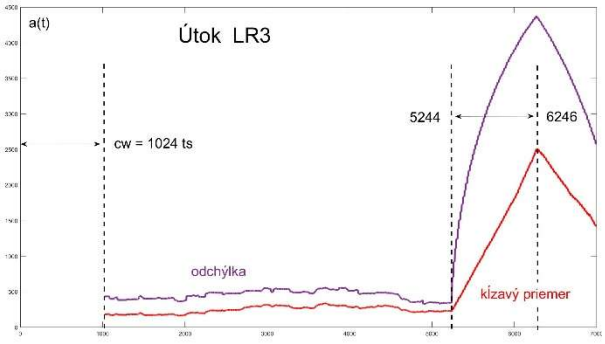


Fig. 29: Attack LR3, mean and deviation

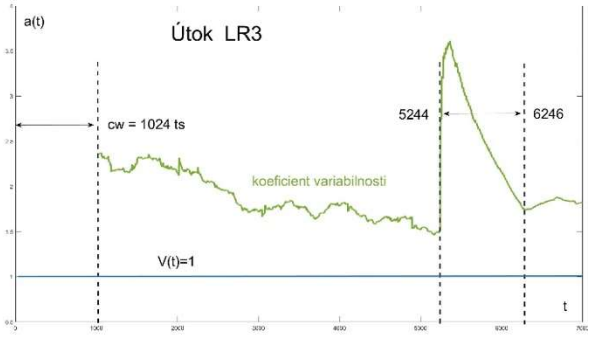


Fig. 30: Attack LR3, coefficient of variability

In the case of LR3, the deviation during normal traffic is greater than the mean flow intensity; therefore, the coefficient of variability is V(t)>1 throughout the entire course. The value 1 now cannot be used for fast detection. However, at the moment of the attack, the deviation values again grew faster than the linear growth of the mean intensity, which manifested itself in a marked step increase in the coefficient of variability. It should be possible to recognize this change using prediction methods (see the following chapter).

Other simple statistical parameters are the coefficient of skewness and the coefficient of kurtosis. These are in fact the third and fourth central moments scaled by the standard deviation:

$$\mu_3 = \frac{1}{\sigma^3} E[X - EX]^3 \qquad \mu_4 = \frac{1}{\sigma^4} E[X - EX]^4 \qquad \sigma^2 = E[X - EX]^2$$

In our case, the realizations of the variable X are the values of the increments a(t), and the coefficients describe the properties of the probability distribution of the increments in the given compute window cw. As the attack traffic was gradually loaded into the evaluated window cw, we assumed a marked change in the probability distribution and hence also a change in the coefficients. This assumption was confirmed for almost all of the analysed flows. Both

statistical coefficients responded with a marked increase in their values right at the start of the DDoS attack.

First, we present graphs of the coefficients applied to two selected standard attacks, followed by some problematic situations.

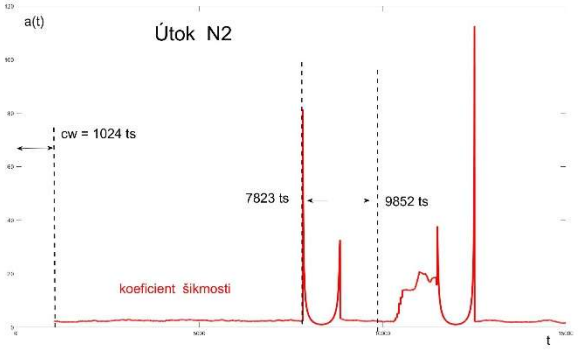


Fig. 31: Attack N2, coefficient of skewness

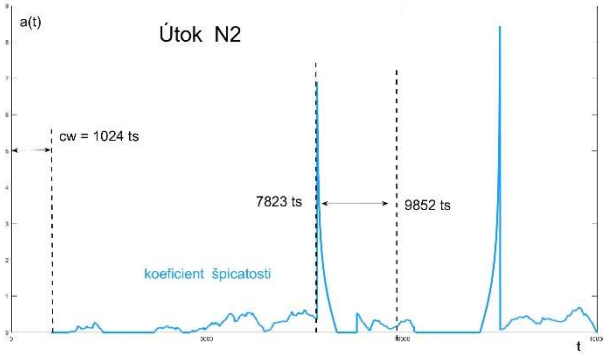


Fig. 32: Attack N2, coefficient of kurtosis

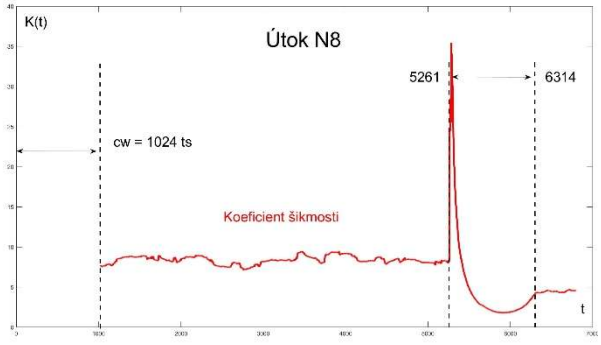


Fig. 33: Attack N8, coefficient of skewness

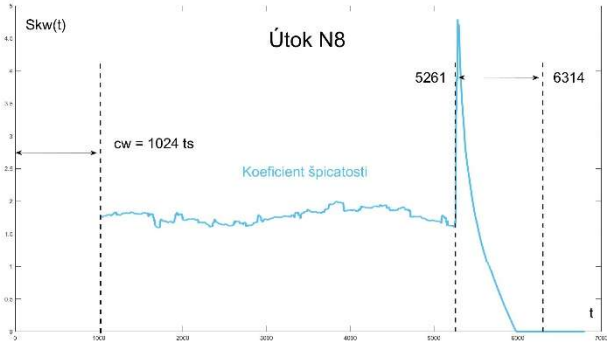


Fig. 34: Attack N8, coefficient of kurtosis

In standard type N attacks, the coefficients of skewness and kurtosis fluctuate only slightly during calculation on standard traffic owing to stationarity, and at the moment of the onset of the DDoS attack their values increase markedly. We assume that we will be able to recognize this marked jump automatically using suitable prediction methods.

A substantially less favourable situation occurred with non-standard attacks, e.g. S1. Owing to the non-stationarity of normal traffic and the frequent occurrence of high peaks, both coefficients oscillate strongly, as a result of which the jump at the onset of the attack is lost in the overall course of the values of both coefficients. In such a case, the onset of the DDoS attack cannot be recognized automatically.

To visualize the response of the coefficients to a change in the character of normal traffic and to the occurrence of peaks, we displayed the values of the coefficients in a common graph together with the increments of flow S1.

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

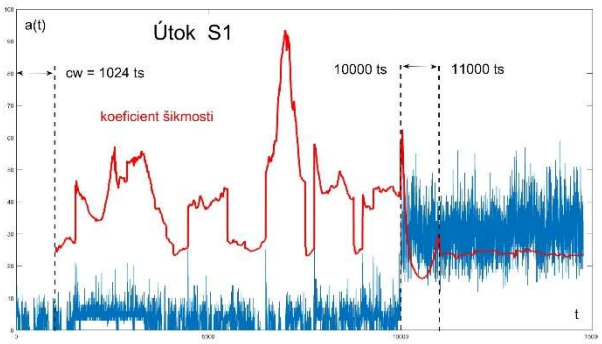


Fig. 35: Attack S1, coefficient of skewness

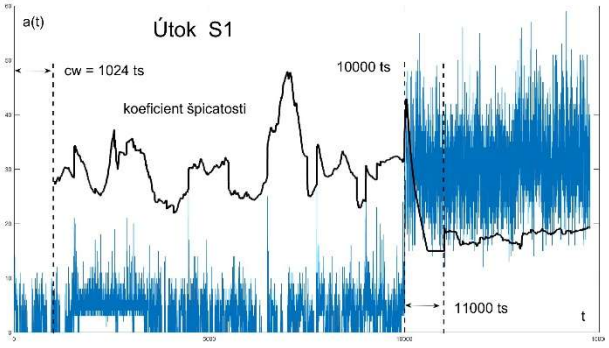


Fig. 36: Attack S1, coefficient of kurtosis

The response of the coefficients in the attack with low mean intensity LR3 turned out as positively as in the standard type N attacks:

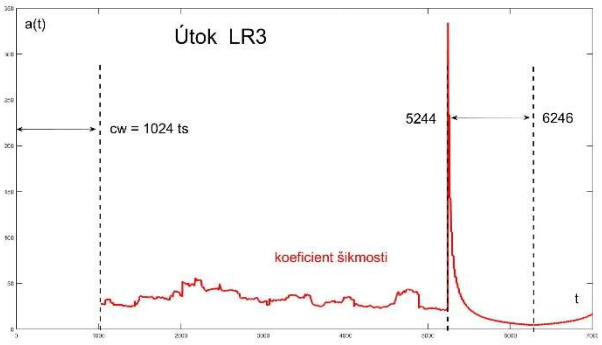


Fig. 37: Attack LR3, coefficient of skewnes

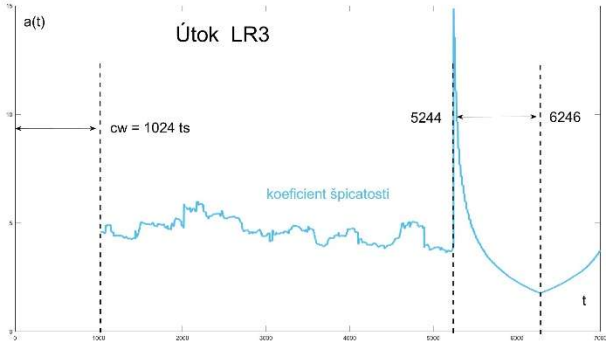


Fig. 38: Attack LR3, coefficient of kurtosis

Again, we can see the possibility of automatically distinguishing the marked increase in the values of the coefficients at the start of the attack from the values describing normal traffic.

5.2.2 Detection values for Autoregression and Correlation

The next two coefficients describe the autoregressive and correlation dependence between two consecutive cw windows, or between the random variables X(t) and X(t-1), whose realizations are the flow increments a(t) in the windows considered.

$$X(t) = c(t).X(t-1) + \varepsilon(t), \qquad \rho(t) = \frac{E[(X(t) - EX(t))(X(t-1) - EX(t-1))]}{\sigma(t)\sigma(t-1)}$$

A great advantage of these coefficients is that their values lie in the interval (-1,1). At the moment the attack was being loaded into the evaluated cw windows, for most attacks their values began to grow rapidly towards the upper limit of 1. For this reason, it is possible to set a certain limit value (threshold) for both coefficients, the exceeding of which would signal the start of a DDoS attack. The lower this value, the earlier the attack is signalled; on the other hand,

attacks that are too low may cause false alerts. On the basis of the experiments carried out, and in an effort to identify as many DDoS attacks as possible without false alerts, we set the threshold for the autoregressive coefficient to c(t) = 0.9 and for the correlation coefficient to ρ(t) = 0.75.

In the following graphs, we again displayed the evolution of the coefficients for attacks N8, N2, S1 and LR3. Attack N8 represents the behaviour of the coefficients in most standard attacks; in N2 we see a standard attack in which, however, the autocorrelation coefficient has all values greater than 0.9 and the detection cannot be used. In the non-standard attack S1, both coefficients exceeded the detection levels during the attack. For the low-intensity attack LR3, this method is unusable.

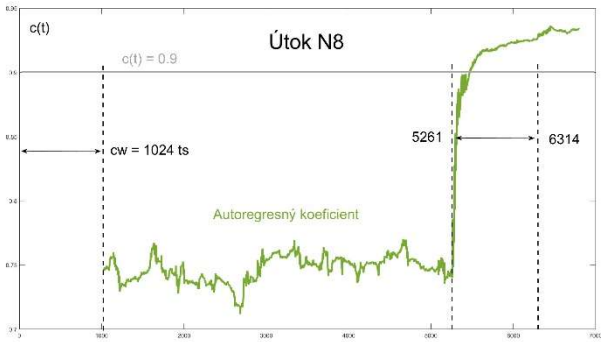


Fig. 39: Attack N8, Autoregressive coefficient

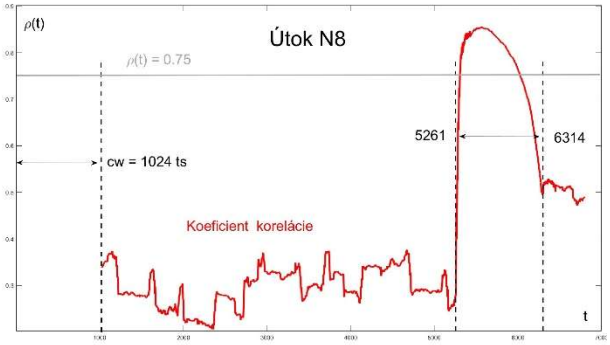


Fig. 40: Attack N8, Correlation coefficient

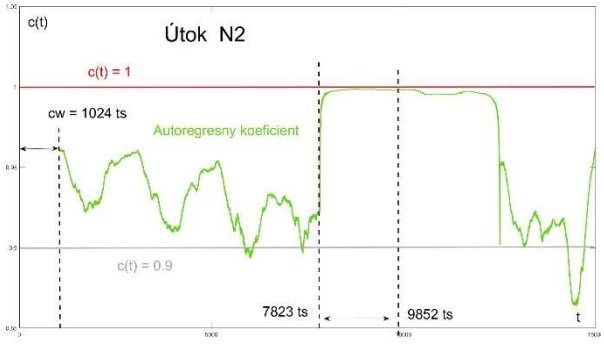


Fig. 41: Attack N2, Autoregressive coefficient

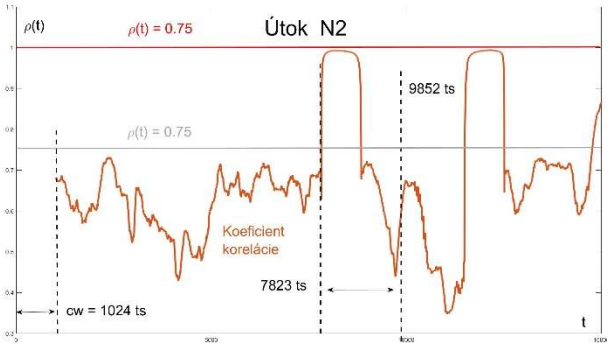


Fig. 42: Attack N2, Correlation coefficient

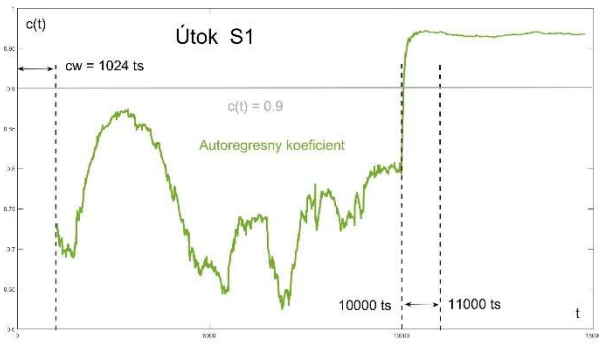


Fig. 43: Attack S1, Autoregressive coefficient

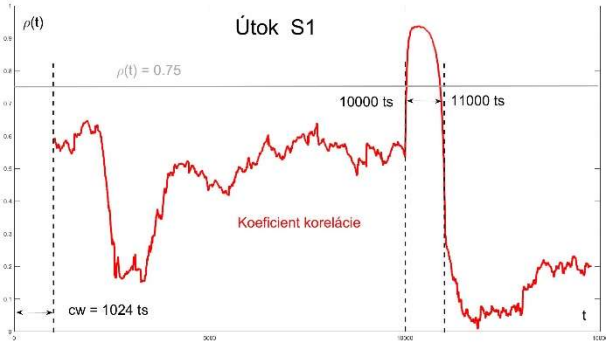


Fig. 44: Attack S1, Correlation coefficient

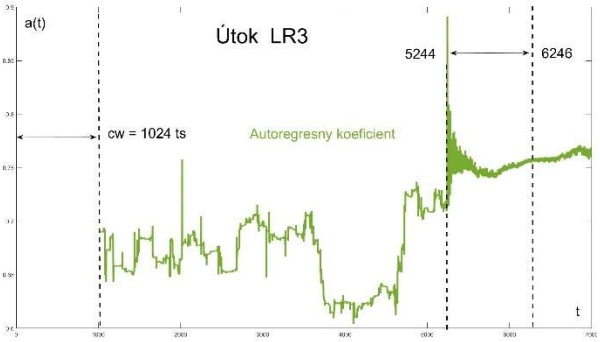


Fig. 45: Attack LR3, Autoregressive coefficient

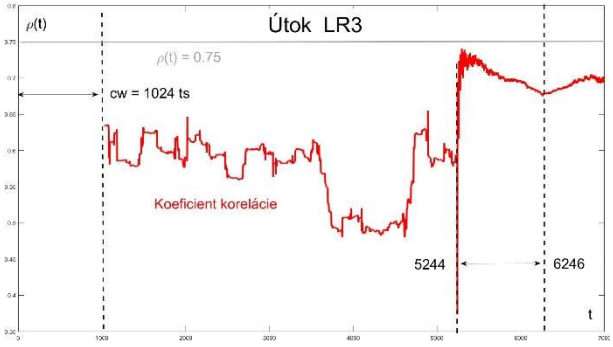


Fig. 46: Attack LR3, Correlation coefficient

For most attacks, it can be assumed that exceeding the detection values for autoregression and correlation can be used, in combination with other coefficients, for early simple detection of a flooding attack.

5.2.3 Response of Divergence to a DDoS attack

The last analysed parameter, the Kullback-Leibler Divergence, in a certain way compares the probability distribution P(.) of the increments a(t) in consecutive overlapping compute windows cw:

$$dvg(t) = \sum_{k=1}^{n} P(t-1) ln(\mathrm{P(t-1)/P(t)})$$

The advantage of Divergence is that, on the observed samples, in most cases it responded immediately to the onset of a DDoS attack with a high impulse (peak). The disadvantage is that it tended to respond in this way also to peaks in normal traffic, which could cause a large number of false alerts.

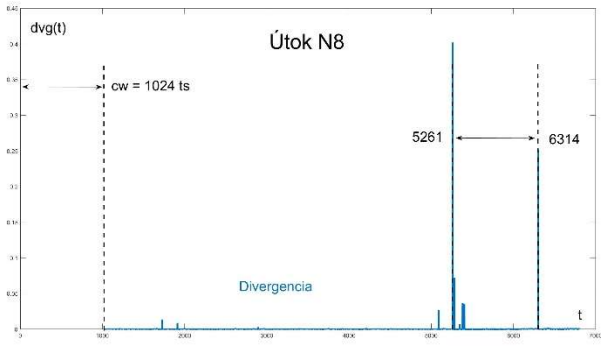


Fig. 47: Attack N8, Divergence

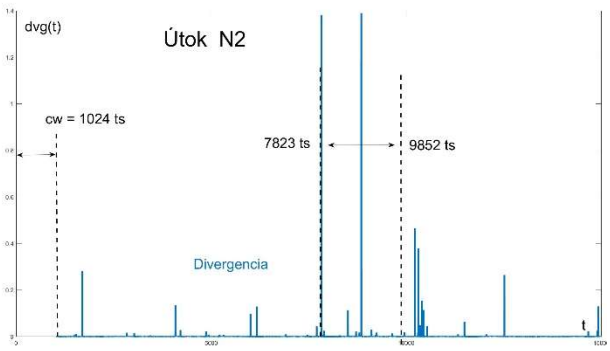


Fig. 48: Attack N2, Divergence

A further disadvantage of using Divergence is the fact that the high impulse at the onset of the attack lasts very briefly compared with the other parameters; we therefore consider its application in combination with the other parameters to be problematic. We have therefore postponed the use of Divergence to the future.

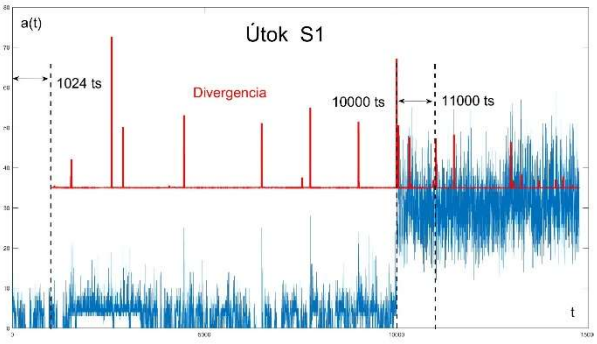


Fig. 49: Attack S1, Divergence

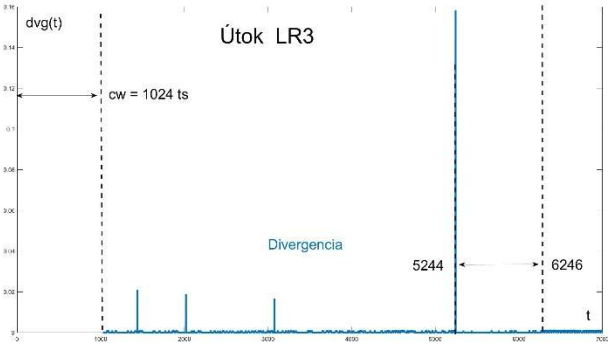


Fig. 50: Attack LR3, Divergence

5.3 Use of the 3σ-Tunnel for recognizing DDOS attacks

On the basis of experiments, we chose 1000 previous values for calculating the mean and the deviation, and we chose the interval width (EX-3σ, EX+3σ) (with a smaller interval size, frequent false alerts occurred). We called this method the prediction 3σ-tunnel.

A natural question arises as to why not apply the prediction tunnel directly to the IP flow increments a(t) and not bother with calculating statistical coefficients. After carrying out experiments, we found that although exceeding the upper limit of the tunnel detects the onset of a DDoS attack relatively early, false alarms are also reported frequently. When the multiple of the standard deviation was increased, the number of false alerts decreased, but so did the ability to detect the attack. In the following figures, we show only the upper limit of the 3σ-Tunnel on selected attacks.

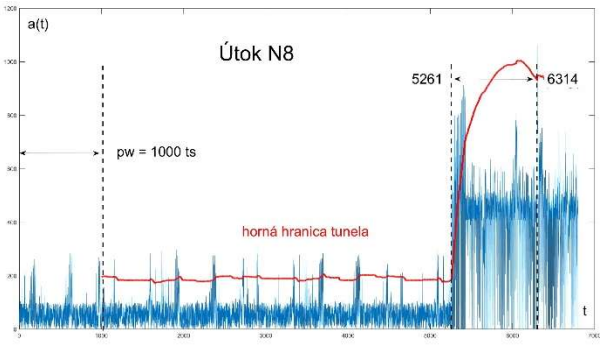


Fig. 51: Attack N8, upper limit of the 3σ-Tunnel

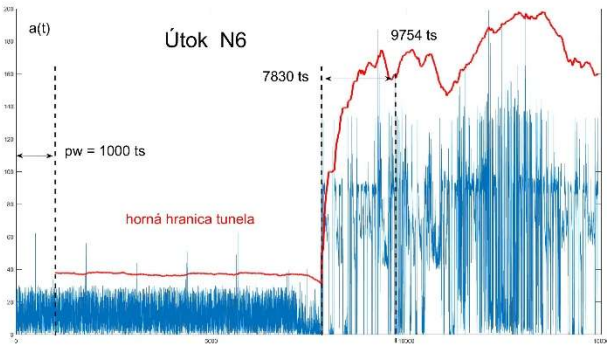


Fig. 52: Attack N6, upper limit of the 3σ-Tunnel

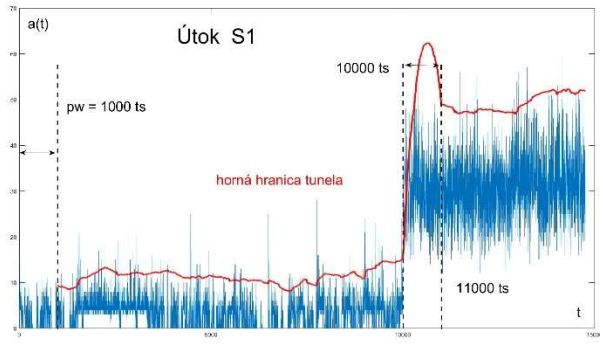


Fig. 53: Attack S1, upper limit of the 3σ-Tunnel

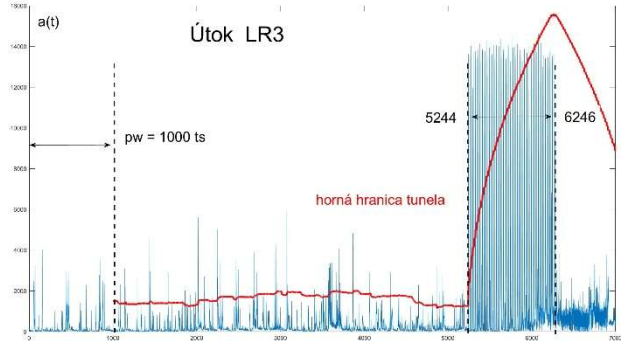


Fig. 54: Attack LR3, upper limit of the 3σ-Tunnel

In the following graphs, we will demonstrate the behaviour of the 3σ-Tunnel on the parameter with the greatest variability among the statistical coefficients considered, namely the Hurst exponent (estimated by RS statistics with removal of the AR1 autoregression).

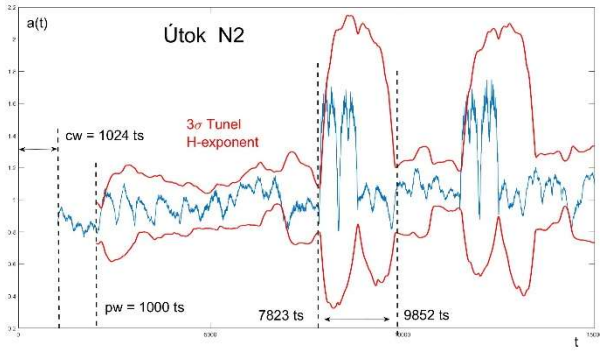


Fig. 55: N2, 3σ-Tunnel for the H-exponent

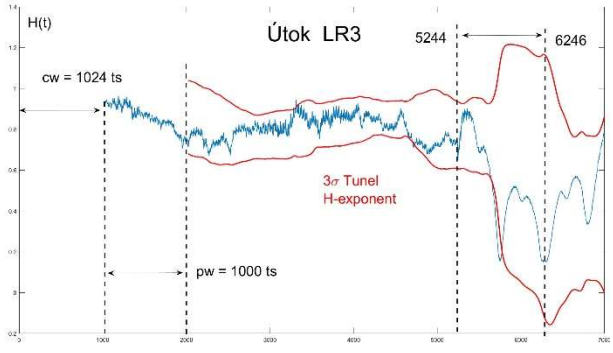


Fig. 56: LR3, 3σ-Tunnel for the H-exponent

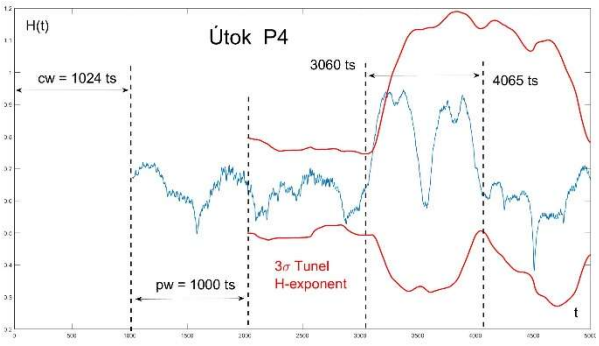


Fig. 57: P4, 3σ-Tunnel for the H-exponent

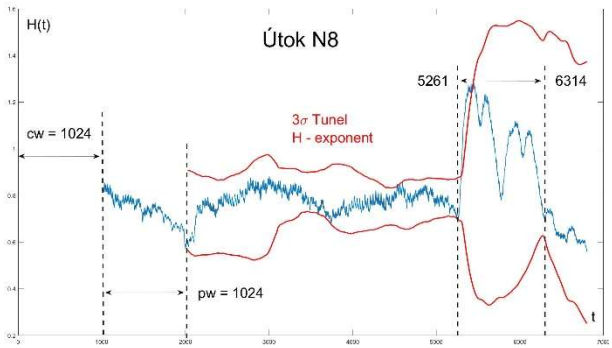


Fig. 58: N8, 3σ-Tunnel for the H-exponent

In most cases (except for attack LR3), the 3σ-Tunnel recognized the start of the DDoS attack early with a minimal, sometimes even zero, number of false alerts.

As already stated, on the basis of experiments we set the size of the prediction window to 1000 ts and the interval width for the 1001st value to three times the standard deviation. With this setting, in most experiments (various coefficients and various attacks) we achieved early recognition of the start of the attack with a minimal number of false alerts. In the preceding figures, this is 3 cases out of 4. A tunnel set up in this way has two useful properties:

1. It has a sufficiently "long memory" to be able to cover the local variability of the statistical coefficient and not cause a large number of false alerts
2. It has a sufficiently "long memory" so as not to be able to respond to a relatively large increase in the values of the investigated coefficient at the start of a DDoS attack (large relative to the relative variability)

Of course, in order to be able to draw more relevant conclusions and recommendations, it is necessary to carry out a number of experiments on records of various DDoS attacks.

5.4 Detection functions for automatic recognition of DDoS attacks

In the preceding chapters, we presented two ways of recognizing DDoS attacks using statistical coefficients. The first way was to determine detection values for suitable parameters: 1.00 for variability V(t) and the H-exponent, 0.90 for the autoregressive coefficient c(t) and 0.75 for the correlation coefficient ρ(t). The second way, using the prediction tunnel, was analysed in the preceding chapter.

When using threshold values, we introduced a two-valued logical function Υ[.], which evaluates the exceeding of the threshold value for a given statistical coefficient. In the following graphs, we show variability V(t), the autoregressive coefficient c(t) and the correlation coefficient ρ(t) and the course of their function Υ[.] for attacks S1a N2:

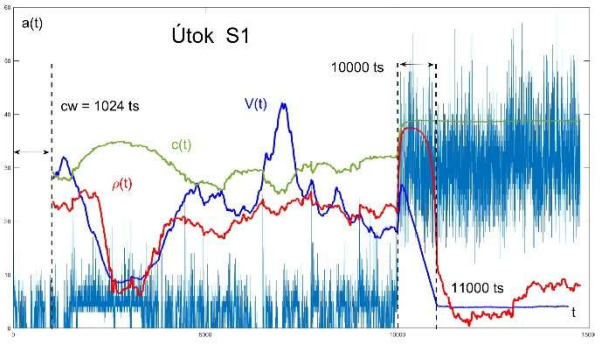


Fig. 59: S1, coefficients V(t), c(t), ρ(t)

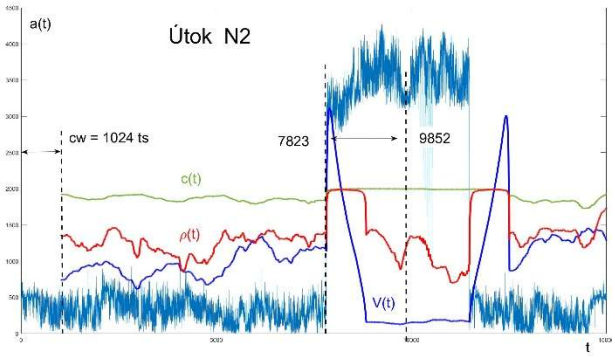


Fig. 60: N2, coefficients V(t), c(t), ρ(t)

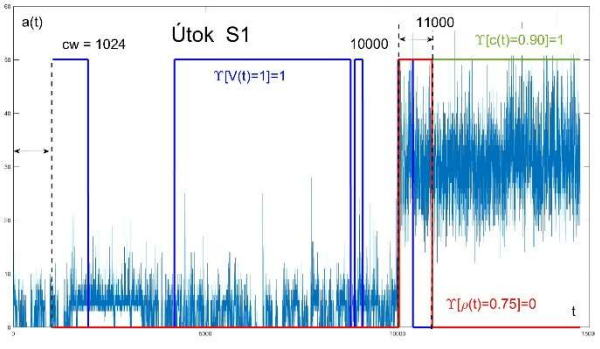


Fig. 61: S1, course of the functions Y[.]

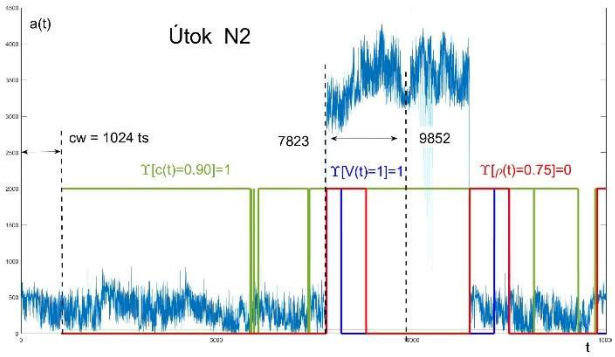


Fig. 62: N2, course of the functions Y[.]

The sets of values on which the coefficients exceed their detection limits will be called detection intervals. In Figs. 61 and 62, we can see how these intervals overlap one another. In order to eliminate false alerts and at the same time achieve timely recognition of attacks, we introduce a new detection function D(t) for the observed coefficients, where, for simplicity, we denote, e.g. for the coefficient V(t), the function Y[V(t)>1.00] as $Y_V(t)$:

$$D(t) = Y_V(t).Y_C(t) + Y_V(t).Y\rho(t) + Y_C(t).Y\rho(t)$$

The following figures show the course of the detection function for attacks S1 and N2:

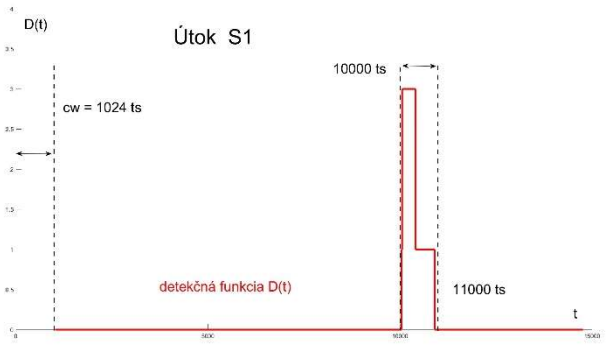


Fig. 63: S1, course of the detection function D(t)

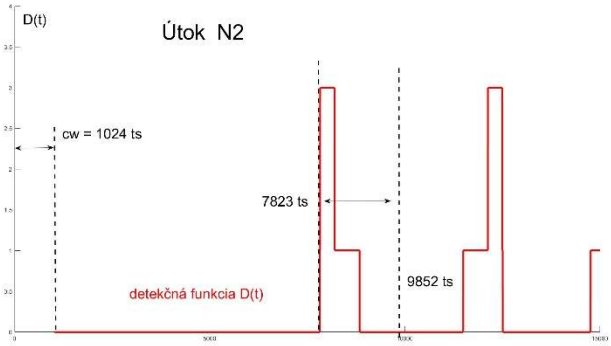


Fig. 64: N2, course of the detection function D(t)

Owing to the form of the function D(t), we have eliminated possible false alerts, and at the same time, on the basis of the values 1 or 3, we can recognize at what time two or all 3 statistical parameters detected the attack.

In Tab. 3, we present the evaluation of the use of the detection function D(t) for the analysed attacks. The first value represents the number of false alerts, the second determines the time when at least two coefficients detected the attack, and the third value indicates the time when all three statistical coefficients V(t), c(t) and ρ(t) detected the attack.

attack	S1	N2	LR3	S5	N6	N7	N8
D(t)	0 – 34 - 52	0 - 5 - 5	x	x	0 – 75 - 119	0 - 228 - 480	0 - 47 - 210

Tab. 3: Evaluation of DDoS attack recognition using the detection function D(t)

Attack recognition according to the threshold value was effective mainly for standard attacks, while in several cases two coefficients exceeded their threshold values substantially earlier than all three. In non-standard attacks such as LR3, P4 and S5, no two detection intervals, if they arose at all, had a common intersection, and therefore the attack was not recognized

We will apply the same methodology to attack recognition using the prediction tunnel. In the preceding chapter, taking into account the attack recognition time and the number of false alerts, we determined the three most effective statistical coefficients: skewness K(t), kurtosis Skw(t) and variability V(t). Since we selected coefficients whose values increase markedly at the onset of a DDoS attack, we will deal only with the upper limit of the prediction tunnel. We evaluate the time for which the upper limit is exceeded as a detection interval and describe it by a two-valued function Γ[.], where, e.g. for the coefficient of kurtosis, we denote $\Gamma_K(t) = \Gamma[K(t)>Hr(t)]$, where Hr(.) is the upper limit of the prediction interval for the coefficient K(.).

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

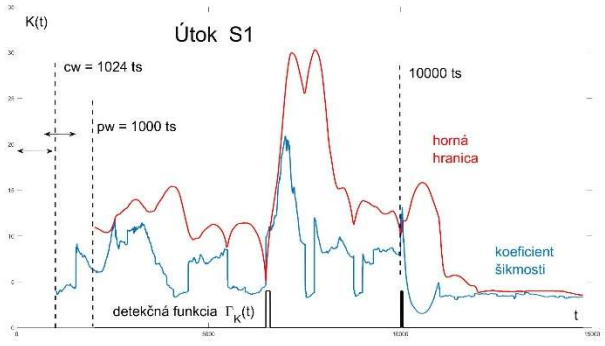


Fig. 65: S1, course of the function Γ(t)

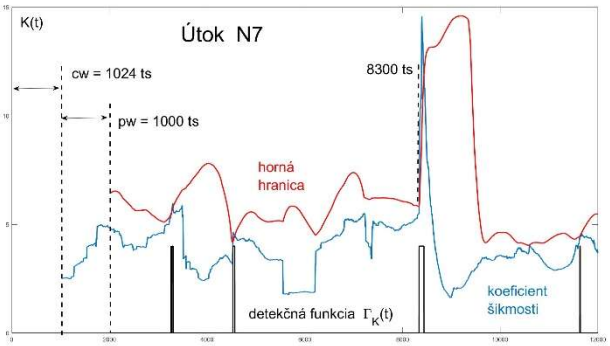


Fig. 66: N7, course of the function Γ(t)

Similarly to the previous method, recognition by means of a threshold value, we now also introduce a detection function that will be a combination of the identifiers Γ(t):

$$D_P(t) = \Gamma_K(t).\Gamma_{Skw}(t) + \Gamma_K(t).\Gamma_V(t) + \Gamma_{Skw}(t).\Gamma_V(t)$$

In Table 4, we present the evaluation of the use of the detection function $D_P(t)$.

attack	S1	N2	LR3	S5	N6	N7	N8
$D_P(t)$	1 – 6 - 28	0 - 1 - 1	1 - 1 - 1	1 - 18 - 41	1 - 12 - 12	1 - 39 - 49	0 - 1 - 1

Tab. 4: Evaluation of DDoS attack recognition using the detection function D(t)

5.5 Conclusion

On the basis of the experiments carried out so far, we see that the use of the selected statistical coefficients has substantially greater potential for the fast recognition of DDoS attacks (*FAD – Fast Attack Detection*) than standard monitoring of the average and peak intensity. In a DDoS attack, there is a strong assumption that these intensities will increase markedly. However, the average intensity grows only linearly depending on the size of the sliding window, which leads to late detection of the attack, and recognizing an attack by setting some threshold for the peak rate may lead to frequent false alerts owing to the occurrence of peaks in the IP flow.

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

By comparing Tables 3 and 4, we see that MPT recognized every type of attack considered, and substantially earlier than MDH; however, for most attacks it signalled one false alert. We

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

assume that this problem could be eliminated in several ways, for example by adding a further suitable detection coefficient, but by some type of smoothing of the coefficient values before applying the prediction tunnel, e.g. exponential or Holt smoothing.

ANNEXES

File: V7 Príloha (KPB3) – Analýza dostupných datasetov.docx