Thesis of Collective Project

Network Intrusion Detection using Machine Learning

Authored by :

Mr. **BILAL DAR RADI**	GSTR2	- ENSAT
Mrs. **MARIAM CHARAFI**	GSTR2	- ENSAT
Mr. **OUSSAMA BOUZAFFOUR**	GSTR2	- ENSAT
Mr. **YAHYA DENNOUNE**	GSTR2	- ENSAT

Pedagogical tutor:

Mr. **YOUSSEF SBAYTRI**

Abdelmalek Essaâdi University

National School of Applied Sciences Tangier

Class of : 2023/2024

List of Figures

Contents

Chapter 1

Generality of information security

What is IDS ?

Intrusion Detection System (IDS) is a network security technology originally built for detecting vulnerability exploits against a target application or computer. The IDS is also a listen-only device. The IDS monitors traffic and reports results to an administrator. It cannot automatically take action to prevent a detected exploit from taking over the system.

Attackers are capable of exploiting vulnerabilities quickly once they enter the network. Therefore, the IDS is not adequate for prevention. Intrusion detection and intrusion prevention systems are both essential to security information and event management.

1.1 How IDS Works :

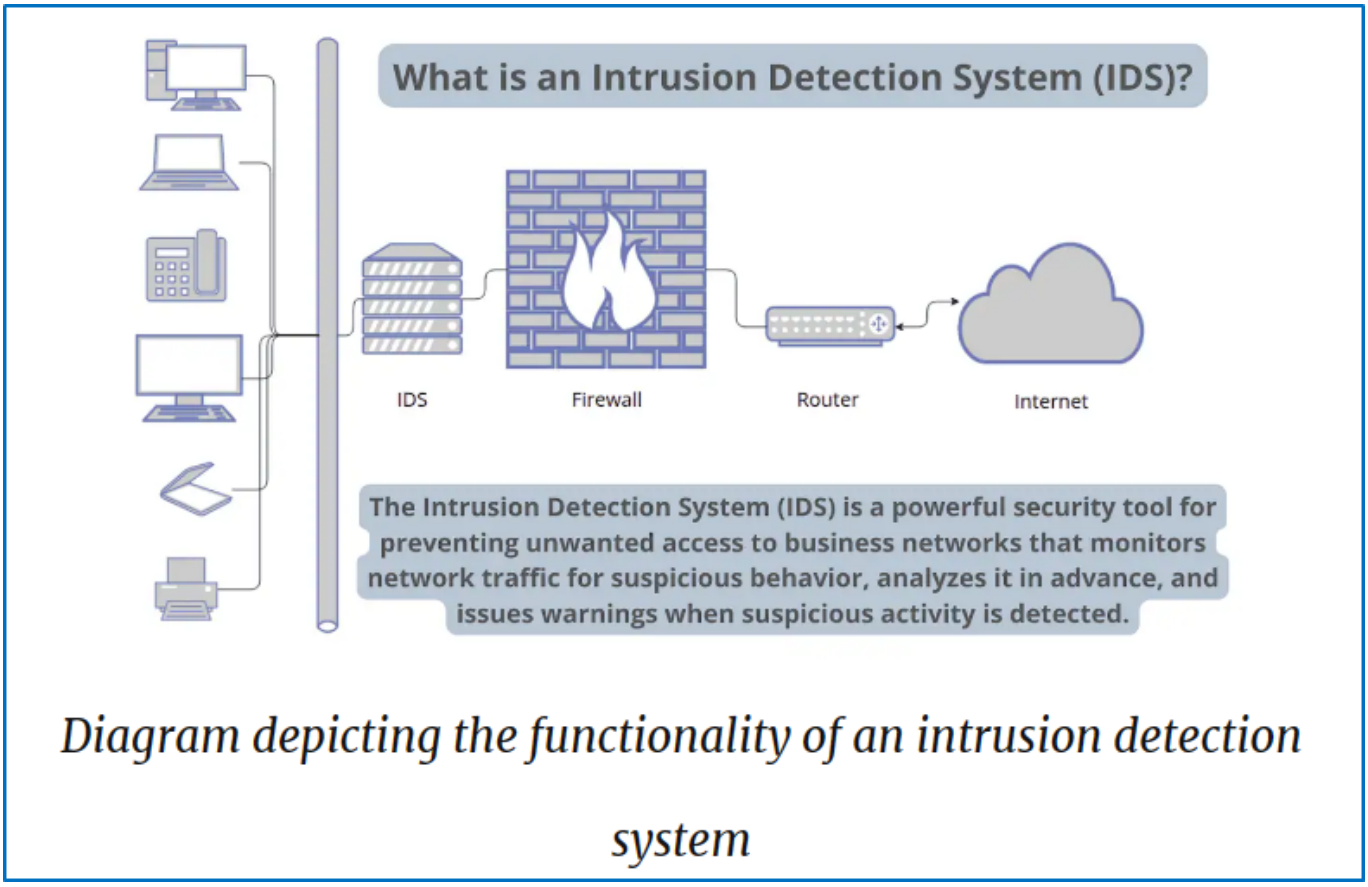


Diagram depicting the functionality of an intrusion detection system

An IDS only needs to detect potential threats. It is placed out of band on the network infrastructure. Consequently, it is not in the real-time communication path between the sender and receiver of information.

IDS solutions often take advantage of a TAP (Test Access Point) or SPAN (Switched Port Analyzer) port to analyze a copy of the inline traffic stream. This ensures that the IDS does not impact inline network performance.

When IDS was developed, the depth of analysis required to detect intrusion could not be performed quickly enough. The speed would not keep pace with components on the direct communications path of the network infrastructure.

Network intrusion detection systems are used to detect suspicious activity to catch hackers before damage is done to the network. There are network-based and host-based intrusion detection systems. Host-based IDSes are installed on client computers; network-based IDSes are on the network itself.

An IDS works by looking for deviations from normal activity and known attack signatures. Anomalous patterns are sent up the stack and examined at protocol and application layers. It can detect events like DNS poisonings, malformed information packets and Christmas tree scans.

An IDS can be implemented as a network security device or a software application. To protect data and systems in cloud environments, cloud-based IDSes are also available.[14]

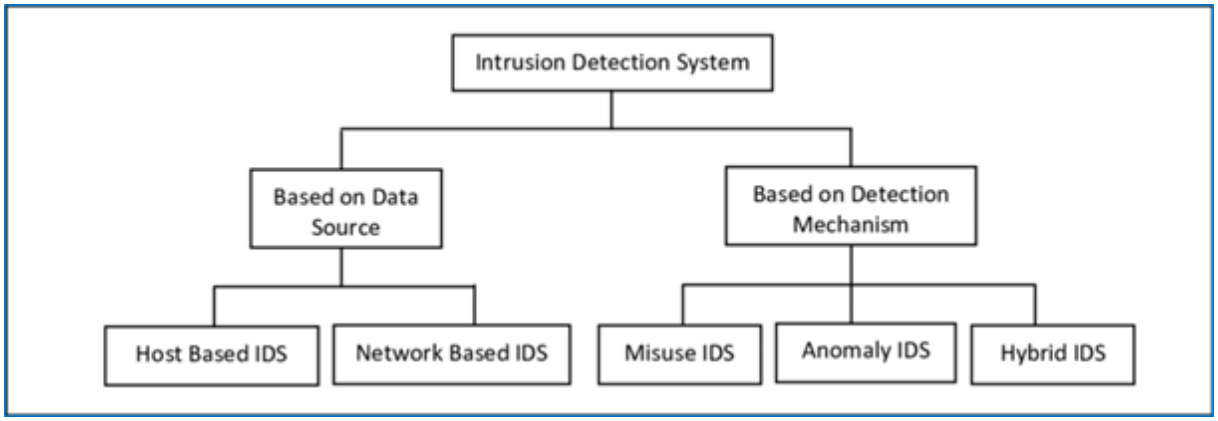
An IDS is composed of the following three components:

Sensors: which sense the network traffic or system activity and generate events.

Console: to monitor events and alerts and control the sensors.

Detection Engine: that records events logged by the sensors in a database and uses a system of rules to generate alerts from the received security events.

1.2 Types of Detection Systems :



There are several types of IDS that can be deployed to aid security administrators in their endeavors. Two types, network-based intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS) are most prevalent in modernsecurity deployments.

There are other types of IDS, however, which include file integrity and log file checkers, and decoy devices known as honeypots. Additionally, there exist hybrid systems that combine some of the different functionalities mentioned earlier. We'll discuss each of these IDS in this section.[15]

1.2.1 Network intrusion detection system (NIDS):

Network-based intrusion detection systems (NIDS) are devices intelligently distributed within networks that passively inspect traffic traversing the devices on which they sit. NIDS can be hardware or software-based systems and, depending on the manufacturer of the system, can attach to various network mediums such as Ethernet, FDDI, and others. Oftentimes, NIDS have two network interfaces. One is used for listening to network conversations in promiscuous mode and the other is used for control and reporting.

With the advent of switching, which isolates uni-cast conversations to ingress and egress switch ports, network infrastructure vendors have devised port mirroring techniques to replicate all network traffic to the NIDS. There are other means of supplying traffic to the IDS such as network taps. Cisco uses Switched Port Analyzer (SPAN) functionality to facilitate this capability on their network devices and, in some network equipment, includes NIDS components directly within the switch.

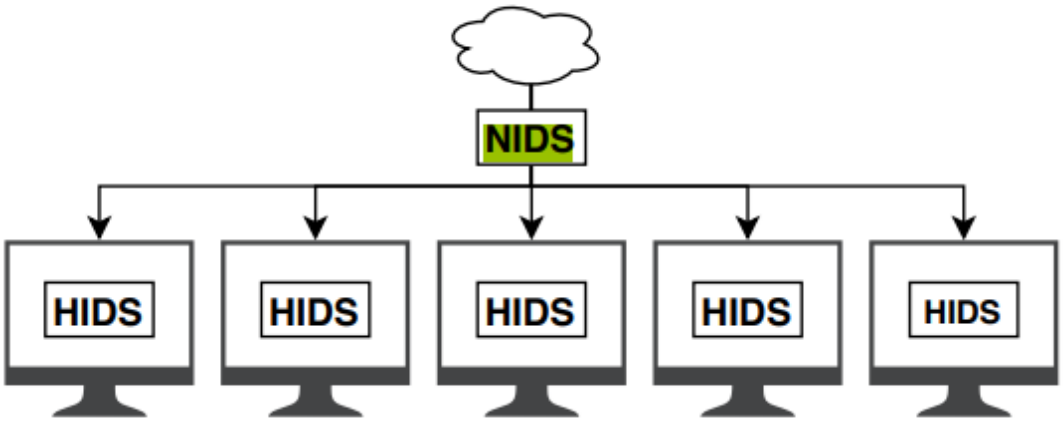
While there are many NIDS vendors, all systems tend to function in one of two ways; NIDS are either signature-based or anomaly-based systems. Both are mechanisms that separate benign traffic from its malicious brethren. Potential issues with NIDS include high-speed network data overload, tuning difficulties, encryption, and signature development lag time. We'll cover how IDS work and the difficulties involved with them later in this section.

Second description:

Network intrusion detection systems (NIDSs) monitor inbound and outbound traffic to devices across the network. NIDS are placed at strategic points in the network, often immediately behind firewalls at the network perimeter so that they can flag any malicious traffic breaking through.

NIDS may also be placed inside the network to catch insider threats or hackers who hijacked user accounts. For example, NIDS might be placed behind each internal firewall in a segmented network to monitor traffic flowing between subnets.[16]

To avoid impeding the flow of legitimate traffic, a NIDS is often placed "out-of-band," meaning that traffic doesn't pass directly through it. A NIDS analyzes copies of network packets rather than the packets themselves. That way, legitimate traffic doesn't have to wait for analysis, but the NIDS can still catch and flag malicious traffic.



1.2.2 Host intrusion detection system (HIDS):

First description:

HIDS; are installed on a specific endpoint, like a laptop, router, or server. The HIDS only monitors activity on that device, including traffic to and from it. A HIDS typically works by taking periodic snapshots of critical operating system files and comparing these snapshots over time. If the HIDS notices a change, such as log files being editd or configurations being altered, it alerts the security team.[16]

Security teams often combine network-based intrusion detection systems and host-based intrusion detection systems. The NIDS looks at traffic overall, while the HIDS can add extra protection around high-value assets.

A HIDS can also help catch malicious activity from a compromised network node, like ransomware spreading from an infected device.

Second description:

host-based intrusion detection systems (HIDS) are systems that sit at service end-points rather than in the network transit points like NIDS. The first type of IDS that's widely implemented, Host IDS, is installed on servers and is more focused on analyzing the specific operating system and application functionality residing on the HIDS host. HIDS are often critical in detecting internal attacks directed towards an organization's servers such as DNS, mail, and web servers. HIDS can detect a variety of potential attack situations such as file permission changes and improperly formed client-server requests.

File Integrity and Log File Checkers

File integrity and log file checking agents are a form of HIDS that focus on the operating systems binary files and the log files normally produced by OS-based security mechanisms such as login logs. File integrity software systems are best installed immediately after operating system installation. The software creates a local database and MD5 hashes of operating system binaries and configuration files. Should system binaries or other files change in any way, nightly processes that compare current hashes against original file hashes will detect the change and alert administrators. Log file checkers run regularly as well and parse system and application logs to search for signature-based alerts. For instance, multiple failed logins on a server would typically be detected and reported by log-checking software.

1.2.3 Others:

While Host IDS and Network IDS are the most commonly deployed forms of IDS, other types of IDS such as Hybrid IDS and honeypots can be useful tools in detecting potential security situations.

Hybrid IDS:

Hybrid IDS are systems that combine both Host IDS and limited Network IDS functionality on the same security platform. A Hybrid IDS can monitor system and application events and verify a file system's integrity like Host IDS, yet because the monitoring network interface runs in a non-promiscuous mode, the Network IDS functionality only serves to analyze traffic destined for the device itself. A Hybrid IDS is often deployed on an organization's most critical servers.

While NIDS and HIDS are the most common, security teams can use other IDSs for specialized purposes. A **Protocol-based IDS (PIDS)** monitors connection protocols between servers and devices. PIDS are often placed on web servers to monitor HTTP or HTTPS connections.

An **Application protocol-based IDS (APIDS)** works at the application layer, monitoring application-specific protocols. An APIDS is often deployed between a web server and an SQL database to detect SQL injections.

		Advantages	Disadvantages	Data source
Technology	HIDS	• HIDS can check end-to-end encrypted communications behaviour. • No extra hardware required. • Detects intrusions by checking hosts file system, system calls or network events. • Every packet is reassembled • Looks at the entire item, not streams only	• Delays in reporting attacks • Consumes host resources • Needs to be installed on each host. • It can monitor attacks only on the machine where it is installed.	• Audits records, log files, Application Program Interface (API), rule patterns, system calls.
	NIDS	•Detects attacks by checking network packets. •Not required to install on each host. •Can check various hosts at the same period. •Capable of detecting the broadest ranges of network protocols	•Challenge is to identify attacks from encrypted traffic. •Dedicated hardware is required. •It supports only identification of network attacks. •Difficult to analysis high-speed network. •The most serious threat is the insider attack.	•Simple Network Management Protocol (SNMP) •Network packets (TCP/UDP/ICMP), •Management Information Base (MIB) •Router NetFlow records

Figure 1.1: Summary of comparisons between HIDS and NIDS

1.3 How intrusion detection systems work

1.3.1 Signature-based intrusion detection system (SIDS):

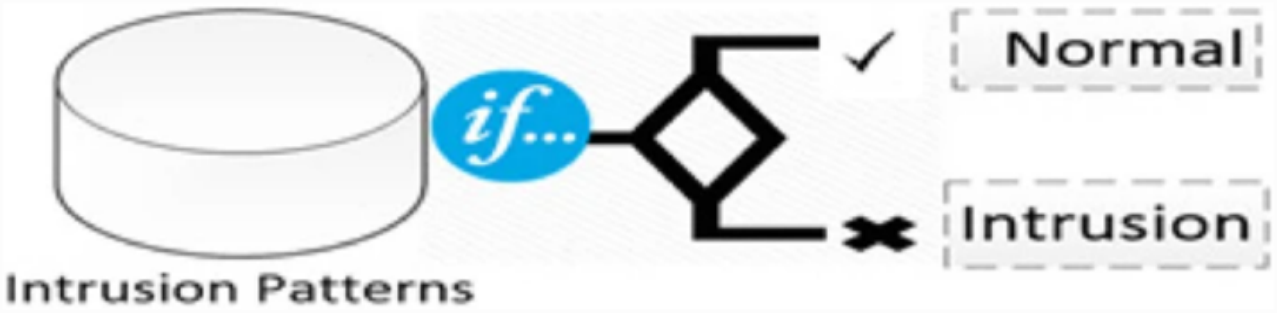
Signature intrusion detection systems (SIDS) are based on pattern matching techniques to find a known attack; these are also known as Knowledge-based Detection or Misuse Detection (Khraisat et al., 2018). In SIDS, matching methods are used to find a previous intrusion. In other words, when

an intrusion signature matches with the signature of a previous intrusion that already exists in the signature database, an alarm signal is triggered. For SIDS, host's logs are inspected to find sequences of commands or actions which have previously been identified as malware. SIDS have also been labelled in the literature as Knowledge-Based Detection or Misuse Detection (Modi et al., 2013) [17].

The figure below demonstrates the conceptual working of SIDS approaches. The main idea is to build a database of intrusion signatures and to compare the current set of activities against the existing signatures and raise an alarm if a match is found.

For example, a rule in the form of "if: antecedent -then: consequent" may lead to "if (source IP address=destination IP address) then label as an attack ".

Fig. 1



Conceptual working of SIDS approaches

SIDS usually gives an excellent detection accuracy for previously known intrusions (Kreibich & Crowcroft, 2004). However, SIDS has difficulty in detecting zero-day attacks for the reason that no matching signature exists in the database until the signature of the new attack is extracted and stored. SIDS are employed in numerous common tools, for instance, Snort (Roesch, 1999) and NetSTAT (Vigna & Kemmerer, 1999). Traditional approaches to SIDS examine network packets and try matching against a database of signatures. But these techniques are unable to identify attacks that span several packets. As modern malware is more sophisticated it may be necessary to extract signature information over multiple packets. This requires the IDS to recall the contents of earlier packets. With regards to creating a signature for SIDS, generally, there have been a number of methods where signatures are created as state machines (Meiners et al., 2010), formal language string patterns or semantic conditions (Lin et al., 2011).

The increasing rate of zero-day attacks (Symantec, 2017) has rendered SIDS techniques progressively less effective because no prior signature exists for any such attacks. Polymorphic variants of the malware and the rising amount of targeted attacks can further undermine the adequacy of this traditional

paradigm. A potential solution to this problem would be to use AIDS techniques, which operate by profiling what is an acceptable behavior rather than what is anomalous, as described in the next section.

1.3.2 Anomaly-based intrusion detection system (AIDS):

Network behavior is the major parameter on which the anomaly detection systems rely upon. If the network behavior is within the predefined behavior, then the network transaction is accepted or else it triggers the alert in the anomaly detection system.

Acceptable network performance can be either predetermined or learned through specifications or conditions defined by the network administrator. The crucial stage of behavior determination is regarding the ability of detection system engine toward multiple protocols at each level. The IDS engine must be able to understand the process of protocols and its goal. Despite the fact that the protocol analysis is very expensive in terms of computation, the benefits like increasing rule set assist in lesser levels of false-positive alarms. Defining the rule sets is one of the key drawbacks of anomaly-based detection.[18]

The efficiency of the system depends on the effective implementation and testing of rule sets on all the protocols. In addition, a variety of protocols that are used by different vendors impact the rule defining the process. In addition to the aforesaid, custom protocols also add complexity to the process of rule defining. For accurate detection, the administration should clearly understand the acceptable network behavior. However, with strong incorporation of rules and protocol, the anomaly detection procedure would likely to perform more efficiently.

However, if the malicious behavior falls under the accepted behavior, in such conditions it might get unnoticed. The major benefit of the anomaly-based detection system is about the scope for detection of novel attacks. This type of intrusion detection approach could also be feasible, even if the lack of signature patterns matches and also works in the condition that is beyond regular patterns of traffic.

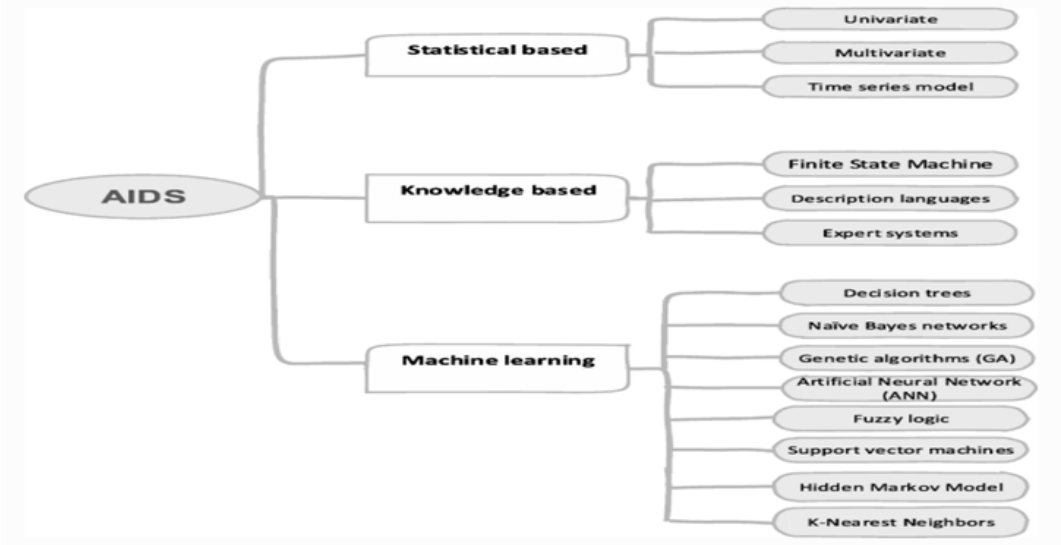


Figure 1.2: Classification of AIDS Methods :

Detection Source			HIDS	NIDS	Capability
Detection methods	SIDS		Wagner and Soto (2002)	Hubballi and Suryanarayanan (2014)	P
	AIDS	Statistics based	Ara, Louzada & Diniz (2017)	Tan, et al. (2014); Camacho, et al. (2016)	Z
		Knowledge-based	Mitchell and Chen (2015) Creech and Hu (2014b)	Hendry and Yang (2008) Shakshuki, et al. (2013) Zargar, et al. (2013)	
		Machine learning	Du, et al. (2014) Wang, et al. (2010)	Elhag, et al. (2015); Kim, et al. (2014); Hu, et al. (2014)	
	SIDS+ AIDS		Alazab, et al. (2014); Stavroulakis and Stamp (2010); Liu, et al. (2015)		P + Z

Figure 1.3: Comparisons of IDS technology types, using examples from the literature. "P" indicates pre-defined attacks and "Z" indicates zero-day attacks

Others:

- **Perimeter intrusion detection system (PIDS):** A PIDS solution is placed on a network to detect intrusion attempts taking place on the perimeter of organizations' critical infrastructures.
- **Virtual machine-based intrusion detection system (VMIDS):** A VMIDS solution detects intrusions by monitoring virtual machines. It enables organizations to monitor traffic across all the devices and systems that their devices are connected to.
- **Stack-based intrusion detection system (SBIDS):** SBIDS is integrated into an organization's Transmission Control Protocol/Internet Protocol (TCP/IP), which is used as a communications protocol on private networks. This approach enables the IDS to watch packets as they move

through the organization's network and pulls malicious packets before applications or the operating system can process them.[19]

1.4 Cybersecurity Threats and attacks:

Network Intrusion Detection using Deep Learning Lakshit Sama Master of Research In the simplest sense, a cybersecurity threat, or cyberthreat, is an indication that a hacker or malicious actor is attempting to gain unauthorized access to a network for the purpose of launching a cyberattack.

Cyberthreats can range from the obvious, such as an email from a foreign potentate offering a small fortune if you'll just provide your bank account number, to the deviously stealthy, such as a line of malicious code that sneaks past cyberdefenses and lives on the network for months or years before triggering a costly data breach. The more security teams and employees know about the different types of cybersecurity threats, the more effectively they can prevent, prepare for, and respond to cyberattacks.[20] [21]

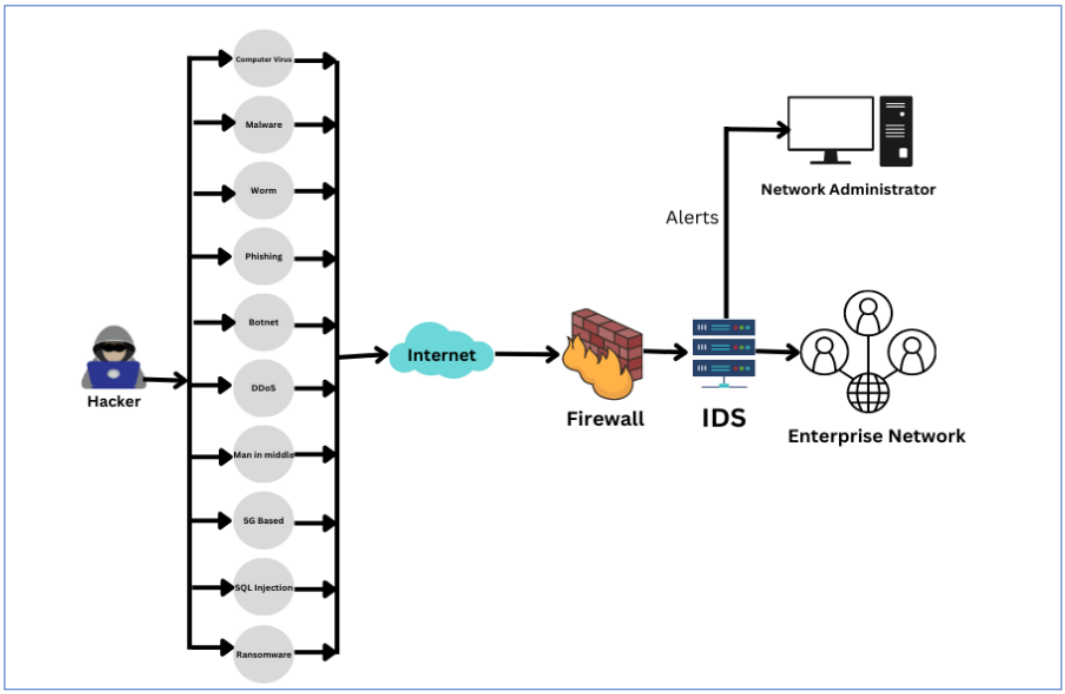


Figure 1.4: Different types of attacks and working of IDS

1.4.1 Malware :

Malware—short for "malicious software"—is software code written intentionally to harm a computer system or its users. Almost every modern cyberattack involves some type of malware. Threat actors use malware attacks to gain unauthorized access and render infected systems inoperable, destroying data,

stealing sensitive information, and even wiping files critical to the operating system. Common types of malwares include:

- **Ransomware** locks a victim's data or device and threatens to keep it locked, or leak it publicly, unless the victim pays a ransom to the attacker. According to the IBM Security X-Force Threat Intelligence Index 2023, ransomware attacks represented 17 percent of all cyberattacks in 2022.
- A **Trojan horse** is malicious code that tricks people into downloading it by appearing to be a useful program or hiding within legitimate software. Examples include remote access Trojans (RATs), which create a secret backdoor on the victim's device, or dropper Trojans, which install additional malware once they gain a foothold on the target system or network.
- **Spyware** is a highly secretive malware that gathers sensitive information, like usernames, passwords, credit card numbers and other personal data, and transmits it back to the attacker without the victim knowing.
- **Worms** are self-replicating programs that automatically spread to apps and devices without human interaction:

1.4.2 Social engineering and phishing:

Frequently referred to as "human hacking," social engineering manipulates targets into taking actions that expose confidential information, threaten their own or their organization's financial well-being, or otherwise compromise personal or organizational security.

Phishing is the best-known and most pervasive form of social engineering. Phishing uses fraudulent emails, email attachments, text messages or phone calls to trick people into sharing personal data or login credentials, downloading malware, sending money to cybercriminals, or taking other actions that might expose them to cybercrimes. Common types of phishing include:

- **Spear phishing**—highly targeted phishing attacks that manipulate a specific individual, often using details from the victim's public social media profiles to make the scam more convincing.
- **Whale phishing**—spear phishing that targets corporate executives or wealthy individuals.
- **Business email compromise (BEC)**—scams in which cybercriminals pose as executives, vendors, or trusted business associates to trick victims into wiring money or sharing sensitive data.

1.4.3 MITM attacks:

Man-in-the-middle (MITM) types of cyber-attacks refer to breaches in cybersecurity that make it possible for an attacker to eavesdrop on the data sent back and forth between two people, networks,

or computers. It is called a “man in the middle” attack because the attacker positions themselves in the “middle” or between the two parties trying to communicate. In effect, the attacker is spying on the interaction between the two parties.

In a MITM attack, the two parties involved feel like they are communicating as they normally do. What they do not know is that the person actually sending the message illicitly modifies or accesses the message before it reaches its destination. Some ways to protect yourself and your organization from MITM attacks is by using strong encryption on access points or to use a virtual private network (VPN).

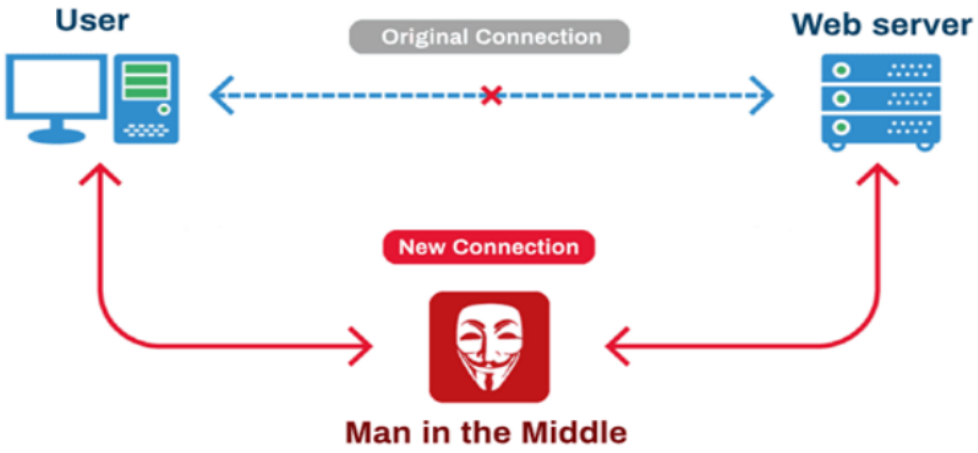


Figure 1.5: Man-in-the-middle attack

1.4.4 Ransomware:

With Ransomware, the victim’s system is held hostage until they agree to pay a ransom to the attacker. After the payment has been sent, the attacker then provides instructions regarding how the target can regain control of their computer. The name "ransomware” is appropriate because the malware demands a ransom from the victim.

In a ransomware attack, the target downloads ransomware, either from a website or from within an email attachment. The malware is written to exploit vulnerabilities that have not been addressed by either the system’s manufacturer or the IT team. The ransomware then encrypts the target’s workstation. At times, ransomware can be used to attack multiple parties by denying access to either several computers or a central server essential to business operations.

Affecting multiple computers is often accomplished by not initiating systems captivation until days or even weeks after the malware’s initial penetration. The malware can send AUTORUN files that go from one system to another via the internal network or Universal Serial Bus (USB) drives that connect

to multiple computers. Then, when the attacker initiates the encryption, it works on all the infected systems simultaneously.

In some cases, ransomware authors design the code to evade traditional antivirus software. It is therefore important for users to remain vigilant regarding which sites they visit and which links they click. You can also prevent many ransomware attacks by using a next-generation firewall (NGFW) that can perform deep data packet inspections using artificial intelligence (AI) that looks for the characteristics of ransomware.

Figure 1.6: Ranswormware attack

1.4.5 Password attacks:

Passwords are the access verification tool of choice for most people, so figuring out a target's password is an attractive proposition for a hacker. This can be done using a few different methods. Often, people keep copies of their passwords on pieces of paper or sticky notes around or on their desks. An attacker can either find the password themselves or pay someone on the inside to get it for them.

An attacker may also try to intercept network transmissions to grab passwords not encrypted by the network. They can also use social engineering, which convinces the target to input their password to solve a seemingly "important" problem. In other cases, the attacker can simply guess the user's password, particularly if they use a default password or one that is easy to remember such as "1234567."

Attackers also often use brute-force methods to guess passwords. A brute-force password hack uses basic information about the individual or their job title to try to guess their password. For example, their name, birthdate, anniversary, or other personal but easy-to-discover details can be used in different

combinations to decipher their password. Information that users put on social media can also be leveraged in a brute-force password hack. What the individual does for fun, specific hobbies, names of pets, or names of children are sometimes used to form passwords, making them relatively easy to guess for brute-force attackers.

A hacker can also use a dictionary attack to ascertain a user's password. A dictionary attack is a technique that uses common words and phrases, such as those listed in a dictionary, to try and guess the target's password.

One effective method of preventing brute-force and dictionary password attacks is to set up a lock-out policy. This locks out access to devices, websites, or applications automatically after a certain number of failed attempts. With a lock-out policy, the attacker only has a few tries before they get banned from access. If you have a lockout policy in place already and discover that your account has been locked out because of too many login attempts, it is wise to change your password.

If an attacker systematically uses a brute-force or dictionary attack to guess your password, they may take note of the passwords that did not work. For example, if your password is your last name followed by your year of birth and the hacker tries putting your birth year before your last name on the final attempt, they may get it right on the next try.

1.4.6 SQL injection attacks:

Structured Query Language (SQL) injection is a common method of taking advantage of websites that depend on databases to serve their users. Clients are computers that get information from servers, and an SQL attack uses an SQL query sent from the client to a database on the server. The command is inserted, or "injected", into a data plane in place of something else that normally goes there, such as a password or login. The server that holds the database then runs the command and the system is penetrated.

If an SQL injection succeeds, several things can happen, including the release of sensitive data or the modification or deletion of important data. Also, an attacker can execute administrator operations like a shutdown command, which can interrupt the function of the database.

To shield yourself from an SQL injection attack, take advantage of the least-privileged model. With least-privileged architecture, only those who absolutely need to access key databases are allowed in. Even if a user has power or influence within the organization, they may not be allowed to access specific areas of the network if their job does not depend on it.

For example, the CEO can be kept from accessing areas of the network even if they have the right to

know what is inside. Applying a least-privileged policy can prevent not just bad actors from accessing sensitive areas but also those who mean well but accidentally leave their login credentials vulnerable to attackers or leave their workstations running while away from their computers.

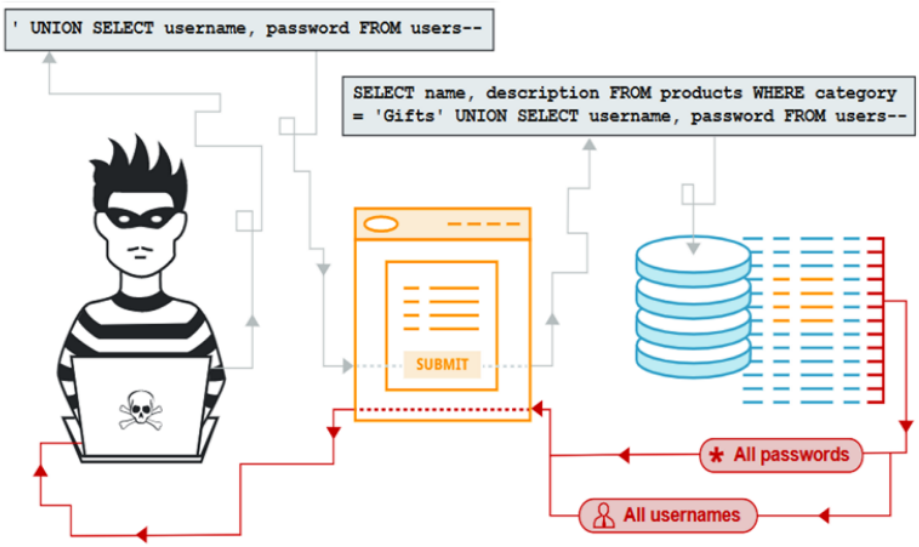


Figure 1.7: SQL attack technique

1.4.7 URL interpretation:

With URL interpretation, attackers alter and fabricate certain URL addresses and use them to gain access to the target's personal and professional data. This kind of attack is also referred to as URL poisoning. The name "URL interpretation" comes from the fact that the attacker knows the order in which a web-page's URL information needs to be entered. The attacker then "interprets" this syntax, using it to figure out how to get into areas they do not have access to.

To execute a URL interpretation attack, a hacker may guess URLs they can use to gain administrator privileges to a site or to access the site's back end to get into a user's account. Once they get to the page they want, they can manipulate the site itself or gain access to sensitive information about the people who use it.

For example, if a hacker attempts to get into the admin section of a site called GetYourKnowledgeOn.com, they may type in http://getyourknowledgeon.com/admin, and this will bring them to an admin login page. In some cases, the admin username and password may be the default "admin" and "admin" or very easy to guess. An attacker may also have already figured out the admin's password or narrowed it down to a few possibilities. The attacker then tries each one, gains access, and can manipulate, steal, or delete data at will.

To prevent URL interpretation attacks from succeeding, use secure authentication methods for any sensitive areas of your site. This may necessitate multi-factor authentication (MFA) or secure passwords consisting of seemingly random characters.

1.4.8 DNS spoofing:

With Domain Name System (DNS) spoofing, a hacker alters DNS records to send traffic to a fake or "spoofed" website. Once on the fraudulent site, the victim may enter sensitive information that can be used or sold by the hacker. The hacker may also construct a poor-quality site with derogatory or inflammatory content to make a competitor company look bad. In a DNS spoofing attack, the attacker takes advantage of the fact that the user thinks the site they are visiting is legitimate. This gives the attacker the ability to commit crimes in the name of an innocent company, at least from the perspective of the visitor.

To prevent DNS spoofing, make sure your DNS servers are kept up-to-date. Attackers aim to exploit vulnerabilities in DNS servers, and the most recent software versions often contain fixes that close known vulnerabilities.

1.4.9 Session hijacking:

Session hijacking is one of multiple types of MITM attacks. The attacker takes over a session between a client and the server. The computer being used in the attack substitutes its Internet Protocol (IP) address for that of the client computer, and the server continues the session without suspecting it is communicating with the attacker instead of the client. This kind of attack is effective because the server uses the client's IP address to verify its identity. If the attacker's IP address is inserted partway through the session, the server may not suspect a breach because it is already engaged in a trusted connection.

To prevent session hijacking, use a VPN to access business-critical servers. This way, all communication is encrypted, and an attacker cannot gain access to the secure tunnel created by the VPN.

1.4.10 Brute force attacks:

A brute-force attack gets its name from the "brutish" or simple methodology employed by the attack. The attacker simply tries to guess the login credentials of someone with access to the target system. Once they get it right, they are in.

While this may sound time-consuming and difficult, attackers often use bots to crack the credentials. The attacker provides the bot with a list of credentials that they think may give them access to the

secure area. The bot then tries each one while the attacker sits back and waits. Once the correct credentials have been entered, the criminal gains access.

To prevent brute-force attacks, have lock-out policies in place as part of your authorization security architecture. After a certain number of attempts, the user attempting to enter the credentials gets locked out. This typically involves "freezing" the account so even if someone else tries from a different device with a different IP address, they cannot bypass the lockout.

It is also wise to use random passwords without regular words, dates, or sequences of numbers in them. This is effective because, for example, even if an attacker uses software to try to guess a 10-digit password, it will take many years of non-stop attempts to get it right.

1.4.11 Web attacks:

Web attacks refer to threats that target vulnerabilities in web-based applications. Every time you enter information into a web application, you are initiating a command that generates a response. For example, if you are sending money to someone using an online banking application, the data you enter instructs the application to go into your account, take money out, and send it to someone else's account. Attackers work within the frameworks of these kinds of requests and use them to their advantage.

Some common web attacks include SQL injection and cross-site scripting (XSS), which will be discussed later in this article. Hackers also use cross-site request forgery (CSRF) attacks and parameter tampering. In a CSRF attack, the victim is fooled into performing an action that benefits the attacker. For example, they may click on something that launches a script designed to change the login credentials to access a web application. The hacker, armed with the new login credentials, can then log in as if they are the legitimate user.

Parameter tampering involves adjusting the parameters that programmers implement as security measures designed to protect specific operations. The operation's execution depends on what is entered in the parameter. The attacker simply changes the parameters, and this allows them to bypass the security measures that depended on those parameters.

To avoid web attacks, inspect your web applications to check for—and fix—vulnerabilities. One way to patch up vulnerabilities without impacting the performance of the web application is to use anti-CSRF tokens. A token is exchanged between the user's browser and the web application. Before a command is executed, the token's validity is checked. If it checks out, the command goes through—if not, it is blocked. You can also use Same Site flags, which only allow requests from the same site to be processed, rendering any site built by the attacker powerless.

1.4.12 Insider threats:

Sometimes, the most dangerous actors come from within an organization. People within a company's own doors pose a special danger because they typically have access to a variety of systems, and in some cases, admin privileges that enable them to make critical changes to the system or its security policies.

In addition, people within the organization often have an in-depth understanding of its cybersecurity architecture, as well as how the business reacts to threats. This knowledge can be used to gain access to restricted areas, make changes to security settings, or deduce the best possible time to conduct an attack.

One of the best ways to prevent insider threats in organizations is to limit employees' access to sensitive systems to only those who need them to perform their duties. Also, for the select few who need access, use MFA, which will require them to use at least one thing they know in conjunction with a physical item they have to gain access to a sensitive system. For example, the user may have to enter a password and insert a USB device. In other configurations, an access number is generated on a handheld device that the user has to log in to. The user can only access the secure area if both the password and the number are correct.

While MFA may not prevent all attacks on its own, it makes it easier to ascertain who is behind an attack—or an attempted one—particularly because only relatively few people are granted access to sensitive areas in the first place. As a result, this limited access strategy can work as a deterrent. Cybercriminals within your organization will know it is easy to pinpoint who the perpetrator is because of the relatively small pool of potential suspects.

1.4.13 Trojan horses:

A Trojan horse attack uses a malicious program that is hidden inside a seemingly legitimate one. When the user executes the presumably innocent program, the malware inside the Trojan can be used to open a backdoor into the system through which hackers can penetrate the computer or network. This threat gets its name from the story of the Greek soldiers who hid inside a horse to infiltrate the city of Troy and win the war. Once the "gift" was accepted and brought within the gates of Troy, the Greek soldiers jumped out and attacked. In a similar way, an unsuspecting user may welcome an innocent-looking application into their system only to usher in a hidden threat. To prevent Trojan attacks, users should be instructed not to download or install anything unless its source can be verified. Also, NGFWs can be used to examine data packets for potential threats of Trojans.

1.4.14 Drive-by attacks:

In a drive-by attack, a hacker embeds malicious code into an insecure website. When a user visits the site, the script is automatically executed on their computer, infecting it. The designation "drive by" comes from the fact that the victim only has to "drive by" the site by visiting it to get infected. There is no need to click on anything on the site or enter any information.

To protect against drive-by attacks, users should make sure they are running the most recent software on all their computers, including applications like Adobe Acrobat and Flash, which may be used while browsing the internet. Also, you can use web-filtering software, which can detect if a site is unsafe before a user visits it.

1.4.15 XSS attacks

With XSS, or cross-site scripting, the attacker transmits malicious scripts using clickable content that gets sent to the target's browser. When the victim clicks on the content, the script is executed. Because the user has already logged into a web application's session, what they enter is seen as legitimate by the web application. However, the script executed has been altered by the attacker, resulting in an unintended action being taken by the "user."

For example, an XSS attack may change the parameters of a transfer request sent through an online banking application. In the falsified request, the intended recipient of the transferred money has their name replaced with that of the attacker. The attacker may also change the amount being transferred, giving themselves even more money than the target initially intended to send.

One of the most straightforward ways of preventing XSS attacks is to use a whitelist of allowable entities. This way, anything other than approved entries will not be accepted by the web application. You can also use a technique called sanitizing, which examines the data being entered, checking to see if it contains anything that can be harmful.

1.4.16 Eavesdropping attacks:

Eavesdropping attacks involve the bad actor intercepting traffic as it is sent through the network. In this way, an attacker can collect usernames, passwords, and other confidential information like credit cards. Eavesdropping can be active or passive.

With active eavesdropping, the hacker inserts a piece of software within the network traffic path to collect information that the hacker analyzes for useful data. Passive eavesdropping attacks are different

in that the hacker "listens in," or eavesdrops, on the transmissions, looking for useful data they can steal.

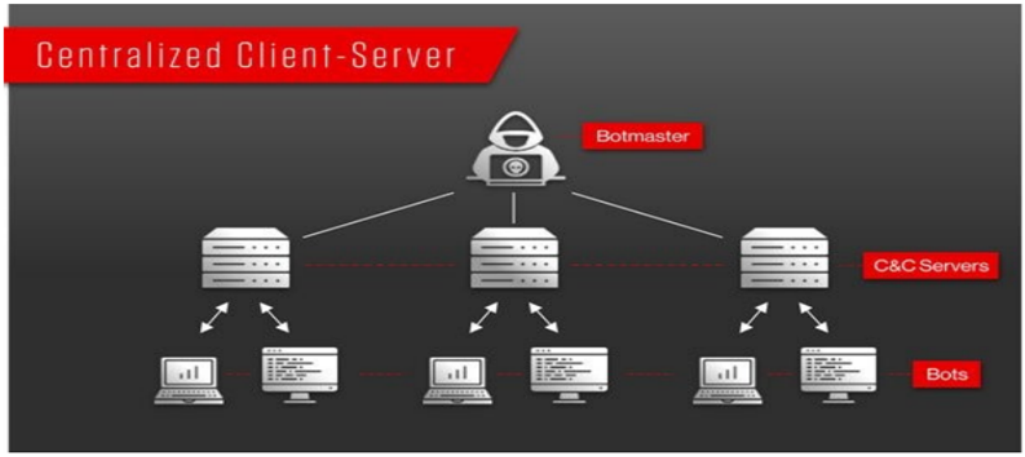
Both active and passive eavesdropping are types of MITM attacks. One of the best ways of preventing them is by encrypting your data, which prevents it from being used by a hacker, regardless of whether they use active or passive eavesdropping.

1.4.17 Birthday attack:

In a birthday attack, an attacker abuses a security feature: hash algorithms, which are used to verify the authenticity of messages. The hash algorithm is a digital signature, and the receiver of the message checks it before accepting the message as authentic. If a hacker can create a hash that is identical to what the sender has appended to their message, the hacker can simply replace the sender's message with their own. The receiving device will accept it because it has the right hash. The name "birthday attack" refers to the birthday paradox, which is based on the fact that in a room of 23 people, there is more than a 50% chance that two of them have the same birthday. Hence, while people think their birthdays, like hashes, are unique, they are not as unique as many thinks. To prevent birthday attacks, use longer hashes for verification. With each extra digit added to the hash, the odds of creating a matching one decrease significantly.

1.4.18 Botnet:

A botnet [short for bot network] is a hacker-controlled network of hijacked computers and gadgets infected with bot software. The bot network may be rented out to other hackers and used to disseminate spam and perform Distributed Denial of Service [DDoS] assaults. Botnets can also exist without a command and control (C&C) server by transferring commands from one bot to another via peer-to-peer [P2P] architecture and other management channels. Botnets made up of linked devices have become more prevalent as the internet of things (IoT) develops and becomes more widely used IRC clients were initially used by botnet operators to deliver instructions and carry out DDoS operations. Botnets (as shown in the Figure bellow) have been spotted mining bitcoins, intercepting data in transit, sending logs containing sensitive user information to the botnet master, and using the user's PC resources in recent months.



1.4.19 5G based attacks:

Attacks based on 5G technology are a more advanced type of network security threats. While 5G networks allow for faster data transfers, they also increase the risk of cyberattacks. Hackers are leveraging 5G devices to launch swarm-based network security assaults against numerous systems, mobile devices, and IoT (Internet of Things) networks. In addition, the attacker has the ability to make changes in real time .

1.4.20 DoS (Denial of Service) and DDoS Attacks:

At some point or another, we have all probably encountered website crashes. A rush in website traffic, whether due to a product introduction, a new promotional plan, or a sale, might cause the server to crash. Cyberattacks in the form of DoS and DDoS attacks, on the other hand, can cause website breakdowns. As a result of the malicious traffic overload, the system fails, and users are unable to access the website. The goal of these types of network security assaults is to bring the victim network's IT infrastructure to a halt [63]. DoS attacks differ from DDoS attacks in that hackers initiate DoS attacks over a single host network. DDoS attacks are more complex, and attackers can hack targeted systems with several computers. DDoS attacks are difficult to detect since they are launched from multiple hacked systems.

1.5 Top 5 open-source intrusion detection systems:

While most large enterprises have enterprise-grade technology in place, intrusion detection systems are also crucial for small and medium businesses to protect users, internal servers and public cloud environments. If you don't have a lot of budget at your disposal, open-source intrusion detection tools

are worth looking at.

Here are the five best open-source intrusion detection systems on the market currently:

- Snort
- Zeek
- OSSEC
- Suricata
- Security Onion

Snort

Snort is the oldest IDS and almost a de-facto standard IDS in the open-source world. Even though it doesn't have a real GUI, it offers a high level of customization, which makes it the IDS of choice for organizations. It can be used to detect a variety of attacks like buffer overflows, stealth port scans, CGI attacks, OS fingerprinting attempts, and more. The Snort community thrives on the backbone of passionate source developers that provide support for the software. Snort is available for Linux, Windows, Fedora, Centos, and FreeBSD. While the interface isn't very user-friendly, there are several applications available in the market such as Snorby, BASE, Squil, and Anaval that can perform in-depth analysis on the data collected by Snort.

Zeek

Formerly known as Bro, Zeek is a powerful network monitoring tool that focuses on general traffic analysis. It uses a domain-specific language that does not rely on traditional signatures. This means that you can design tasks for its policy engine. For example, you can configure the tool to automatically download suspicious files, send them for analysis, notify relevant authorities if anything is uncovered, blacklist the source and shut down the device that downloaded it. Zeek runs on Unix, Linux, Free BSD, and Mac OS X and can detect suspicious signatures and anomalies. Zeek's user community is supported by some well-known universities, supercomputing centers, research labs, and also lots of open-science communities.

OSSEC

OSSEC is an open-source host-based IDS system that performs log analysis, file integrity monitoring, Windows registry monitoring, centralized policy enforcement, rootkit detection, real-time alerting and active response. OSSEC runs on all major operating systems, including Linux, OpenBSD, FreeBSD, MacOS, Solaris and Windows. It has a client/server architecture that sends alerts and logs to a

centralized server for analysis even if the host system is fully compromised. The OSSEC installer is extremely light (under 1MB) and the majority of analysis occurs on the server making OSSEC very light on CPU usage. Another advantage of the architecture is ease of use as the administrator can centrally manage all agents from a single server.

Suricata

Suricata is a robust network threat detection engine that is capable of real time intrusion detection, inline intrusion prevention (IPS), network security monitoring (NSM) and offline pcap processing. Even though the architecture of Suricata is different from Snort, it behaves like Snort and can use the same signatures. While Snort is single thread, meaning it can only use one CPU at a time, Suricata is multi-threaded, able to take advantage of all available CPUs. It also has built-in hardware acceleration technology that can leverage the power of graphic cards to inspect network traffic. Suricata has the ability to invoke Lua scripts which can be used to peer into traffic or decode malware. Suricata is available on Linux, FreeBSD, OpenBSD, macOS / Mac OS X, and Windows and has very loyal community support.

Security Onion

Security Onion is an open-source tool designed for threat hunting, intrusion detection, enterprise security monitoring and log management. The interesting part of this tool is that it combines the power of other security tools like Snort, Kibana, Zeek, Wazuh, CyberChef, NetworkMiner, Suricata, and Logstash. This feature makes it highly comprehensive and versatile, covering pretty much every angle of IT security. Setting-up and working with multiple tools can be complicated but Security Onion features an intuitive set-up wizard that simplifies the set-up process. One of the drawbacks of this tool is that some of the tools have overlapping capabilities and navigating between tools can be tricky.

1.6 Commercial intrusion detection systems:

BluVector

Formerly known as Cortex and now owned by Comcast, BluVector's advanced threat detection solution uses artificial intelligence (AI) to complement an existing security stack. The AI detects fileless malware and zero-day threats and is designed to become more powerful the longer it sits in the environment.

Pros:

• On premise • Collects logs • Builds off of trusted Suricata and Zeek technology • Integrates with other tools • Open platform – data is easily available • Takes in data from multiple intel feeds and

sandboxes • Proprietary machine learning algorithm adds to capabilities • Broad MITRE ATT&CK coverage, does not use signature technology • Built-in tuning assistant to reduce false positives easily

Cons:

• Requires local resources, not built to support the cloud • No published license costs makes it difficult to compare with other solutions

Check Point Quantum IPS

Check Point embeds their Quantum IPS into their next generation firewall (NGFW) solutions to scan packets passing through the device. This device can replace a variety of other devices (firewalls, VPNs, etc.) and provides both IDS and IPS functionality.

Pros:

• Up to 15 Gbps integrated IPS performance • Detailed and customizable reports • Vulnerability detection for HTTP, POP, IMAP, SMTP, and more • Policies can be configured by vendor, product, protocol, file type, and threat year • Updates every two hours via a security gateway • Built-in antivirus, anti-bot and sandboxing • Blocks DNS tunneling, signature-less attacks, known CVEs. • Uses both signature and anomaly detection

Cons:

• Sold as hardware (secure gateway) only • No support for off-site (cloud, remote) resources that are not rerouted through the gateway • Internal network traffic must be routed through the gateway for protection

Fidelis Network

Fidelis Cybersecurity's Network IPS product analyzes network traffic to calculate the risk of all assets and communication in the network. The tool integrates with other Fidelis tools that protect other assets such as endpoints, cloud applications, and containers.

Pros:

• Uses the MITRE ATT&CK knowledge base to identify and respond to threats • Can decrypt and analyze encrypted network traffic • Supports cloud and local network • Tracks shadow IT deployments • Integrates with other security solutions • Part of an extended detection and response (XDR) solution • Offers sandboxing capabilities • Identifies account takeover, insider threat and hacker activity • Built-in OCR scanner to scan image and PDF attachments for emails • 24/7 global phone and web support • 15-day free trial

Cons:

• Complex configuration requirements • More expensive solution

Hillstone Networks

Hillstone Networks offers high-speed dedicated appliances for network IPS and next generation firewalls. Hillstone IPS hardware has been installed in over 20,000 customers since 2006 and offers a range of appliances to meet a flexible range of needs.

Pros:

• 13,000 signatures built-in, custom signatures, and anomaly detection • Sandboxing capabilities for investigation • Detection capabilities from layer 3 to layer 7 • Application aware • Options for anti-spam and URL-blocking • Cloud-based management of distributed devices

Cons:

• Appliance-only offerings • Appliances will need to be upgraded to accommodate growth • More expensive solution

Chapter 2

Machine learning

Introduction

Since their evolution, humans have been using many types of tools to accomplish various tasks in a simpler way. The creativity of the human brain led to the invention of different machines. These machines made human life easy by enabling people to meet various life needs, including traveling, industries, and computing. And Machine learning is the one among them. According to Arthur Samuel Machine learning is defined as the field of study that gives computers the ability to learn without being explicitly programmed. Arthur Samuel was famous for his checkers playing program. Machine learning (ML) is used to teach machines how to handle the data more efficiently. Sometimes after viewing the data, we cannot interpret the extract information from the data. In that case, we apply machine learning. With the abundance of datasets available, the demand for machine learning is in rise. Many industries apply machine learning to extract relevant data. The purpose of machine learning is to learn from the data. Many studies have been done on how to make machines learn by themselves without being explicitly programmed. Many mathematicians and programmers apply several approaches to find the solution of this problem which are having huge data sets.

2.1 Machine learning overview

2.1.1 History of Machine Learning

.

Machine learning is an area related to both cybernetics and computer science (or Control Science and Computer Science), attracting recently an overwhelming interest both of professionals and of the general public. In the last few years thanks to successes of computer science (the emergence of GPUs, leading to significant improvements in the performance of computers and development of special software, allowing to work with big data) machine learning is often attributed to computer science. However, historically, learning algorithms that provide convergence and sufficient convergence rate of the learning process arose within cybernetics/control. Below a look of a control theorist at the history of machine learning is presented. The author is a mathematician by education and has some experience in pattern recognition and control based on adaptation and learning in the 1960s.

With the evolution in machine learning in recent years, it has been started to be widely used in many areas. Cyber security has become one of the popular areas that has been benefiting machine learning solutions for detecting malicious activities. Since signature-based approaches are now evaluated as insufficient to detect current cyber threats, machine learning approaches have gained significant importance to fill this gap.

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts has published a paper "(Mcculloch and Pitts, 1990)" where they explained the working principle of human nervous system. They have proposed a model imitating the communication between neurons using an electrical circuit which is now called the born of neural networks. In 1950, "The Imitation Game (also called as Turing Test)" was invented by Alan Turing. The aim of this test was to determine if the machine is capable of behaving/thinking as an intelligent minded like humans. In 1952, Arthur Samuel has written a checkers-playing program and completed this first learning program in 1955. Since it is the first ever learning program as it ran, it is accepted as a milestone in artificial intelligence. In 1957, the psychologist Frank Rosenblatt has designed the first artificial neural network based on ideas about the work of the human nervous system. He has created a neural network called "Rosenblatt Perceptron" which consists of a single artificial neuron. This was a binary single neuron model used for pattern and shape recognition. However, its single neuron approach was not sufficient for solving non-linear separable problems. This artificial model is acknowledged as an ancestor of today's complex neural networks. Following this progress, in 1959, ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element) model was

created by Bernard Widrow and Marcian Hoff at Stanford University. This model also consisted of single-layer artificial neural network and differs in terms of weight adjustment from the previous model.

In 1967, the nearest neighbor algorithm was developed by Marcello Pelillo to map routes in order to find the most efficient route as a solution for travelling salesperson. This then constituted the basis of basic pattern recognition. The next important milestone of neural networks can be recorded as in 1986. The term back propagation which is used to train deep neural networks rediscovered to be used in neural networks and was first announced by researchers Rumelhart, Hinton and Williams from Stanford psychology department, in addition to the previous work of Widrow and Hoff. In 1990s, studies conducted in machine learning has changed their approach from knowledge-driven to data-driven. Support-vector machines (SVM) and Recurrent Neural Networks (RNN) have gained significance during this era.

Following these developments, in 1997, IBM computer Deep Blue beat the world chess champion Garry Kasparov. Then, in 1998, as a result of research at AT and T Bell Laboratories on digit recognition by using back-propagation, an important success was achieved in terms of accuracy in detecting handwritten postcodes from the US Postal Service.

In 2000s, unsupervised machine learning methods gained importance with Support Vector Clustering and kernel methods. In 2006, the term deep learning was first introduced by Geoffrey Hinton. Following his previous article on backpropagation in 1986, Hinton has published (Hinton et al., 2006) where a new approach is introduced to train multiple layered networks of restricted Boltzmann machines (RBM).

2.1.2 Machine Learning algorithms

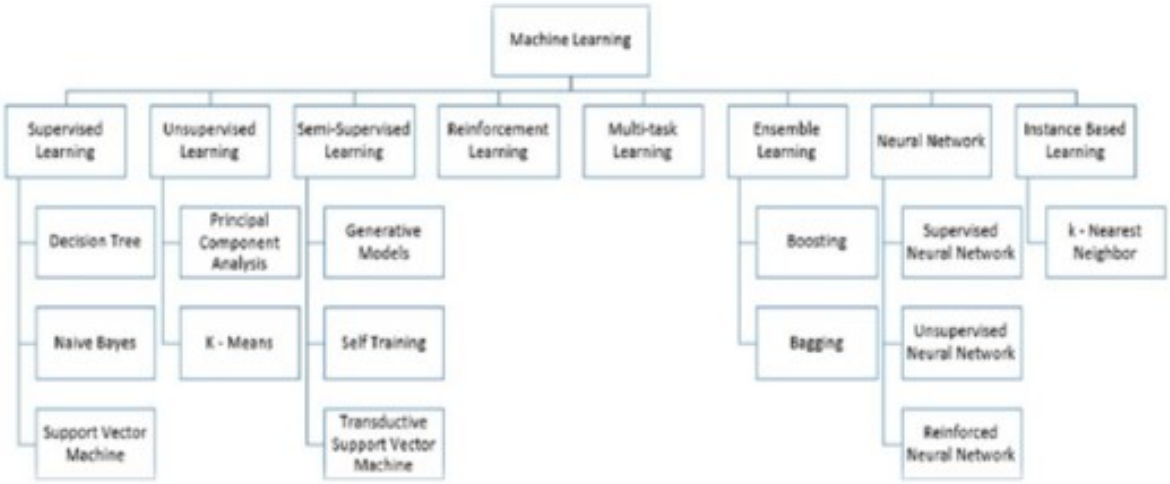


Figure 2.1: ML Algorithms

The integration of machine learning algorithms holds the potential to enhance network security through automated calculations and decision-making, such as identifying packet types within network traffic. These algorithms can be categorized as follows:

2.1.2.1 Supervised algorithms

Supervised algorithms demand a fully labeled dataset for effective operation. When applying supervised machine learning algorithms to network intrusion detection, the dataset is typically divided into two components: training data and testing data. The primary objective is to construct a model through training the algorithms with labeled data. This trained model is then utilized to predict the behavior of 'unknown' elements in the test data. Supervised learning is adept at detecting known attacks but falls short in identifying new or zero-day attacks

2.1.2.2 Unsupervised algorithms

Unsupervised learning relies on clustering techniques to develop models from unlabeled data. This approach distinguishes between malicious inputs and benign host logs or network traffic. Unsupervised methods analyze data attributes based on their statistical properties, without prior knowledge or labeling. The unsupervised algorithms discover hidden patterns or data groupings without the need for human intervention.

2.1.2.3 Semi-supervised

Learning Semi-supervised learning is an approach that merges elements of supervised and unsupervised learning techniques. Typically, only a small subset of the data is annotated with labels, while the majority remains unlabeled. This hybrid approach harnesses the robust performance of supervised learning while leveraging the cost-effectiveness of unsupervised learning.

2.1.2.4 Reinforcement Learning

This approach fundamentally differs from the other three in its core principle. Here, the algorithm encounters penalties for incorrect decisions made during training and receives rewards for making correct ones. Consequently, the algorithm formulates its own decision-making rule. In this project, we opt for supervised learning methods to capitalize on the availability of a meticulously labeled dataset. Our goal is to achieve high-performance benefits without incurring excessive costs. The machine learning methods utilized during the application phase encompass Naive Bayes, Random Forest, ID3, AdaBoost, and K

Nearest Neighbors. The selection of these methods is driven by the intention to combine well-established algorithms with diverse characteristics. In this context, we delve into a comprehensive examination of the algorithms employed, as outlined below [11].

2.1.2.5 Deep Learning

The security of computer networks has been in the focus of research for years. The organization has come to realize that information and network security technology has become very important in protecting its information. Any successful attempt or unsuccessful attempt to compromise the integrity, confidentiality, and availability of any information resource or the information itself is considered a security attack or an intrusion. Every day new kinds of attacks are being faced by industries. One of the solutions to this problem is by using an Intrusion Detection System (IDS). Machine Learning is one of the techniques used in the IDS to detect attacks. Machine learning is concerned with the design and development of algorithms and methods that allow computer systems to autonomously acquire and integrate knowledge to continuously improve them to finish their tasks efficiently and effectively. In recent years, Machine Learning Intrusion Detection system has been giving high accuracy and good detection of novel attacks. Intrusion detection system (IDS) is a security technique attempting to detect various attacks. They are the set of techniques that are used to detect suspicious activity both on host and network level.

Several classifications of intrusion detection methods have been proposed in the earlier period. Currently the two basic methods of detection (analytical method) are signature-based and anomaly-based. The signature-based method, also known as misuse detection, seems for a specific signature to match, signaling an intrusion. They can detect many or all known attack patterns, but the weakness of signature-based intrusion detection systems is the incapability of identifying new types of attacks or variations of known attacks.

Another useful method for intrusion detection is called anomaly detection. Anomaly detection applied to intrusion detection and computer security has been an active area of research. In anomaly based IDSs, the normal behavior of the system or network traffic are represented and, for any behavior that varies over a predefined threshold, an anomalous activity is identified. On the other hand, in anomaly based IDSs, the number of false positives generated are higher than those based on signatures. An important issue in anomaly based IDSs is how these systems should be trained, i.e., how to define what is a normal behavior of a system or network environment (which features are relevant) and how to represent this behavior computationally.

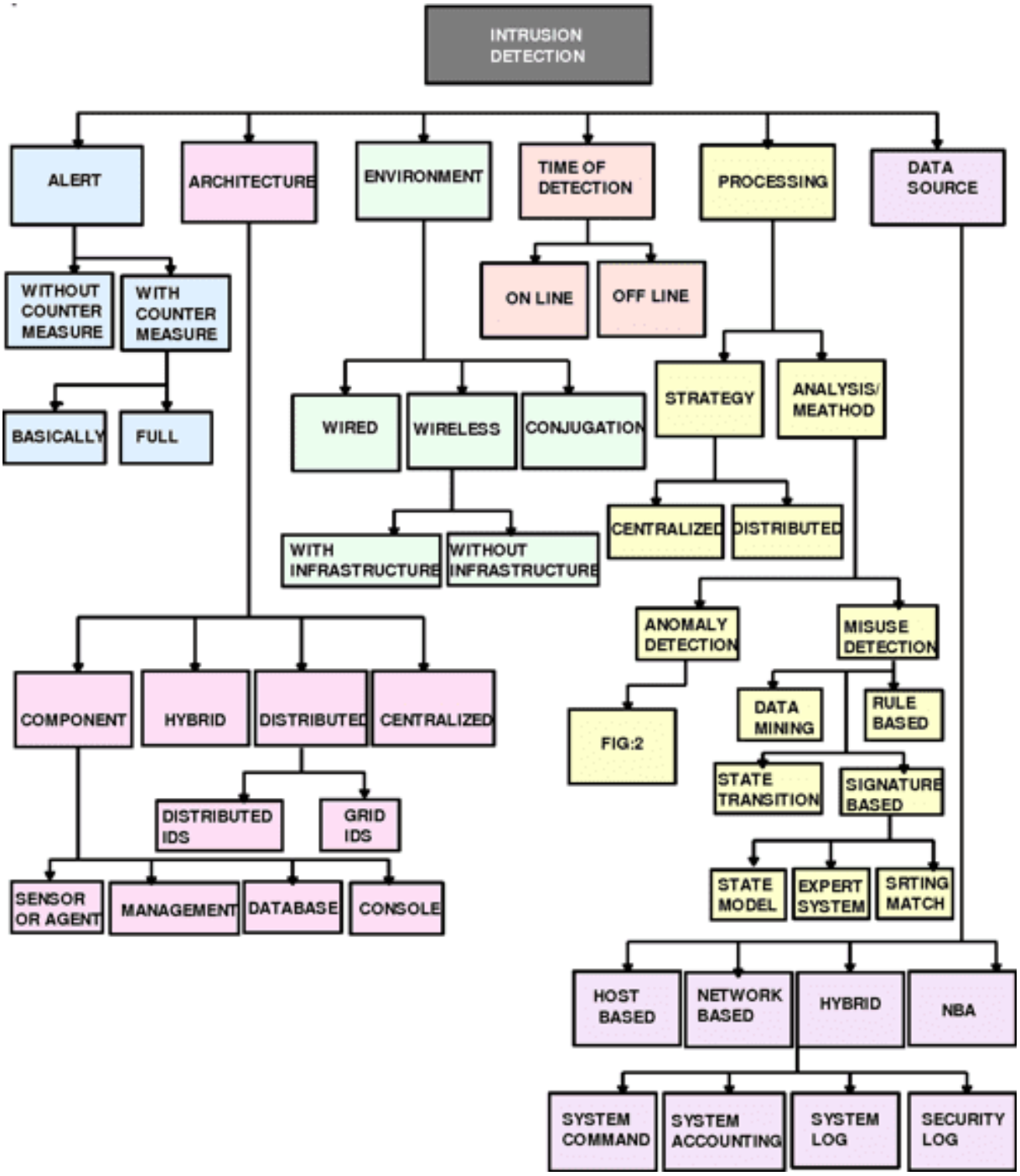


Figure 2.2: Taxonomy of IDS

According to the type of processing related to the "behavioral" model of the target system, anomaly detection techniques can be classified into three main categories; statistical based, knowledge-based, and machine learning based.

The idea of applying machine learning techniques for intrusion detection is to automatically build the model based on the training data set. This data set contains a collection of data instances each of which can be described using a set of attributes (features) and the associated labels. The attributes can be of

different types such as categorical or continuous. The nature of attributes determines the applicability of anomaly detection techniques. For example, distance-based methods are initially built to work with continuous features and usually do not provide satisfactory results on categorical attributes. The labels associated with data instances are usually in the form of binary values i.e. normal and anomalous.

Machine learning techniques are based on establishing an explicit or implicit model. A singular characteristic of these schemes is the need for labeled data to train the behavioral model, a procedure that places severe demands on resources. In many cases, the applicability of machine learning principles coincides with that for statistical techniques, although the former is focused on building a model that improves its performance on the basis of previous results. Hence, machine learning for IDS has the ability to change its execution strategy as it acquires new information. This feature could make it desirable to use such schemes for all situations.

2.2 Research on Machine Learning-Based IDS

In this section, we introduce various ways to apply machine learning to IDS design for different data types. The different types of data reflect different attack behaviors, which include host behaviors and network behaviors. Host behaviors are reflected by system logs, and network behaviors are reflected by network traffic. There are multiple attack types, each of which has a unique pattern. Thus, selecting appropriate data sources is required to detect different attacks according to the attack characteristics. For instance, one salient feature of a DOS attack is to send many packets within a very short period of time; therefore, flow data is suitable for detecting a DOS attack. A covert channel involves data-leaking activity between two specific IP addresses, which is more suited to detection from session data.

2.2.1 Packet-Based Attack Detection

Packets, which are the basic units of network communication, represent the details of each communication. Packets consist of binary data, meaning that they are incomprehensible unless they are first parsed. A packet consists of a header and application data. The headers are structured fields that specify IP addresses, ports and other fields specific to various protocols. The application data portion contains the payload from the application layer protocols. There are three advantages to using packets as IDS data sources: (1) Packets contain communication contents; thus, they can effectively be used to detect U2L and R2L attacks. (2) Packets contain IPs and timestamps; thus, they can locate the attack sources precisely. (3) Packets can be processed instantly without caching; thus, detection can occur in real time.

However, individual packets do not reflect the full communication state nor the contextual information of each packet, so it is difficult to detect some attacks, such as DDOS. The detection methods based on packets mainly include packet parsing methods and payload analysis methods.

2.2.1.1 Packet Parsing-Based Detection

Various types of protocols are used in network communications, such as HTTP and DNS. These protocols have different formats; the packet parsing-based detection methods primarily focus on the protocol header fields. The usual practice is to extract the header fields using parsing tools (such as Wireshark or the Bro) and then to treat the values of the most important fields as feature vectors. Packet parsing-based detection methods apply to shallow models.

The header fields provide basic packet information from which feature can be extracted used with using classification algorithms to detect attacks. Mayhew et al. [40] proposed an SVM- and K-means-based packet detection method. They captured packets from a real enterprise network and parsed them with Bro. First, they grouped the packets according to protocol type. Then, they clustered the data with the K-means++ algorithm for the different protocol datasets. Thus, the original dataset was grouped into many clusters, where the data from any given cluster were homologous. Next, they extracted features from the packets and trained SVM models on each cluster. Their precision scores for HTTP, TCP, Wiki, Twitter, and E-mail protocols reached 99.6 %, 92.9 %, 99 %, 96 %, and 93 %, respectively.

In packet parsing-based detection, unsupervised learning is a common way to solve the high false alarm rate problem. Hu et al. [41] proposed a fuzzy C-means based packet detection method. The fuzzy C mean algorithm introduces fuzzy logic into the standard K-means algorithm such that samples belong to a cluster with a membership degree rather than as a Boolean value such as 0 or 1. They used Snort to process the DARPA 2000 dataset, extracting Snort alerts, source IPs, destination IPs, source ports, destination ports, and timestamps. Then, they used this information to form feature vectors and distinguished false alerts from true alerts by clustering the packets. To reduce the influence of initialization, they ran the clustering algorithms ten times. The results showed that the fuzzy C-means algorithm reduced the false alarm rate by 16.58 % and the missed alarm rate by 19.23 %.

2.2.1.2 Payload Analysis-Based Detection

Apart from packet parsing-based detection, payload analysis-based detection places emphasis on the application data. The payload analysis-based methods are suitable for multiple protocols because they do not need to parse the packet headers.

As a type of unstructured data, payloads can be processed directly by deep learning models [42]. It should be noted that this method does not include encrypted payloads. Shallow models depend on manual features and private information in packets, leading to high labor costs and privacy leakage problems. Deep learning methods learn features from raw data without manual intervention. Min et al. [43] utilized a text-based CNN to detect attacks from payloads. They conducted experiments on the ISCX 2012 dataset and detected attacks with both statistical and content features. The statistical features mainly came from packet headers and included protocols, IPs, and ports. The content features came from the payloads. First, payloads from different packets were concatenated. Next, the concatenated payloads were encoded by skip-gram word embedding. Then, the content features were extracted with a CNN. Finally, they trained a random forest model to detect attacks. The final model reached an accuracy of 99.13

Combining various payload analysis techniques can achieve comprehensive content information, which is able to improve the effect of the IDS. Zeng et al. [44] proposed a payload detection method with multiple deep learning models. They adopted three deep learning models (a CNN, an LSTM, and a stacked autoencoder) to extract features from different points of view. Among these, the CNN extracted local features, the RNN extracted time series features, and the stacked autoencoder extracted text features. The accuracy of this combined approach reached 99.22 % on the ISCX 2012 dataset.

Extracting payload features with unsupervised learning is also an effective detection method. Yu et al. [45] utilized a convolutional autoencoder to extract payload features and conducted experiments on the CTU-UNB dataset. This dataset includes the raw packets of 8 attack types. To take full advantage of convolutions, they first converted the packets into images. Then, they trained a convolutional autoencoder model to extract features. Finally, they classified packets using learned features. The precision, recall and F-measure on the test set reached 98.44 %, 98.40 %, and 98.41 % respectively.

To enhance the robustness of IDSs, adversarial learning becomes a novel approach. Adversarial learning can be used for attacks against IDS. Meanwhile, it is also a novel way to improve detection accuracy of IDS. Rigaki et al. [46] used a GAN to improve the malware detection effect. To evade detection, malware applications try to generate packets similar to normal packets. Taking the malware FLU as an example, the command & control (C & C) packets are very similar to packets generated by Facebook. They configured a virtual network system with hosts, servers, and an IPS. Then, they started up the malware FLU and trained a GAN model. The GAN guided the malware to produce packets similar to Facebook. As the training epochs increased, the packets blocked by the IPS decreased and packet that passed inspection increased. The result was that the malicious packets generated by the GAN were more similar to normal packets. Then, by analyzing the generated packets, the robustness

of the IPS was improved.

2.2.2 Flow-Based Attack Detection

Flow data contains packets grouped in a period, which is the most widespread data source for IDSs. The KDD99 and the NSL-KDD datasets are both flow data. Detecting attacks with flow has two benefits: (1) Flow represents the whole network environment, which can detect most attacks, especially DOS and Probe. (2) Without packet parsing or session restructuring, flow preprocessing is simple. However, flow ignores the content of packets; thus, its detection effect for U2R and R2L is unsatisfactory. When extracting flow features, packets must be cached packets; thus, it involves some hysteresis. Flow-based attack detection mainly includes feature engineering and deep learning methods. In addition, the strong heterogeneity of flow may cause poor detection effects. Traffic grouping is the usual solution to this problem.

2.2.2.1 Feature Engineering-Based Detection

Traditional machine learning models cannot directly address flow data; therefore, feature engineering is an essential step before these models can be applied. Feature engineering-based methods adopt a "feature vectors + shallow models" mode. The feature vectors are suitable for most machine learning algorithms. Each dimension of the feature vectors has clear interpretable semantics. The common features include the average packet length, the variance in packet length, the ratio of TCP to UDP, the proportion of TCP flags, and so on. The advantages of these types of detection methods are that they are simple to implement, highly efficient, and can meet real-time requirements.

The existing feature engineering-based IDSs often have high detection accuracy but suffer from a high false alarm rate. One solution is to combine many weak classifiers to obtain a strong classifier. Goeschel et al. [47] proposed a hybrid method that included SVM, decision tree, and Naïve Bayes algorithms. They first trained an SVM model to divide the data into normal or abnormal samples. For the abnormal samples, they utilized a decision tree model to determine specific attack types. However, a decision tree model can identify only known attacks, not unknown attacks. Thus, they also applied a Naïve Bayes classifier to discover unknown attacks. By taking advantage of three different classifier types this hybrid method achieved an accuracy of 99.62

Another research objective is to accelerate the detection speed. Kuttranont et al. [48] proposed a KNN-based detection method and accelerated calculation via parallel computing techniques running on a graphics processing unit (GPU). They modified the neighbor-selecting rule of the KNN algorithm.

The standard KNN selects the top K nearest samples as neighbors, while the improved algorithm selects a fixed percentage (such as 50 %) of the neighboring samples as neighbors. The proposed method considers the unevenness of data distribution and performs well on sparse data. These experiments were conducted using the KDD99 dataset, achieving an accuracy of 99.30 %. They also applied parallel computing and the GPU to accelerating calculation. The experimental results showed that the method with the GPU was approximately 30 times faster than that without the GPU.

The unsupervised learning methods are also applied to IDS, a typical way is to divide data with clustering algorithms. The standard K-means algorithm is inefficient on big datasets. To improve detection efficiency, Peng et al. [13] proposed an improved K-means detection method with mini batch. They first carried out data preprocessing on the KDD99 dataset. The nominal features were transformed into numerical types, and each dimension of the features was normalized by the max-min method. Then, they reduced the dimensions using the principal components analysis (PCA) algorithm. Finally, they clustered the samples with the K-means algorithm, but they improved K-means from two aspects. (1) They altered the method of initialization to avoid becoming stuck in a local optimum. (2) They introduced the mini-batch trick to decrease the running time. Compared with the standard K-means, the proposed method achieved higher accuracy and runtime efficiency.

2.2.2.2 Deep Learning-Based Detection

Feature engineering depends on domain knowledge, and the quality of features often becomes a bottleneck of detection effects. Deep learning-based detection methods learn feature automatically. These types of methods work in an end-to-end fashion and are gradually becoming the mainstream approach in IDS studies.

Deep learning methods can directly process raw data, allowing them to learn features and perform classification at the same time. Potluri et al. [49] proposed a CNN-based detection method. They conducted experiments on the NSL-KDD and the UNSW-NB 15 datasets. The data type in these datasets is a feature vector. Because CNNs are good at processing 2-dimensional (2D) data, they first converted the feature vectors into images. Nominal features were one-hot coded, and the feature dimensions increased from 41 to 464. Then, each 8-byte chunk was transformed into one pixel. Blank pixels were padded with 0. The end result was that the feature vectors were transformed into images of 8*8 pixels. Finally, they constructed a three-layer CNN to classify the attacks. They compared their model with other deep networks (ResNet 50 and GoogLeNet), and the proposed CNN performed best, reaching accuracies of 91.14 % on the NSL-KDD and 94.9 % on the UNSW-NB 15.

Unsupervised deep learning models can also be used to extract features; then, shallow models can be used to perform classification. Zhang et al. [50] extracted features with a sparse autoencoder and detected attacks with an XGBoost model. They used data from the NSL-KDD dataset. Due to the imbalanced nature of this dataset, they sampled the dataset using SMOTE. The SMOTE algorithm oversamples the minority classes and divides the majority classes into many sub-classes so that every class is balanced. The sparse auto encoder introduces a sparsity constraint into the original auto encoder, enhancing its ability to detect unknown samples. Finally, they classified the data using an XGBoost model. Their model achieved accuracies on the Normal, DOS, Probe, R2L, and U2R classes of 99.96 %, 99.17 %, 99.50 %, 97.13 %, and 89.00 %, respectively.

Deep learning models have made great strides in big data analysis; however, their performances are not ideal on small or unbalanced datasets. Adversarial learning approaches can improve the detection accuracy on small datasets. Zhang et al. [51] conducted data augmentation with a GAN. The KDD99 dataset is both unbalanced and lacks new data, which leads to poor generalizability of machine learning models. To address these problems, they utilized a GAN to expand the dataset. The GAN model generated data similar to the flow data of KDD99. Adding this generated data to the training set allows attack variants to be detected. They selected 8 types of attacks and compared the accuracies achieved on the original dataset compared to the expanded dataset. The experimental results showed that adversarial learning improved 7 accuracies in 8 attack types.

2.2.2.3 Payload Analysis-Based Detection

Flow includes all traffic within a period, and many types of traffics may act as white noise in attack detection. Training machine learning models with such data probably leads to overfitting. One natural approach is to group traffic to decrease heterogeneity. The grouping methods include protocol-based and data-based methods.

The traffic features of various protocols have significant differences; thus, grouping traffic by protocol is a valid step toward improving accuracy. Teng et al. [52] proposed an SVM detection method based on protocol grouping using the data of the KDD99 dataset, which involves various protocols. They first divided the dataset based on protocol type, and considered only TCP, UDP, and ICMP protocols. Then, according to the characteristics of these different protocols, they selected features for each sub dataset. Finally, they trained SVM models on the 3 sub datasets, obtaining an average accuracy of 89.02

Grouping based on data characteristics is another traffic grouping approach. One typical method is clustering. Ma et al. [53] proposed a DNN and spectral clustering-based detection method. The

heterogeneity of flow may cause low accuracy. Therefore, they first divided the original dataset into 6 subsets, in which each subset was highly homogeneous. Then, they trained DNN models on every subset. The accuracy of their approach on the KDD99 and the NSL-KDD datasets reached 92.1 %.

2.2.3 Session-Based Attack Detection

A session is the interaction process between two terminal applications and can represent high-level semantics. A session is usually divided on the basis of a 5-tuple (client IP, client port, server IP, server port, and protocol). There are two advantages of detection using sessions. (1) Sessions are suitable for detecting an attack between specific IP addresses, such as tunnel and Trojan attacks. (2) Sessions contain detailed communications between the attacker and the victim, which can help localize attack sources. However, session duration can vary dramatically. As a result, a session analysis sometimes needs to cache many packets, which may increase lag. The session-based detection methods primarily include statistics-based features and sequence-based features.

2.2.3.1 Statistic-Based Feature Detection Methods

Session statistical information includes the fields in packet headers, the number of packets, the proportion of packets coming from different directions, and so on. This statistical information is used to compose feature vectors suitable for shallow models. The sessions have high layer semantics; thus, they are easily described by rules. Decision tree or rule-based models may be appropriate methods. Unfortunately, the methods based on statistical features ignore the sequence information, and they have difficulties detecting intrusions related to communication content.

Because statistical information includes the basic features of sessions, supervised learning methods can utilize such information to differentiate between normal sessions and abnormal sessions. The existing session-based detection methods often face problems of low accuracy and have high runtime costs. Ahmim et al. [54] proposed a hierarchical decision tree method in which, reduce the detection time, they analyzed the frequency of different types of attacks and designed the detection system to recognize specific attacks. They used data from the CICIDS 2017 dataset that included 79-dimensional features and 15 classes. The proposed detection system had a two-layer structure. The first layer consisted of two independent classifiers (i.e., a decision tree and a rule-based model), which processed part of the features. The second layer was a random forest classifier, which processed all the features from the dataset as well as the output of the first layer. They compared multiple machine learning models on 15 classes; their proposed methods performed best on 8 of the 15 classes. Moreover, the proposed method

had low time consumption, reflecting its practicability.

Session-based detection using supervised learning models depends on expert knowledge, which is difficult to expand to new scenarios. To address this problem, Alseiari et al. [55] proposed an unsupervised method to detect attacks in smart grids. Due to the lack of smart grid datasets, they constructed a dataset through simulation experiments. First, they captured and cached packets to construct sessions. Then, they extracted 23-dimensional features from the sessions. Next, they utilized mini batch K-means to divide the data into many clusters. Finally, they labeled the clusters. This work was based on two hypotheses. The first was that normal samples were the majority. The second one was that the distances among the normal clusters were relatively short. When the size of a cluster was less than 25 % of the full sample amount or a cluster centroid was far away from all other the other cluster centroids, that cluster was judged as abnormal. No expert knowledge was required for any part of this process. The proposed methods were able to detect intrusion behaviors in smart grids effectively and locate the attack sources while holding the false alarm rate less to than 5 %.

2.2.3.2 Sequence Feature-Based Detection

Different from flow, the packets in sessions have a strict order relationship. The sequence features mainly contain the packet length sequence and the time interval sequence. Analyzing the sequence can obtain detailed session interaction information. Most machine learning algorithms cannot deal with sequences, and related methods are relatively rare. At present, most sequence feature-based detection adopts the RNN algorithm.

Encoding raw data is a common preprocessing step for RNN methods. The bag of words (BoW) model is a frequently used text processing technology. Yuan et al. [56] proposed a DDOS detection method based on the LSTM using UNB ISCX 2012 dataset. They first extracted 20-dimensional features from the packets and encoded them with BoW. Then, they concatenated the packets in sequence, resulting in matrices with a size of m*n, where m was the number of packets in a session and n was the dimension of a packet, and both m and n were variable. Finally, they trained a CNN to extract local features and an LSTM to classify the sessions. They provided comprehensive experimental results, reaching accuracy, precision, recall, and F-measure scores of 97.606 %, 97.832 %, 97.378 %, and 97.601 %, respectively.

One of the drawbacks of the BoW is that it is unable to represent the similarity between words. Word embedding approaches overcome that problem. Radford et al. [57] proposed a session detection method based on a bi-LSTM. Because LSTMs had made great strides in NLP, they expressed the

sessions as a specific language. They conducted experiments on the ISCX IDS dataset. First, they grouped packets on the basis of IP addresses to obtain sessions. Then, they encoded the sessions with the word embedding. Finally, they trained an LSTM model to predict abnormal sessions. To utilize the contextual information, they adopted a bi-LSTM model to learn the sequence features in two directions.

In addition to text processing technology, the character-level CNN is a novel encoding method. Wang et al. [58] proposed a hierarchical deep learning detection method in which a session contains not only packet contents but also the packet time sequence. Then, they designed a hierarchical deep learning method using a CNN to learn the low-level spatial features and an LSTM to learn the high-level time features, where the time features are based on the spatial features. They conducted experiments on the DARPA 1998 and the ISCX 2012 datasets. They first applied the CNN to extract spatial features from packets. Next, they concatenated the spatial features in sequence and extracted time features using the LSTM model. The resulting model achieved accuracies between 99.92 % and 99.96 %, and detection rates between 95.76 % and 98.99 %.

2.2.4 Log-Based Attack Detection

Logs are the activity records of operating systems or application programs; they include system calls, alert logs, and access records. Logs have definite semantics. There are three benefits to using logs as a data source in IDSs. (1) Logs include detailed content information suitable for detecting SQL injection, U2R, and R2L attacks. (2) Logs often carry information about users and timestamps that can be used to trace attackers and reveal attack times. (3) Logs record the complete intrusion process; thus, the result is interpretable. However, one problem is that log analysis depends on cyber security knowledge. Additionally, the log formats of different application programs do not have identical formats, resulting in low scalability. The log-based attack detection primarily includes hybrid methods involving rules and machine learning, log feature extraction-based methods, and text analysis-based methods.

2.2.4.1 Rule and Machine Learning-Based Hybrid Methods

Hybrid methods combine rule-based detection and machine learning, which together achieve better performances than do single detection systems. Many rule-based detection systems (e.g., Snort) generate masses of alerts; however, most of the alerts involve only operations that do not match the rules; therefore, these are often not real intrusion behaviors. The hybrid methods take the log output of the rule-based systems as inputs; then, machine learning models are used to filter out the meaningless alerts.

Many IDSs suffer from high false alarm rates, which cause real attacks to be embedded among many

meaningless alerts. Ranking alerts via machine learning models forms a possible solution. To reduce the false alarm rate, Meng et al. [59] proposed a KNN method to filter alarms. They conducted experiments in a real network environment and generated alerts using Snort. Then, they trained a KNN model to rank the alerts. There were 5 threat levels in total in their experiment, and the results showed that the KNN model reduced the number of alerts by 89

Some IDSs perform a function similar to human interaction, in which alerts are ranked by machine learning to reduce analyst workloads. McElwee et al. [60] proposed an alert filtering method based on a DNN. They first collected the log generated by McAfee. Then, they trained a DNN model to find important security events in the logs. Next, the extracted important events were analyzed by security experts. Then, the analysis results were used as training data to enhance the DNN model, forming an interaction and promotion cycle. The proposed hybrid system can reduce analyst workloads and accelerate security analyses.

2.2.4.2 Log Feature Extraction-Based Detection

This method involves extracting log features according to domain knowledge and discovering abnormal behaviors using the extracted features, which is suitable for most machine learning algorithms. Using a sliding window to extract features is a common approach. The sliding window makes use of the contextual information contained in logs. In addition, the sliding window is a streaming method that has the benefit of low delay.

Intrusion behaviors may leave traces of system calls, and analyzing these system calls with classification algorithms can detect intrusions. Tran et al. [61] proposed a CNN method to analyze system calls. Every underlying operation that involves the operating system will use system calls; thus, analyzing the system call path can reproduce the complete intrusion process. They conducted experiments on the NGIDS-DS and the ADFA-LD datasets, which include a series of system calls. First, they extracted features with a sliding window. Then, they applied a CNN model to perform classification. The CNN was good at finding local relationships and detecting abnormal behaviors from system calls.

Model interpretation is another important research direction, which has attracted extensive attention. Tuor et al. [62] proposed an interpretable deep learning detection method using data from the CERT Insider Threat dataset, which consists of system logs. They first extracted 414-dimensional features using a sliding window. Then, they adopted a DNN and an RNN to classify logs. The DNN detected attacks based on the log contents, and the RNN detected attacks based on the log sequences. The proposed methods reduced the analysis workload by 93.5 % and reached a detection rate of 90 %.

Furthermore, they decomposed the abnormal scores into the contributions of each behavior, which was a helpful analysis. Interpretable models are more convincing than are uninterpretable models.

Some logs lack labeled information; consequently, supervised learning is inappropriate. Unsupervised learning methods are usually used with unlabeled logs. Bohara et al. [63] proposed an unsupervised learning detection method in the enterprise environment. They conducted experiments on the VAST 2011 Mini Challenge 2 dataset and extracted features from the host and network logs. Due to the different influences of each feature, they selected features using the Pearson correlation coefficient. Then, they clustered the logs with the K-means and DBSCAN algorithms. By measuring the salient cluster features, the clusters were associated with abnormal behaviors. Finally, they analyzed the abnormal clusters manually to determine the specific attack types.

2.2.4.3 Text Analysis-Based Detection

The text analysis-based detection regards logs as plain text. The methods utilize mature text processing techniques such as the n-gram to analyze logs. Compared with log feature extraction-based methods, this method understands log content at the semantic level and therefore has stronger interpretability.

In log-based detection, extracting text features from logs and then performing classification is the usual approach. When analyzing texts, a small number of keywords have large impacts on the whole text. Thus, the keywords in the field of cyber security aid in improving the detection effect. Uwagbole et al. [64] proposed an SQL-injection detection method for the Internet of Things (IoT). They collected and labeled logs from a real environment. The logs provide the contextual information of the SQL injection attack. First, they extracted 479,000 high-frequency words from the logs and then added 862 keywords that appear in SQL queries to compose a dictionary. Then, they removed duplicate and missing records from the log and balanced the data with SMOTE. Next, they extracted features using the n-gram algorithm and selected features using Chi-square tests. Finally, they trained an SVM model to perform classification, achieving accuracy, precision, recall, and F-measure scores of 98.6 %, 97.4 %, 99.7 % and 98.5 %, respectively.

In an actual network environment, normal samples are in the majority, and abnormal samples are rare. One-class classification, a type of unsupervised learning method, uses only normal samples for training, which solves the problem of a lack of abnormal samples. Vartouni et al. [65] proposed a web attack detection method based on the isolate forest model. They used the data of the CSIC 2010 dataset. First, they extracted 2572-dimensional features from HTTP logs with the n-gram. Then, they utilized an autoencoder to remove irrelevant features. Finally, they trained an isolation forest model to

discover abnormal webs, which reached an accuracy of 88.32 %.

2.3 Machine Learning Techniques for Intrusion Detection

Several machine learning-based schemes have been applied to IDS. Some of the most important techniques are explained in the following subsections.

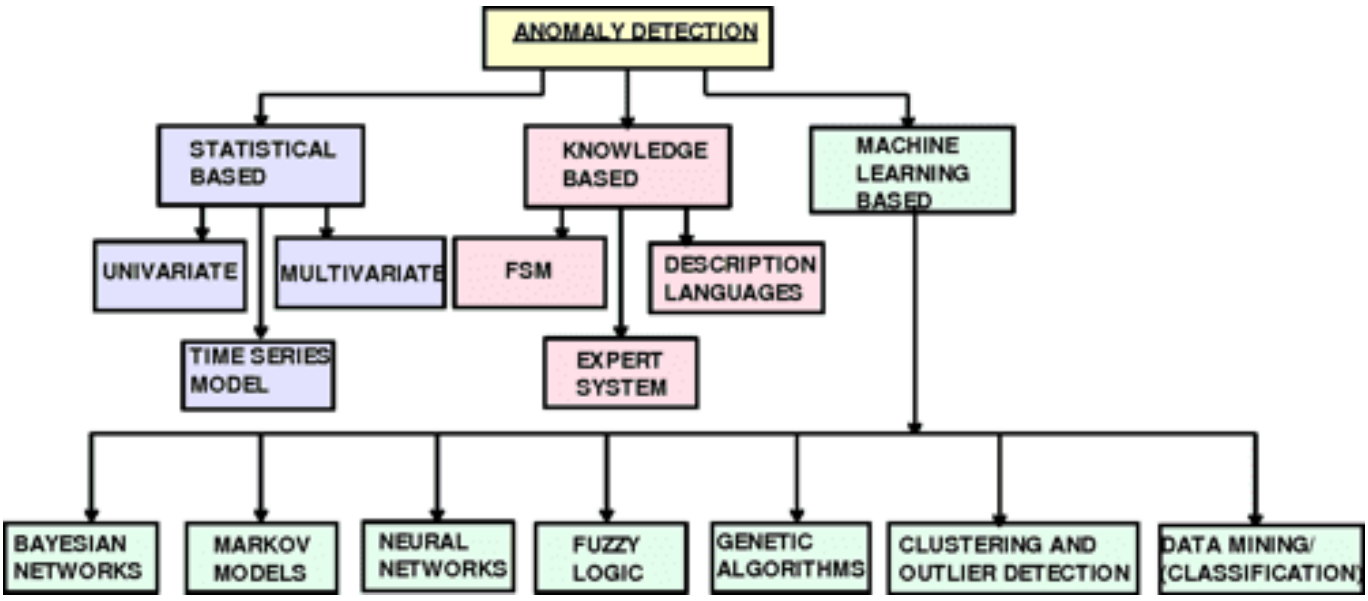


Figure 2.3: Classification of anomaly detection

2.3.1 Bayesian Network

A Bayesian network is a model that encodes probabilistic relationships among important variables. This technique is generally used for intrusion detection in combination with statistical schemes, a procedure that yields several advantages, including the capability of encoding interdependencies between variables and of predicting events, as well as the ability to incorporate both prior knowledge and data. Conditional probability P (A|B) is used for calculating the probability of at once the condition B is present. However, in the real world applications. one needs to know about the conditional probability P (B|A) for B once its evidence A is present. In this Bayes theorem, the goal is to calculate the probability of a given hypothesis H considering its sign or evidence E already exists. The H can be assumed to be a sampled column feature vector and noted as x = x1 , x2 , ... In the following text the E (Evidence) and the C (Class) sign can be replaced (where C = c1 ,c2 , . . .), if it makes it easier for the reader to understand the concept. The formula to calculate this probability is presented below

$$p\left(\frac{H}{E}\right) = \frac{P(H) * P\left(\frac{E}{H}\right)}{P(E)}$$

Where P (E |H) is the conditional probability of the evidence E once the hypothesis H is at hand. P(H) is the probability of the hypothesis H. P (E) is the probability of the evidence E. P (H |E) is the posterior probability of the hypothesis H once the evidence E is available. A framework of NIDS based on a Naïve Bayes algorithm is proposed in [4]. The framework constructs the patterns of the network services over data sets labeled by the services. The framework detects attacks in the datasets using the naïve Bayes Classifier algorithm using the built patterns. Compared to the Neural network-based approach, their approach achieves higher detection rate, less time consuming and has a low-cost factor. However, it generates more false positives. Naïve Bayesian network is a restricted network that has only two layers and assumes complete independence between the information nodes. This poses a limitation of this research work. In order to minimize this problem so as to reduce the false positives, active platform or event-based classification may be thought of using Bayesian networks. Researchers have designed several systems dealing with the problem of false alarms in recent years. The author proposed to use Bayesian networks to perform reasoning on complementary security evidence, and thus to potentially reduce false alert rates.

2.3.2 Markov models

There are two subtypes of Markov models: Markov chains and hidden Markov models. A Markov chain is a set of states that are interconnected through certain transition probabilities, which determine the topology and the capabilities of the model. During a first training phase, the probabilities associated with the transitions are estimated from the normal behavior of the target system. The detection of anomalies is then carried out by comparing the anomaly score (associated probability) obtained for the observed sequences with a fixed threshold. In the case of a hidden Markov model, the system of interest is assumed to be a Markov process in which states and transitions are hidden. Only the so-called productions are observable. Markov-based techniques have been extensively used in the context of host IDS, normally applied to system calls. A hybrid fuzzy-based anomaly IDS using hidden Markov model (HMM) detection engine and a normal database detection engine to reduce FAR is proposed in [3]. Development of host-based anomaly IDS has been studied with highlighting places on system call-based

HMM training explained in [11].

2.3.3 Neural networks

Artificial Neural Networks: - Inspired from known facts about how the brain works, researchers in the area of artificial intelligence (AI) have developed computational models which exhibit performance somewhat comparable to that of the brain [7]. Artificial neural networks (ANNs) are adaptive parallel.

distributed information processing models that consist of: (a) a set of simple processing units (nodes, neurons), (b) a set of synapses (connection weights), (c) the network architecture (pattern of connectivity), and (d) a learning process used to train the network. [5] Based on the advantages and disadvantages of the improved GA and LM algorithm, in this paper, the Hybrid Neural Network Algorithm (HNNA) is presented. Firstly, the algorithms use the advantage of the improved GA with strong whole searching capacity to search global optimal point in the whole question domain. Then, it adopts the strong point of the LM algorithm with fast local searching to find search near the global optimal point. The paper used respectively the three algorithms, namely the Improved GA, LM algorithm and HNNA, to adjust the input and output parameters of the ANN model, and adopt the theories of the fusion of the multi-classifiers to structure the Intrusion Detection System. By repeating an experiment, it is found that the HNNA is better in stability and convergence precision than LM algorithm and improved GA from the training result. The testing results are also proving that the detection rate of the multiple classifier intrusion detection system based on HNNA learning algorithm, including all attack categories that has a few or many training samples, is higher than the IDS that use LM and improved GA learning algorithm, and the false negative rate is less. So, the HNNA is proved to be feasible in theory and practice. In [6] according to the difference between the attack categories, they adjust the 41-dimensional input features of the neural-network-based multiple classifier intrusion detection system. After repeated experiment, they find that every adjusted sub-classifier is better in convergence precision, shorter in training time than the 41-features subclassing, moreover, the whole intrusion detection system is higher in the detection rate, and less in the false negative rate than the 41-features multiple classifier intrusion detection system. So, the scheme of the adjusting input features is able to optimize the neural-network-based multiple classifier intrusion detection system, and proved to be feasible in practice

2.3.4 Fuzzy logic techniques

Fuzzy logic is derived from fuzzy set theory under which reasoning is approximate rather than precisely deduced from classical predicate logic. Fuzzy techniques are thus used in the field of anomaly detection mainly because the features to be considered can be seen as fuzzy variables [1]. The application of fuzzy logic for computer security was first proposed in [8]. Fuzzy Intrusion Recognition Engine (FIRE) for detecting intrusion activities is proposed in [9] and the anomaly-based IDS is implemented using the data mining techniques and the fuzzy logic. The fuzzy logic part of the system is responsible for both handling the large number of input parameters and dealing with the inaccuracy of the input data. Three fuzzy characteristics used in this work are COUNT, UNIQUENESS and VARIANCE. The implemented fuzzy inference engine uses five fuzzy sets for each data element (HIGH, MEDIUM-HIGH, MEDIUM LOW and MEDIUM-LOW) and suitable fuzzy rules to detect the intrusion. In their report authors have not specified how they have derived their fuzzy set. The fuzzy set is a very important issue for the fuzzy inference engine and in some cases genetic approach can be implemented to select the best combination. The proposed system is tested using data collected from the local area network in the college of Engineering at Iowa State University and the results are reported in this paper. The reported results are descriptive and not numerical; therefore, it is difficult to evaluate the performance of the reported work.

2.3.5 Genetic algorithms

Genetic algorithms are classified as global search heuristics, and evolutionary computation that uses techniques inspired by evolutionary biology such as recombination, selection, inheritance and mutation. Thus, genetic algorithms represent another type of machine learning-based technique, capable of deriving classification rules [2] and/or selecting appropriate features or optimal parameters for the detection process [1] . In [10] rule evolution approach based on Genetic Programming (GP) for detecting novel attacks on networks is proposed. In their framework, four genetic operators, namely reproduction, mutation, crossover and dropping condition operators, are used to evolve new rules. New rules are used to detect novel or known network attacks. Experimental results show that rules generated by GPs with part of KDD 1999 Cup data set has a low false positive rate (FPR), a low false negative rate (FNR) and a high rate of detecting unknown attacks. However, an evaluation with full KDD training and testing data is missing in the paper. GA is used to generate genetic operators For producing useful and minimal structural modifications to the fuzzy expression tree represented by chromosomes. However, the training process in this approach is computationally very expensive and time consuming. Bridges and Vaughn

employ GA to tune the fuzzy membership functions and select an appropriate set of features in their intelligent intrusion detection system. GA as evolutionary algorithms was successfully used in different types of IDS. Using GA returned impressive results; the best fitness value was very close to the ideal fitness value. GA is a randomization search method often used for optimization problems. GA was successfully able to generate a model with the desired characteristics of high correct detection rate and low false positive rate for IDS.

2.3.6 Clustering and outlier detection

Clustering techniques work by grouping the observed data into clusters, according to a given similarity or distance measure. The procedure most commonly used for this consists in selecting a representative point for each cluster. Clustering techniques to determine the occurrence of intrusion events only from the raw audit data, and so the effort required to tune the IDS is reduced. One of the most popular and most widely used clustering algorithms is K-Means, which is a non hierarchical Centroid-based approach. Since in cyber infrastructure, a huge set of data is included, it is critical for IDS to learn the signatures and behaviors within this large set of data. Given ML and DL's powerful ability to study, they have played a vital role in decision-making and prediction generation in IDS [23]. In this section, several techniques and methods of ML and DL used in IDS are introduced and reviewed.

2.3.7 Examples of Machine Learning Techniques used on IDS

2.3.7.1 Neural network and decision tree used on IDS

Neural networks and decision trees are often used in SIDS previously [24], but this article applied these two methods to both SIDS and AIDS and therefore, they succeed to make an enhancement of IDS. The neural network was used to enhance the detection of known attacks, and a decision tree was used to detect new attacks. Backpropagation was a learning algorithm from a neural network, which was used in the three-layered structure. On different KDD 99 data sets, they performed two models on network-based IDS and AIDS. There were four attack classes in the KDD 99 contest: DoS, probing, U2R and R2L appended with one normal class, and they were represented by five neurons on the output layer respectively. After the experiment, although the neural network outperformed other works, it failed to distinguish U2R and R2L attacks, while the other two attacks are detected. Therefore, the decision tree is used to enhance it. With the aim of detect new threats, they enhanced the C4.5 program and 60.96 % was the percentage of detection rate that U2R class was enhanced. And the ratio of the false

negative detections of this class diminished from 82.89 % to 21, 93 %. But the lack of the percentage of successful prediction rates revealed that the improved C4.5 algorithm is not suitable to processing cases by relying on the proper labels.

2.3.7.2 SVM used on IDS

Proposed a method to apply support vector machine (SVM) into network-based IDS [25]. Numerous Kernel functions deployed with parameter C values of stabilization were used in this experiment under KDD 99 dataset, and several features were analyzed by using SVM. Firstly, the training dataset and selecting test dataset were performed to enhance the outcome of SVM IDS. And then, the pre-processing is performed to convert the dataset from only labelled as normal or attack name, to the criteria formula of SVM input. There are 4,898,431 sets under training in total, and one to the tenth of the labelled test set is 311,029 instances. After the pre-processing and training, the SVM learning and validation experiments can be performed. In terms of kernel functions, functions like linear, 2poly, and RBF were used and for the binary classification, SVMlight was deployed. Through the training process, SVM became a decision model, and several iterations are performed until a giant successful ratio are gained. Abundant kernel functions with C values were used to find the most suitable kernel, such as Radial Basis Functions, linear and 2-poly. During the test processes, the obtained results were compared with the KDD'99 results to verify the effect.

2.4 Challenges and Future Directions

Table 5 lists papers on machine learning based IDSs which are introduced in this survey. It shows that deep learning methods have become a research hotspot (26 papers are listed, 14 papers adopt deep learning methods). KDD99 and NSL-KDD datasets are still widespread used. Although machine learning methods have made great strides in the field of intrusion detection, the following challenges still exist. Table 5. Methods and papers on machine learning based IDSs.

Methods	Papers	Data Sources	Machine Learning Algorithms	Datasets
Packet parsing	Mayhew et al. [40]	Packet	SVM and K-means	Private dataset
	Hu et al. [41]	Packet	Fuzzy C-means	DARPA 2000
Payload analysis	Min et al. [43]	Packet	CNN	ISCX 2012
	Zeng et al. [44]	Packet	CNN, LSTM, and autoencoder	ISCX 2012
	Yu et al. [45]	Packet	Autoencoder	CTU-UNB
	Rigak et al. [46]	Packet	GAN	Private dataset
Statistic feature for flow	Goeschel et al. [47]	Flow	SVM, decision tree, and Naïve Bayes	KDD99
	Kuttranont et al. [48]	Flow	KNN	KDD99
	Peng et al. [13]	Flow	K-means	KDD99
Deep learning for flow	Potluri et al. [49]	Flow	CNN	NSL-KDD and UNSW-NB15
	Zhang et al. [50]	Flow	Autoencoder and XGBoost	NSL-KDD
	Zhang et al. [51]	Flow	GAN	KDD99
Traffic grouping	Teng et al. [52]	Flow	SVM	KDD99
	Ma et al. [53]	Flow	DNN	KDD99 and NSL-KDD
Statistic feature for session	Ahmim et al. [54]	Session	Decision tree	CICIDS 2017
	Alselari et al. [55]	Session	K-means	Private dataset
Sequence feature for session	Yuan et al. [56]	Session	CNN and LSTM	ISCX 2012
	Radford et al. [57]	Session	LSTM	ISCX IDS
	Wang et al. [58]	Session	CNN	DARPA 1998 and ISCX 2012
Rule-based	Meng et al. [59]	Log	KNN	Private dataset
	McElwee et al. [60]	Log	DNN	Private dataset
Log feature extraction with sliding window	Tran et al. [61]	Log	CNN	NGIDS-DS and ADFA-LD
	Tuor et al. [62]	Log	DNN and RNN	CERT Insider Threat
	Bohara et al. [63]	Log	K-means and DBSCAN	VAST 2011 Mini Challenge 2
Text analysis	Uwagbole et al. [64]	Log	SVM	Private dataset
	Vartouni et al. [65]	Log	Isolate forest	CSIC 2010 dataset

Figure 2.4: Methods and papers on machine learning based IDSs

(1) Lack of available datasets. The most widespread dataset is currently KDD99, which has many problems, and new datasets are required. However, constructing new datasets depends on expert knowledge, and the labor cost is high. In addition, the variability of the Internet environment intensifies the dataset shortage. New types of attacks are emerging, and some existing datasets are too old to reflect these new attacks. Ideally, datasets should include most of the common attacks and correspond to current network environments. Moreover, the available datasets should be representative, balanced and have less redundancy and less noise. Systematic datasets construction and incremental learning may be solutions to this problem.

(2) Inferior detection accuracy in actual environments. Machine learning methods have a certain ability to detect intrusions, but they often do not perform well on completely unfamiliar data. Most the existing studies were conducted using labeled datasets. Consequently, when the dataset does not cover all typical real-world samples, good performance in actual environments is not

guaranteed—even if the models achieve high accuracy on test sets.

(3) Low efficiency. Most studies emphasize the detection results; therefore, they usually employ complicated models and extensive data preprocessing methods, leading to low efficiency. However, to reduce harm as much as possible, IDSs need to detect attacks in real time. Thus, a trade-off exists between effect and efficiency. Parallel computing [66,67] approaches using GPUs [48,68,69] are common solutions.

From summarizing the recent studies, we can conclude that the major trends of IDS research lie in the following aspects.

(1) Utilizing domain knowledge. Combining domain knowledge with machine learning can improve the detection effect, especially when the goal is to recognize specific types of attacks in specific application scenarios.

- The rule-based detection methods have low false alarm rates but high missed alarm rates include considerable expert knowledge. In contrast, the machine learning methods usually have high false alarm rates and low missed alarm rates. The advantages of both methods are complementary. Combining machine learning methods with rule-based systems, such as Snort [70,71,72,73], can result in IDSs with low false alarm rates and low missed alarm rates.
- For specific types of attacks, such as DOS [74,75,76,77,78,79], botnet [80], and phishing web [81], proper feature must be extracted according to the attack characteristics that can be abstracted using domain knowledge.
- For specific application scenarios, such as cloud computing [82,83], IoT [84,85,86], and smart grids [87,88], domain knowledge can be used to provide the environmental characteristics that are helpful in data collection and data preprocessing.

(2) Improving machine learning algorithms. Improvements in machine learning algorithms are the main means to enhance the detection effect. Thus, studies involving deep learning and unsupervised learning methods has an increasing trend.

- Compared with shallow models, deep learning methods learn features directly from raw data, and their fitting ability is stronger. Deep learning models with deep structures can be used for classification, feature extraction, feature reduction, data denoising, and data augmentation tasks. Thus, deep learning methods can improve IDSs from many aspects.
- Unsupervised learning methods require no labeled data; thus they can be used even when a dataset shortage exists. The usual approach involves dividing data using an unsupervised learning model, manually labeling the clusters, and then training a classification model with supervised learning [89,90,91,92].

(3) Developing practical models. Practical IDSs not only need to have high detection accuracy but also high runtime efficiency and interpretability.

- In attack detection, the real-time requirement is essential. Thus, one research direction is to improve the efficiency of machine learning models. Reducing the time required for data collection and storage is also of concern.
- Interpretability is important for practical IDSs. Many machine learning models, especially deep learning models, are black boxes. These models report only the detection results and have no interpretable basis [93]. However, every cyber security decision should be made cautiously. An output result with no identifiable reason is not convincing. Thus, an IDS with high accuracy, high efficiency and interpretability is more practical.

Chapter 3

Example of an Intrusion Detection System Using Machine Learning Algorithms

Introduction

Problem Statement: The task is to build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections.

Dataset Used: KDD Cup 1999 dataset **Dataset Description:**

Data files:

- kddcup.names : A list of features.
- kddcup.data.gz: The full data set
- kddcup.data.10.percent.gz: A 10
- kddcup.newtestdata.10.percent.unlabeled.gz
- kddcup.testdata.unlabeled.gz
- kddcup.testdata.unlabeled.10.percent.gz
- corrected.gz: Test data with corrected labels.
- training.attack.types: A list of intrusion types.
- typo-correction.txt: A brief note on a typo in the data set that has been corrected.

The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between 'bad connections' (intrusion/attacks) and a 'good (normal) connections'. Attacks fall into four main categories:

- DOS: denial-of-service, e.g. syn flood;
- R2L: unauthorized access from a remote machine, e.g. guessing password;
- U2R: unauthorized access to local superuser (root) privileges, e.g., various "buffer overflow" attacks;
- probing: surveillance and another probing, e.g., port scanning.

Features:

feature name	description	type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Figure 3.1: Basic features of individual TCP connections

feature name	description	type
hot	number of “hot” indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of “compromised” conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if “su root” command attempted; 0 otherwise	discrete
num_root	number of “root” accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the “hot” list; 0 otherwise	discrete
is_guest_login	1 if the login is a “guest”login; 0 otherwise	discrete

Figure 3.2: Content features within a connection suggested by domain knowledge

feature name	description	type
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have "SYN" errors	continuous
rerror_rate	% of connections that have "REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have "SYN" errors	continuous
srv_rerror_rate	% of connections that have "REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Figure 3.3: Traffic features computed using a two-second time window

Various Algorithms Applied: Gaussian Naive Bayes, Decision Tree, Random Forest, Support Vector Machine, Logistic Regression.

Approach Used: I have applied various classification algorithms that are mentioned above on the KDD dataset and compare there results to build a predictive model.

3.1 Step 1 – Data Preprocessing:

Code: Importing libraries and reading features list from 'kddcup.names' file.

```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

Python

```python
print(os.listdir('C:\\Users\\admin\\OneDrive\\Desktop\\Intrusion Detection System'))
```

Python

```
['.git', '.ipynb_checkpoints', 'dataset', 'main.ipynb', 'test acc.png', 'test time.png', 'test_accuracy_figure.png', 'test_time_f
```

```python
with open("C:\\Users\\admin\\OneDrive\\Desktop\\Intrusion Detection System\\dataset\\kddcup.names",'r') as f:
    print(f.read())
```

Python

Code: Appending columns to the dataset and adding a new column name 'target' to the dataset

```
cols="""duration,
protocol_type,
service,
flag,
src_bytes,
dst_bytes,
land,
wrong_fragment,
urgent,
hot,
num_failed_logins,
logged_in,
num_compromised,
root_shell,
su_attempted,
num_root,
num_file_creations,
num_shells,
num_access_files,
num_outbound_cmds,
is_host_login,
is_guest_login,
count,
srv_count,
serror_rate,
srv_serror_rate,
```

```
srv_serror_rate,
rerror_rate,
srv_rerror_rate,
same_srv_rate,
diff_srv_rate,
srv_diff_host_rate,
dst_host_count,
dst_host_srv_count,
dst_host_same_srv_rate,
dst_host_diff_srv_rate,
dst_host_same_src_port_rate,
dst_host_srv_diff_host_rate,
dst_host_serror_rate,
dst_host_srv_serror_rate,
dst_host_rerror_rate,
dst_host_srv_rerror_rate"""

columns=[]
for c in cols.split(','):
    if(c.strip()):
        columns.append(c.strip())

columns.append('target')
#print(columns)
print(len(columns))
```

Output:

```
42
```

Code: Reading the 'attack-types' file.

```
with open("C:\\Users\\admin\\OneDrive\\Desktop\\Intrusion Detection System\\dataset\\training_attack_types",'r') as f:
    print(f.read())
```

Python

Output:

```
back dos
buffer_overflow u2r
ftp_write r2l
guess_passwd r2l
imap r2l
ipsweep probe
land dos
loadmodule u2r
multihop r2l
neptune dos
nmap probe
perl u2r
phf r2l
pod dos
portsweep probe
rootkit u2r
satan probe
smurf dos
spy r2l
teardrop dos
warezclient r2l
warezmaster r2l
```

Code: Creating a dictionary of attack-types

```
attacks_types = {
    'normal': 'normal',
'back': 'dos',
'buffer_overflow': 'u2r',
'ftp_write': 'r2l',
'guess_passwd': 'r2l',
'imap': 'r2l',
'ipsweep': 'probe',
'land': 'dos',
'loadmodule': 'u2r',
'multihop': 'r2l',
'neptune': 'dos',
'nmap': 'probe',
'perl': 'u2r',
'phf': 'r2l',
'pod': 'dos',
'portsweep': 'probe',
'rootkit': 'u2r',
'satan': 'probe',
'smurf': 'dos',
'spy': 'r2l',
'teardrop': 'dos',
'warezclient': 'r2l',
'warezmaster': 'r2l',
}
```

Code: Reading the dataset('kddcup.data.10.percent.gz') and adding Attack Type feature in the

training dataset where attack type feature has 5 distinct values i.e. dos, normal, probe, r2l, u2r.

```python
path = "C:\\Users\\admin\\OneDrive\\Desktop\\Intrusion Detection System\\dataset\\kddcup.data_10_percent.gz"
df = pd.read_csv(path,names=columns)

#Adding Attack Type column
df['Attack Type'] = df.target.apply(lambda r:attacks_types[r[:-1]])

df.head()
```

Python

Code: Shape of data-frame and getting data type of each feature

```python
df.shape
```

Python

Output:

```
(494021, 43)
```

Code: Finding missing values of all features.

```python
df['target'].value_counts()
```

Python

Output:

```
smurf.              280790
neptune.            107201
normal.              97278
back.                 2203
satan.                1589
ipsweep.              1247
portsweep.            1040
warezclient.          1020
teardrop.              979
pod.                   264
nmap.                  231
guess_passwd.           53
buffer_overflow.        30
land.                   21
warezmaster.            20
imap.                   12
rootkit.                10
loadmodule.              9
ftp_write.               8
multihop.                7
phf.                     4
perl.                    3
spy.                     2
Name: target, dtype: int64
```

No missing value found, so we can further proceed to our next step.

Code: Finding Categorical Features

```
#Finding categorical features
num_cols = df._get_numeric_data().columns

cate_cols = list(set(df.columns)-set(num_cols))
cate_cols.remove('target')
cate_cols.remove('Attack Type')

cate_cols
```

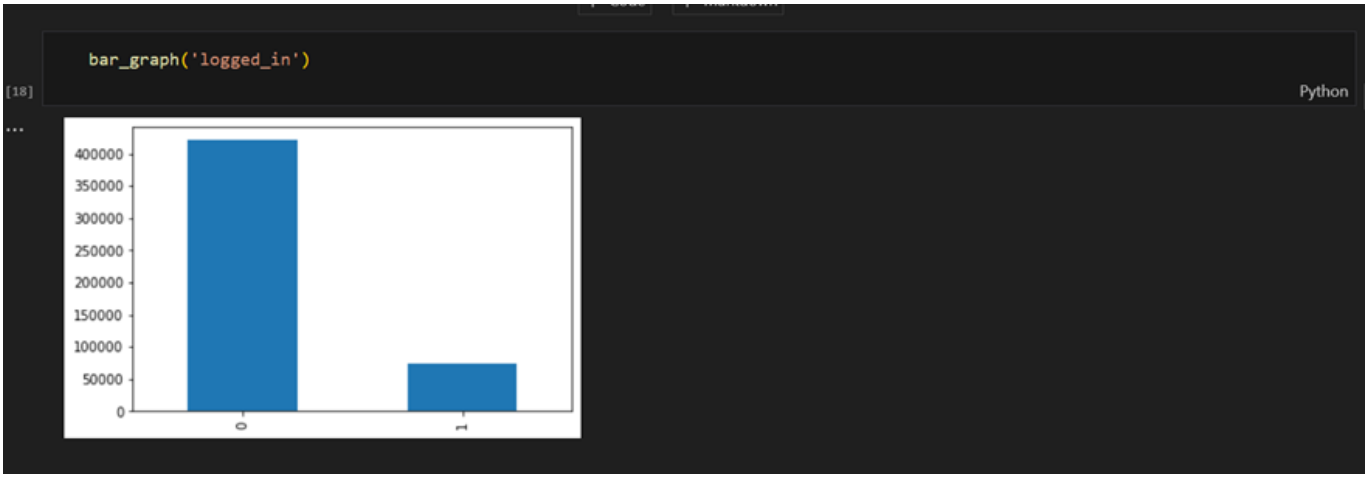
Python

Output:

```
['flag', 'service', 'protocol_type']
```

Visualizing Categorical Features using bar graph :

Protocol type: We notice that ICMP is the most present in the used data, then TCP and almost 20000 packets of UDP type



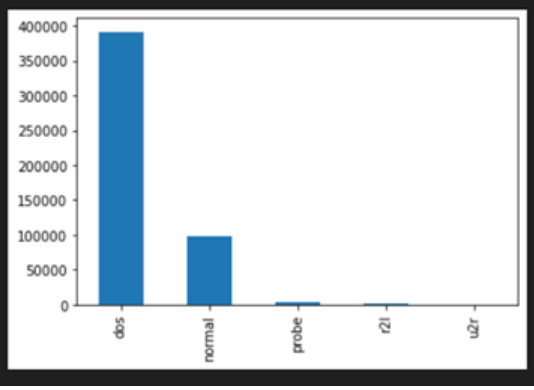
logged-in (1 if successfully logged in; 0 otherwise): We notice that just 70000 packets are successfully logged in.

Target Feature Distribution:

Attack Type(The attack types grouped by attack, it's what we will predict)

```
bar_graph('Attack Type')
```

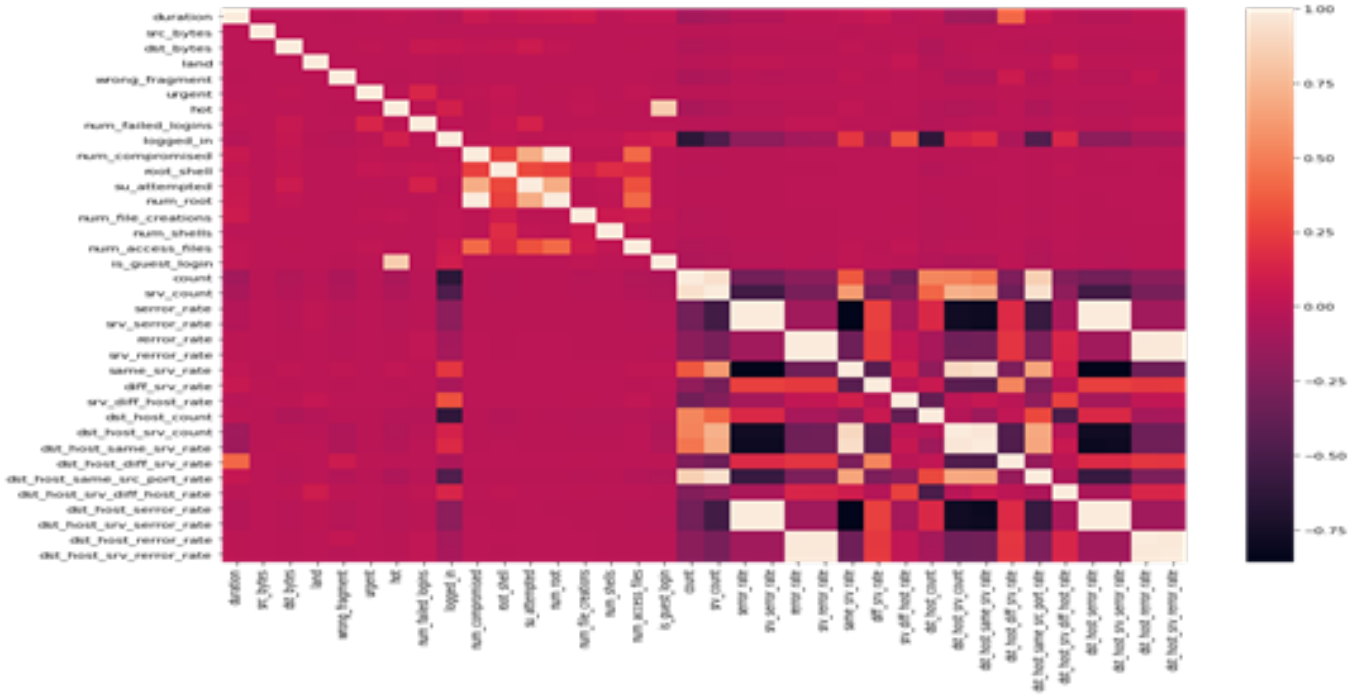
Python



Code: Data Correlation – Find the highly correlated variables using heatmap and ignore them for analysis.

```
df = df.dropna('columns')# drop columns with NaN

df = df[[col for col in df if df[col].nunique() > 1]]# keep columns where there are more than 1 unique values

corr = df.corr()

plt.figure(figsize=(15,12))

sns.heatmap(corr)

plt.show()
```

Python



Code:

```
#This variable is highly correlated with num_compromised and should be ignored for analysis.
#(Correlation = 0.9938277978738366)
df.drop('num_root',axis = 1,inplace = True)

#This variable is highly correlated with serror_rate and should be ignored for analysis.
#(Correlation = 0.9983615072725952)
df.drop('srv_serror_rate',axis = 1,inplace = True)

#This variable is highly correlated with rerror_rate and should be ignored for analysis.
#(Correlation = 0.9947309539817937)
df.drop('srv_rerror_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv_serror_rate and should be ignored for analysis.
#(Correlation = 0.9993041091850098)
df.drop('dst_host_srv_serror_rate',axis = 1, inplace=True)

#This variable is highly correlated with rerror_rate and should be ignored for analysis.
#(Correlation = 0.9869947924956001)
df.drop('dst_host_serror_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv_rerror_rate and should be ignored for analysis.
#(Correlation = 0.9821663427308375)
df.drop('dst_host_rerror_rate',axis = 1, inplace=True)

#This variable is highly correlated with rerror_rate and should be ignored for analysis.
#(Correlation = 0.9851995540751249)
df.drop('dst_host_srv_rerror_rate',axis = 1, inplace=True)
```

```
#This variable is highly correlated with dst_host_srv_count and should be ignored for analysis.
#(Correlation = 0.9865705438845669)
df.drop('dst_host_same_srv_rate',axis = 1, inplace=True)
```

Python

Code: Feature Mapping – Apply feature mapping on features such as : ‘protocol-type’ & ‘flag’.

```
df['protocol_type'].value_counts()
```

Python

```
icmp    283602
tcp     190065
udp      20354
Name: protocol_type, dtype: int64
```

```
#protocol_type feature mapping
pmap = {'icmp':0,'tcp':1,'udp':2}
df['protocol_type'] = df['protocol_type'].map(pmap)
```

Python

Code:

```python
#flag feature mapping
fmap = {'SF':0,'S0':1,'REJ':2,'RSTR':3,'RSTO':4,'SH':5 ,'S1':6 ,'S2':7,'RSTOS0':8,'S3':9 ,'OTH':10}
df['flag'] = df['flag'].map(fmap)
```

Python

```python
df.head()
```

Python

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	same_srv_rate	diff_srv_rate	srv
0	0	1	http	0	181	5450	0	0	0	0	...	1.0	0.0	
1	0	1	http	0	239	486	0	0	0	0	...	1.0	0.0	
2	0	1	http	0	235	1337	0	0	0	0	...	1.0	0.0	
3	0	1	http	0	219	1337	0	0	0	0	...	1.0	0.0	
4	0	1	http	0	217	2032	0	0	0	0	...	1.0	0.0	

Code: Remove irrelevant features such as 'service' before modelling

```python
df.drop('service',axis = 1,inplace= True)
```

Python

```python
df.shape
```

Python

```
(494021, 32)
```

```python
df.head()
```

Python

	duration	protocol_type	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	...	same_srv_rate	diff_sr
0	0	1	0	181	5450	0	0	0	0	0	...	1.0	
1	0	1	0	239	486	0	0	0	0	0	...	1.0	
2	0	1	0	235	1337	0	0	0	0	0	...	1.0	
3	0	1	0	219	1337	0	0	0	0	0	...	1.0	
4	0	1	0	217	2032	0	0	0	0	0	...	1.0	

5 rows × 32 columns

3.2 Step 2 – Modelling :

Code: Importing libraries and splitting the dataset

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
```

Python

Code:

```
df = df.drop(['target',], axis=1)
print(df.shape)

# Target variable and train set
Y = df[['Attack Type']]
X = df.drop(['Attack Type',], axis=1)

sc = MinMaxScaler()
X = sc.fit_transform(X)

# Split test and train data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
print(X_train.shape, X_test.shape)
print(Y_train.shape, Y_test.shape)
```

Python

Output:

```
(494021, 31)
(330994, 30) (163027, 30)
(330994, 1) (163027, 1)
```

Apply various machine learning classification algorithms such as Support Vector Machines, Random Forest, Naive Bayes, Decision Tree, Logistic Regression to create different models.

Code: Python implementation of Gaussian Naive Bayes

```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
```

Python

+ Code + Markdown

```
model1 = GaussianNB()
```

Python

```
start_time = time.time()
model1.fit(X_train, Y_train.values.ravel())
end_time = time.time()
```

Python

```
print("Training time: ",end_time-start_time)
```

Python

```
Training time:  1.0472145080566406
```

Code:

```
start_time = time.time()
Y_test_pred1 = model1.predict(X_test)
end_time = time.time()
```
Python

```
print("Testing time: ",end_time-start_time)
```
Python

```
Testing time:  0.7908921241760254
```

```
print("Train score is:", model1.score(X_train, Y_train))
print("Test score is:",model1.score(X_test,Y_test))
```
Python

```
Train score is: 0.8795114110829804
Test score is: 0.8790384414851528
```

Code: Python implementation of Decision Tree

DECISION TREE

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
```
Python

```
model2 = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
```
Python

```
start_time = time.time()
model2.fit(X_train, Y_train.values.ravel())
end_time = time.time()
```
Python

```
print("Training time: ",end_time-start_time)
```
Python

```
Training time:  1.504838466644287
```

```
start_time = time.time()
Y_test_pred2 = model2.predict(X_test)
end_time = time.time()
```
[68] Python

```
print("Testing time: ",end_time-start_time)
```
[69] Python

```
Testing time:  0.10471415519714355
```

```
print("Train score is:", model2.score(X_train, Y_train))
print("Test score is:",model2.score(X_test,Y_test))
```
[70] Python

```
Train score is: 0.9905829108684749
Test score is: 0.9905230421954646
```

Code: Python code implementation of Random Forest

RANDOM FOREST

```
[71] from sklearn.ensemble import RandomForestClassifier
```
Python

```
[72] model3 = RandomForestClassifier(n_estimators=30)
```
Python

```
[73] start_time = time.time()
     model3.fit(X_train, Y_train.values.ravel())
     end_time = time.time()
```
Python

```
[74] print("Training time: ",end_time-start_time)
```
Python

```
Training time:  11.453320026397705
```

```
[75] start_time = time.time()
     Y_test_pred3 = model3.predict(X_test)
     end_time = time.time()
```
Python

```
[76] print("Testing time: ",end_time-start_time)
```
Python

```
Testing time:  0.6096138954162598
```

```
[77] print("Train score is:", model3.score(X_train, Y_train))
     print("Test score is:",model3.score(X_test,Y_test))
```
Python

```
Train score is: 0.99997583037759
Test score is: 0.9996994362896944
```

Code: Python implementation of Support Vector Classifier

SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
```
78] Python

```
model4 = SVC(gamma = 'scale')
```
79] Python

```
start_time = time.time()
model4.fit(X_train, Y_train.values.ravel())
end_time = time.time()
```
80] Python

```
print("Training time: ",end_time-start_time)
```
81] Python

```
Training time:  126.96016049385071
```

```
start_time = time.time()
Y_test_pred4 = model4.predict(X_test)
end_time = time.time()
```
[82] Python

```
print("Testing time: ",end_time-start_time)
```
[83] Python

```
Testing time:  32.726547718048096
```

```
print("Train score is:", model4.score(X_train, Y_train))
print("Test score is:", model4.score(X_test,Y_test))
```
[84] Python

```
Train score is: 0.9987552644458811
Test score is: 0.9987916112055059
```

Code : Python implementation of Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
model5 = LogisticRegression(max_iter=1200000)
```

```
start_time = time.time()
model5.fit(X_train, Y_train.values.ravel())
end_time = time.time()
```

```
print("Training time: ",end_time-start_time)
```

Output :

```
Training time:  56.67286133766174
```

Code :

```
start_time = time.time()
Y_test_pred5 = model5.predict(X_test)
end_time = time.time()
```

```
print("Testing time: ",end_time-start_time)
```

Output :

```
Testing time:  0.0219895839691162 1
```

Code :

```
print("Train score is:", model5.score(X_train, Y_train))
print("Test score is:",model5.score(X_test,Y_test))
```

Output :

```
Train score is: 0.9935285835997028
Test score is: 0.993528679298521 1
```

Code : Python implementation of Gradient Descent

```
from sklearn.ensemble import GradientBoostingClassifier


model6 = GradientBoostingClassifier(random_state=0)


start_time = time.time()
model6.fit(X_train, Y_train.values.ravel())
end_time = time.time()


print("Training time: ",end_time-start_time)
```

Output :

```
Training time:  446.690997838974
```

Code :

```
start_time = time.time()
Y_test_pred6 = model6.predict(X_test)
end_time = time.time()


print("Testing time: ",end_time-start_time)
```

Output :

```
Testing time:  1.4141650199890137
```

Code :

```
print("Train score is:", model6.score(X_train, Y_train))
print("Test score is:", model6.score(X_test,Y_test))
```

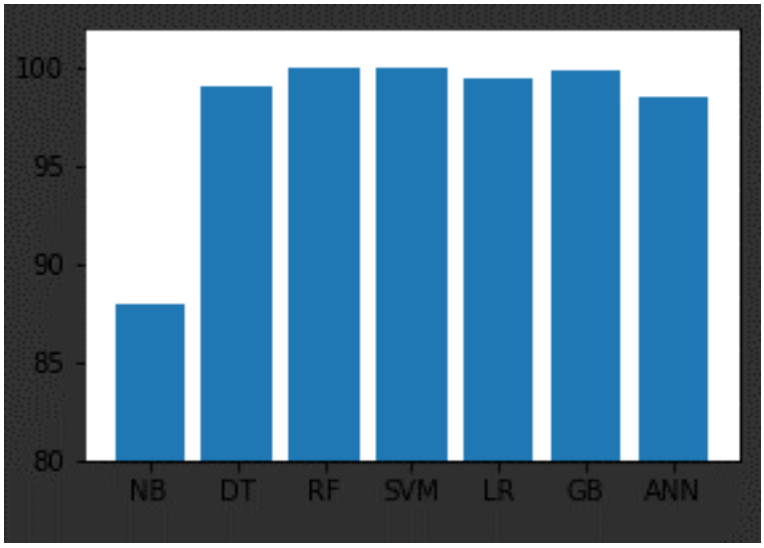
Output :

```
Train score is: 0.9979304760811374
Test score is: 0.9977181693829856
```

Code :

```
names = ['NB','DT','RF','SVM','LR','GB','ANN']
values = [87.951,99.058,99.997,99.875,99.352,99.793,98.485]
f = plt.figure(figsize=(15,3),num=10)
plt.subplot(131)
plt.ylim(80,102)
plt.bar(names,values)
```

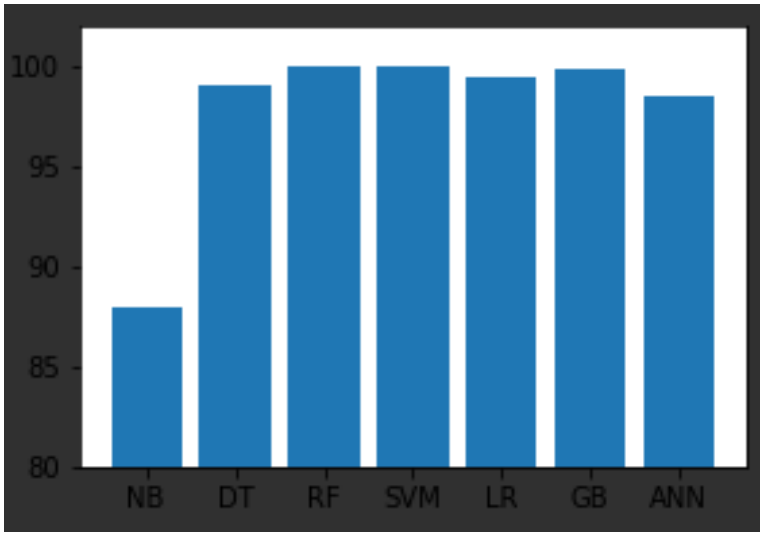
Output :



Code :

```
names = ['NB','DT','RF','SVM','LR','GB','ANN']
values = [87.903,99.052,99.969,99.879,99.352,99.771,98.472]
f = plt.figure(figsize=(15,3),num=10)
plt.subplot(131)
plt.ylim(80,102)
plt.bar(names,values)
```

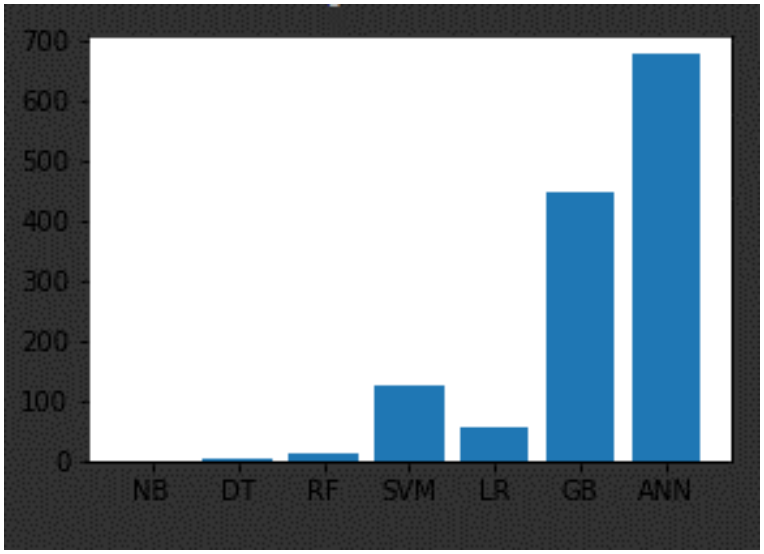
Output :



Code: Analyse the training and testing time of each model.

```
names = ['NB','DT','RF','SVM','LR','GB','ANN']
values = [1.04721,1.50483,11.45332,126.96016,56.67286,446.69099,674.12762]
f = plt.figure(figsize=(15,3),num=10)
plt.subplot(131)
plt.bar(names,values)
```

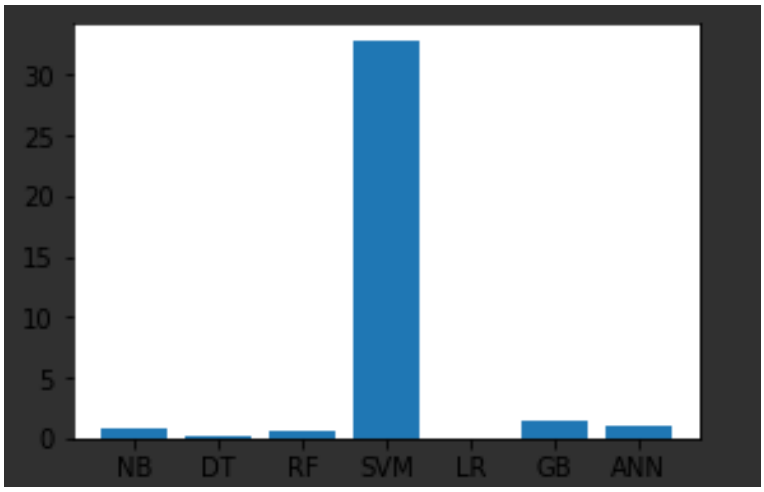
Output :



Code:

```
names = ['NB','DT','RF','SVM','LR','GB','ANN']
values = [0.79089,0.10471,0.60961,32.72654,0.02198,1.41416,0.96421]
f = plt.figure(figsize=(15,3),num=10)
plt.subplot(131)
plt.bar(names,values)
```

Output :



3.3 IDS Snort

We install snort and its configurations using the following command: **sudo apt-get install snort***

```
root@bilaldr:/home/bilaldr/Desktop# sudo apt-get install snort*
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Note : sélection de snort-pgsql pour l'expression rationnelle « snort* »
Note : sélection de snort-rules-default pour l'expression rationnelle « snort* »
Note : sélection de snort-mysql pour l'expression rationnelle « snort* »
Note : sélection de snort-doc pour l'expression rationnelle « snort* »
Note : sélection de snort-common-libraries pour l'expression rationnelle « snort* »
Note : sélection de snort-rules pour l'expression rationnelle « snort* »
Note : sélection de snort pour l'expression rationnelle « snort* »
Note : sélection de snort-common pour l'expression rationnelle « snort* »
Les paquets supplémentaires suivants seront installés :
  libdaq2t64 libdumbnet1 libluajit-5.1-2 libluajit-5.1-common libnetfilter-queue1 oinknaster
Les NOUVEAUX paquets suivants seront installés :
  libdaq2t64 libdumbnet1 libluajit-5.1-2 libluajit-5.1-common libnetfilter-queue1 oinknaster snort snort-common snort-common-libraries
  snort-doc snort-rules-default
```

We modify the configuration file **snort.conf** by adding the IP address and then we use the following command: **sudo nano /etc/snort/snort.conf**

```
# that you can run multiple instances.

###################################################
# Step #1: Set the network variables.  For more information, see README.variables
###################################################

# Setup the network addresses you are protecting
#
# Note to Debian users: this value is overriden when starting
# up the Snort daemon through the init.d script by the
# value of DEBIAN_SNORT_HOME_NET s defined in the
# /etc/snort/snort.debian.conf configuration file
#
ipvar HOME_NET 192.168.1.105

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any
# If HOME_NET is defined as something other than "any", alternative, you can
# use this definition if you do not want to detect attacks from your internal
# IP addresses:
#ipvar EXTERNAL_NET !$HOME_NET

# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET

# List of web servers on your network

^G Aide      ^O Écrire      ^W Chercher    ^K Couper      ^T Exécuter    ^C Emplacement  M-U Annuler    M-A Marquer
^X Quitter   ^R Lire fich.  ^\ Remplacer   ^U Coller      ^J Justifier   ^/ Aller ligne  M-E Refaire    M-6 Copier
```

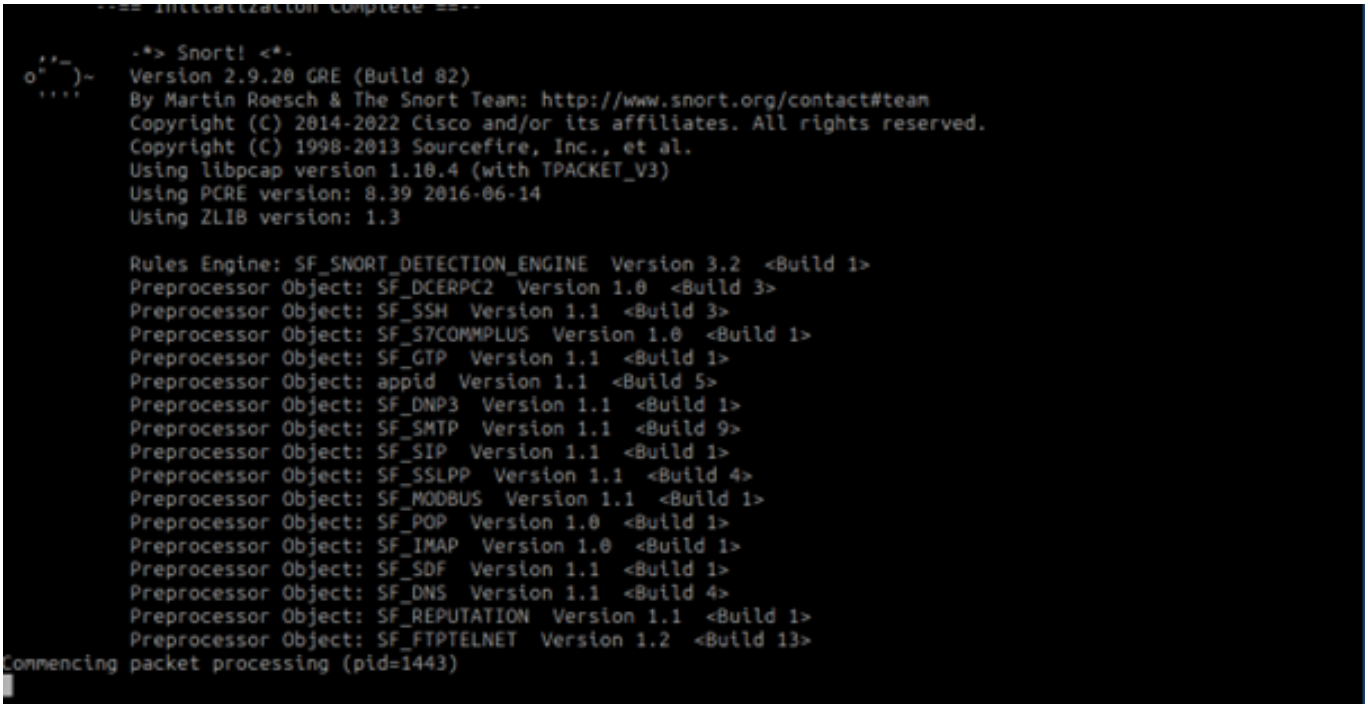
To make sure that Snort is installed, we use the following command: **snort –version**

```
root@bilaldr:/home/bialaldr/Desktop# snort --version

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.20 GRE (Build 82)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.4 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.3

root@bilaldr:/home/bialaldr/Desktop#
```

To verify our configuration, we access the **/etc/snort/rules/** and execute this command **ls -la**. And we get the following:

```
===============================================================================
Snort exiting
root@bilaldr:/home/bialaldr/Desktop# cd /etc/snort/rules/
root@bilaldr:/etc/snort/rules# ls -al
total 1616
drwxr-xr-x 2 root root  12288 01:04 25    ماي .
drwxr-xr-x 3 root root   4096 01:09 25    ماي ..
-rw-r--r-- 1 root root   5520 13:32 19  بريل  attack-responses.rules
-rw-r--r-- 1 root root  17898 13:32 19  بريل  backdoor.rules
-rw-r--r-- 1 root root   3862 13:32 19  بريل  bad-traffic.rules
-rw-r--r-- 1 root root   7994 13:32 19  بريل  chat.rules
-rw-r--r-- 1 root root  12759 13:32 19  بريل  community-bot.rules
-rw-r--r-- 1 root root   1223 13:32 19  بريل  community-deleted.rules
-rw-r--r-- 1 root root   2042 13:32 19  بريل  community-dos.rules
-rw-r--r-- 1 root root   2176 13:32 19  بريل  community-exploit.rules
-rw-r--r-- 1 root root    249 13:32 19  بريل  community-ftp.rules
-rw-r--r-- 1 root root   1376 13:32 19  بريل  community-game.rules
-rw-r--r-- 1 root root    689 13:32 19  بريل  community-icmp.rules
-rw-r--r-- 1 root root   2777 13:32 19  بريل  community-imap.rules
-rw-r--r-- 1 root root    948 13:32 19  بريل  community-inappropriate.rules
-rw-r--r-- 1 root root    257 13:32 19  بريل  community-mail-client.rules
-rw-r--r-- 1 root root   7837 13:32 19  بريل  community-misc.rules
-rw-r--r-- 1 root root    621 13:32 19  بريل  community-nntp.rules
-rw-r--r-- 1 root root    775 13:32 19  بريل  community-oracle.rules
```

Before testing Snort, we execute the following command: **snort -A console -c /etc/snort/snort.conf** to activate snort and to start capturing packets.



To test snort, we launch a DDOS attack from the kali Linux machine onto the ubuntu machine. The ubuntu machine has the following IP: 192.168.1.105

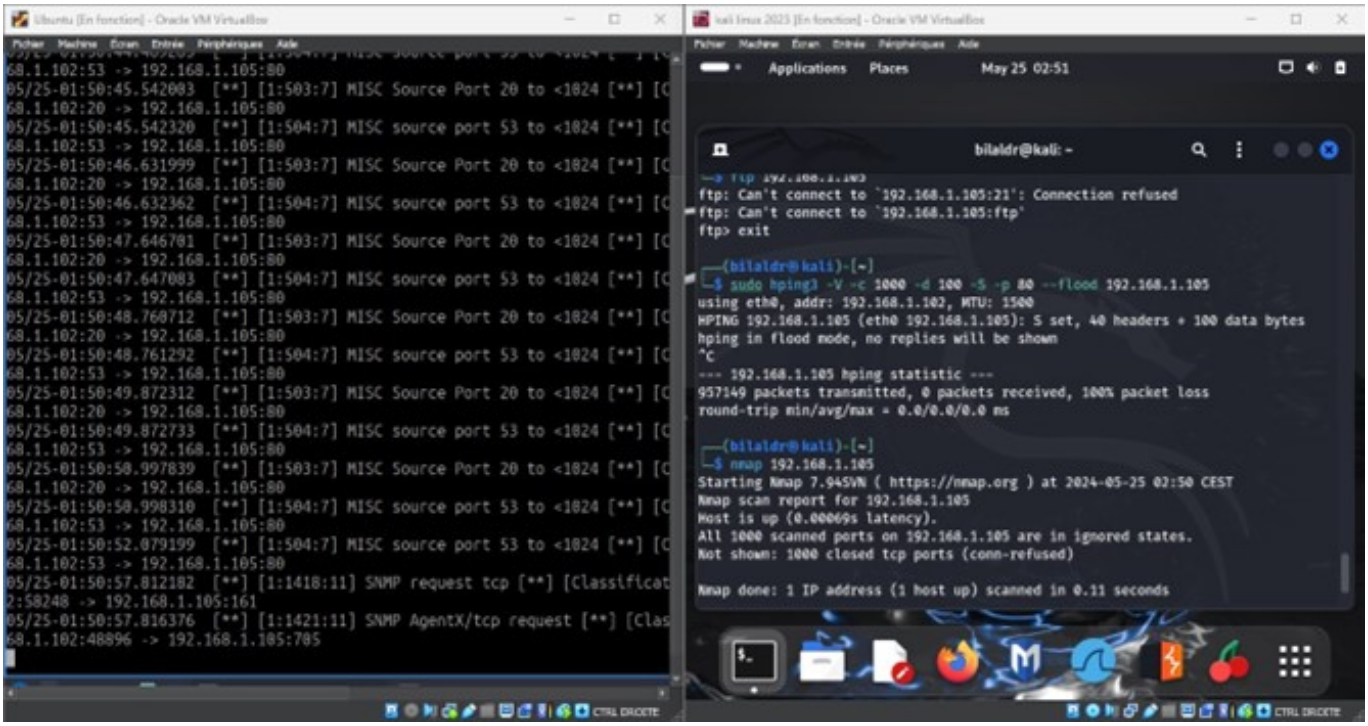
To launch the DDOS attack we use the following command:

sudo hping3 -V -c 1000 -d 100 -S -p 80 –flood 192.168.1.105

hping3 command breakdown:

- -V: Verbose mode
- -c 1000: Send 1000 packets.
- -d 100: Packet size (100 bytes).
- -S: Use a TCP packer with SYN flag activated.
- -p 80: Target port 80
- –flood: Send packets as fast as possible

Then we will scan the 192.168.1.105 IP address using NMAP: **Nmap 192.168.1.105**



Interpretation of results:

Snort was successful at detecting the DDoS attack and the NMAP scan. This confirms that Snort is able to identify malicious activity on the network.

Bibliography

[1] Bridges, Vaughn, "Fuzzy Data mining and genetic algorithms applied to intrusion detection," In: Proceedings of the National Information Systems Security Conference; 2000. pp. 13–31.

[2] Li W. "Using genetic algorithm for network intrusion detection," C.S.G. Department of Energy; 2004. pp. 1–8.

[3] X.D. Hoang, J. Hu, P. Bertok, "A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference," Journal of Network and Computer Applications 32 (2009) 1219–1228.

[4] Mrutyunjaya Panda, and Manas Ranjan Patra " NETWORK INTRUSION DETECTION USING NAÏVE BAYES ", IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.12, December 2007

[5] Li Xiangmei Qin Zhi "The Application of Hybrid Neural Network Algorithms in Intrusion Detection System "978-1-4244-8694-6/11 ©2011 IEEE

[6] Xiangmei Li ,"Optimization of the Neural-NetworkBased Multiple Classifiers Intrusion Detection System ",978-1-4244-5143-2/10 ©2010 IEEE

[7] Naeem Seliya Taghi M. Khoshgoftaar, "Active Learning with Neural Networks for Intrusion Detection", IEEE IRI 2010, August 4-6, 2010, Las Vegas, Nevada, USA 978- 1-4244-8099-9/10

[8] H.H. Hosmer, Security is fuzzy!: applying the fuzzy logic paradigm to the multipolicy paradigm, Proceedings of the 1992-1993 workshop on New security paradigms, ACM New York, NY, USA, 1993, pp. 175- 184.

[9] John E. Dickerson and Julie A. Dickerson, Fuzzy network profiling for intrusion detection, Proceedings of NAFIPS 19th International Conference of the North American Fuzzy Infor mation Processing Society (Atlanta, USA), July 2000, pp. 301-306.

[10] T.Lunt and I.Traore, Unsupervised Anomaly Detection Using an Evolutionary Extension of K-means Algorithm,International Journal on Information and computer Science, Inderscience Pulisher 2 (May, 2008), 107-139.

[11] Jiankun Hu, Xinghuo Yu, Qiu D, Hsiao-Hwa Chen; "A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection," IEEE Transaction on Network, Volume: 23, Issue: 1 DOI: 10.1109/MNET.2009.4804323, Year: 2009, Page(s): 42 – 47.

[12]

[13] Peng, K.; Leung, V.C.; Huang, Q. Clustering approach based on mini batch kmeans for intrusion detection system over big data. IEEE Access 2018, 6, 11897–11906. [Google Scholar] [CrossRef]

[14] **ref14**

[15] Cisco Security Professional's Guide to Secure Intrusion Detection Systems

[16] **ref16**

[17] CS406: Information Security

[18] **ref18** (Anomaly-Based Intrusion Detection System Veeramreddy Jyothsna and Koneti Munivara Prasad)

[19] **ref19**

[20] **ref20**

[21] **ref21**

[40] Mayhew, M.; Atighetchi, M.; Adler, A.; Greenstadt, R. Use of machine learning in big data analytics for insider threat detection. In Proceedings of the MILCOM 2015—2015 IEEE Military Communications Conference, Canberra, Australia, 10–12 November 2015; pp. 915–922. [Google Scholar]

[41] Hu, L.; Li, T.; Xie, N.; Hu, J. False positive elimination in intrusion detection based on clustering. In Proceedings of the 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, China, 15–17 August 2015; pp. 519–523. [Google Scholar]

[42] Liu, H.; Lang, B.; Liu, M.; Yan, H. CNN and RNN based payload classification methods for attack detection. Knowl.-Based Syst. 2019, 163, 332–341. [Google Scholar] [CrossRef]

[43] Min, E.; Long, J.; Liu, Q.; Cui, J.; Chen, W. TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest. Secur. Commun. Netw. 2018, 2018, 4943509. [Google Scholar] [CrossRef]

[44] Zeng, Y.; Gu, H.; Wei, W.; Guo, Y. Deep-Full-Range: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. IEEE Access 2019, 7, 45182–45190. [Google Scholar] [CrossRef]

[45] Yu, Y.; Long, J.; Cai, Z. Network intrusion detection through stacking dilated convolutional autoencoders. Secur. Commun. Netw. 2017, 2017, 4184196. [Google Scholar] [CrossRef]

[46] Rigaki, M.; Garcia, S. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 70–75. [Google Scholar]

[47] Goeschel, K. Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive Bayes for off-line analysis. In Proceedings of the SoutheastCon 2016, Norfolk, VA, USA, 30 March–3 April 2016; pp. 1–6. [Google Scholar]

[48] Kuttranont, P.; Boonprakob, K.; Phaudphut, C.; Permpol, S.; Aimtongkhamand, P.; KoKaew, U.; Waikham, B.; So-In, C. Parallel KNN and Neighborhood Classification Implementations on GPU for Network Intrusion Detection. J. Telecommun. Electron. Comput. Eng. 2017, 9, 29–33. [Google Scholar]

[49] Potluri, S.; Ahmed, S.; Diedrich, C. Convolutional Neural Networks for Multi-class Intrusion Detection System. In Mining Intelligence and Knowledge Exploration; Springer: Cham, Switzerland, 2018; pp. 225–238. [Google Scholar]

[50] Zhang, B.; Yu, Y.; Li, J. Network Intrusion Detection Based on Stacked Sparse Autoencoder and Binary Tree Ensemble Method. In Proceedings of the 2018 IEEE International Conference on Communications Workshops (ICC Workshops), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [Google Scholar]

[51] Zhang, H.; Yu, X.; Ren, P.; Luo, C.; Min, G. Deep Adversarial Learning in Intrusion Detection: A Data Augmentation Enhanced Framework. arXiv 2019, arXiv:1901.07949. [Google Scholar]

[52] Teng, S.; Wu, N.; Zhu, H.; Teng, L.; Zhang, W. SVM-DT-based adaptive and collaborative intrusion detection. IEEE/CAA J. Autom. Sin. 2017, 5, 108–118. [Google Scholar] [CrossRef]

[53] Ma, T.; Wang, F.; Cheng, J.; Yu, Y.; Chen, X. A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. Sensors 2016, 16, 1701. [Google Scholar] [CrossRef]

[54] Ahmim, A.; Maglaras, L.; Ferrag, M.A.; Derdour, M.; Janicke, H. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019; pp. 228–233. [Google Scholar]

[55] Alseiari, F.A.A.; Aung, Z. Real-time anomaly-based distributed intrusion detection systems for advanced Metering Infrastructure utilizing stream data mining. In Proceedings of the 2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE), Offenburg, Germany, 20–23 October 2015; pp. 148–153. [Google Scholar]

[56] Yuan, X.; Li, C.; Li, X. DeepDefense: identifying DDoS attack via deep learning. In Proceedings of the 2017 IEEE International Conference on Smart Computing (SMARTCOMP), Hong Kong, China, 29–31 May 2017; pp. 1–8. [Google Scholar]

[57] Radford, B.J.; Apolonio, L.M.; Trias, A.J.; Simpson, J.A. Network traffic anomaly detection using recurrent neural networks. arXiv 2018, arXiv:1803.10769. [Google Scholar]

[58] Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. IEEE Access 2017, 6, 1792–1806. [Google Scholar] [CrossRef]

[59] Meng, W.; Li, W.; Kwok, L.F. Design of intelligent KNN-based alarm filter using knowledge-based alert verification in intrusion detection. Secur. Commun. Netw. 2015, 8, 3883–3895. [Google Scholar] [CrossRef]

[60] McElwee, S.; Heaton, J.; Fraley, J.; Cannady, J. Deep learning for prioritizing and responding to intrusion detection alerts. In Proceedings of the MILCOM 2017—2017 IEEE Military Communications Conference (MILCOM), Baltimore, MD, USA, 23–25 October 2017; pp. 1–5. [Google Scholar]

[61] Tran, N.N.; Sarker, R.; Hu, J. An Approach for Host-Based Intrusion Detection System Design Using Convolutional Neural Network. In Proceedings of the International Conference on Mobile

Networks and Management, Chiba, Japan, 23–25 September 2017; Springer: Berlin, Germany, 2017; pp. 116–126. [Google Scholar]

[62] Tuor, A.; Kaplan, S.; Hutchinson, B.; Nichols, N.; Robinson, S. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In Proceedings of the Workshops at the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017. [Google Scholar]

[63] Bohara, A.; Thakore, U.; Sanders, W.H. Intrusion detection in enterprise systems by combining and clustering diverse monitor data. In Proceedings of the Symposium and Bootcamp on the Science of Security, Pittsburgh, PA, USA, 19–21 April 2016; pp. 7–16. [Google Scholar]

[64] Uwagbole, S.O.; Buchanan, W.J.; Fan, L. Applied machine learning predictive analytics to SQL injection attack detection and prevention. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 1087–1090. [Google Scholar]

[65] Vartouni, A.M.; Kashi, S.S.; Teshnehlab, M. An anomaly detection method to detect web attacks using Stacked Auto-Encoder. In Proceedings of the 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), Kerman, Iran, 28 February–2 March 2018; pp. 131–134. [Google Scholar]

. . . .