FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Security monitoring of Active Directory environment based on Machine Learning techniques
Student:	Bc. Lukáš Kotlaba
Supervisor:	Ing. Simona Buchovecká, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2021/2022

Instructions

The goal of the thesis is to study the possibility of Machine Learning techniques for security monitoring of Microsoft Active Directory and their implementation in the Splunk tool. Further, the performance of the Machine Learning techniques should be compared to "traditional" signature -based rules.

1. Get familiar with the possibilities of Splunk tool with regards to usage of Machine Learning techniques and identify the algorithms suitable to be used for security monitoring

2. Identify the features from Windows Security Audit log that are suitable for the security monitoring based on selected algorithms

3. Develop a set of detection rules for security monitoring of Microsoft Active Directory based on the selected algorithms

---

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 17 January 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Security Monitoring of Active Directory Environment Based on Machine Learning Techniques

Bc. Lukáš Kotlaba

Department of Information Security
Supervisor: Ing. Simona Buchovecká, Ph.D.

May 5, 2021

Acknowledgements

First of all, I would like to express gratitude to my supervisor Ing. Simona Buchovecká, Ph.D. for all her help and guidance, not only during the creation of this thesis but also throughout my research to date. I would like to thank my colleagues, who made it possible for me to advance professionally and helped to shape my interests in the security field. Most importantly, special thanks go to my family and closest friends, who have been an endless source of support and encouragement.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 5, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Kotlaba, Lukáš. *Security Monitoring of Active Directory Environment Based on Machine Learning Techniques.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstract

Active Directory is a central point of administration and identity management in many organizations. Ensuring its security is indispensable to protect user credentials, enterprise systems, and sensitive data from unauthorized access. Security monitoring of Active Directory environments is typically performed using signature-based detection rules. However, those are not always effective and sufficient, especially for attacks similar to legitimate activity from the auditing perspective. This thesis applies machine learning techniques for detecting two such attack techniques – Password Spraying and Kerberoasting. Several machine learning algorithms are utilized based on features from Windows Event Log and evaluated on data originating from a real Active Directory environment. Best approaches are implemented as detection rules for practical use in the Splunk platform. In experimental comparison with signature-based approaches, the proposed solution was able to improve detection capabilities, and at the same time, reduce the number of false alarms for both considered attack techniques.

Keywords security monitoring, detection rules, machine learning, anomaly detection, Active Directory, Password Spraying, Kerberoasting, Splunk

Abstrakt

Active Directory je nástrojem centralizované administrace a správy identit v mnoha organizacích. Zajištění jeho zabezpečení je nezbytné k ochraně přístupových dat uživatelů, podnikových systémů a citlivých dat před neoprávněným přístupem. Bezpečnostní monitorování prostředí Active Directory se obvykle provádí pomocí detekčních pravidel založených na signaturách. Ty však nejsou vždy účinné a dostatečné, zejména pro útoky, které jsou podobné legitimním aktivitám z hlediska auditních dat. Tato práce aplikuje techniky strojového učení pro detekci dvou takových útočných technik – Password Spraying a Kerberoasting. Algoritmy strojového učení jsou aplikovány s využitím příznaků z auditu událostí systému Windows a vyhodnoceny na datech pocházejících ze skutečného Active Directory prostředí. Nejlepší přístupy jsou implementovány jako detekční pravidla pro praktické použití na platformě Splunk. Navrhované řešení dokázalo zlepšit detekční schopnosti a současně snížit počet falešných poplachů ve srovnání s přístupy založenými na signaturách, a to pro obě zkoumané techniky útoků.

Klíčová slova bezpečnostní monitorování, detekční pravidla, strojové učení, detekce anomálií, Active Directory, Password Spraying, Kerberoasting, Splunk

Contents

List of Figures

List of Tables

List of Listings

Introduction

Microsoft Active Directory is a foundation for identity management and centralized administration of domain networks. Given the prevalence of Microsoft Windows systems, Active Directory has become an integral part of many enterprise networks. Moreover, following the nowadays trends, Active Directory services have been integrated into cloud environments.

Active Directory plays a critical role in the network infrastructure. Due to the sensitivity of data it holds, it represents an interesting target for cyber attackers. Potential compromise of Active Directory may have a severe impact and can undermine the integrity of the whole domain. Security monitoring of Active Directory is therefore crucial for protecting the organizational network.

Attacks targeting Active Directory are typically detected by rules that contain specific conditions or signatures of known attack techniques. These rules are used to analyze relevant log data, and in case the conditions are met, security alerts are generated.

However, traditional detection approaches are not always sufficient. They may produce many false alarms or miss actual attacks due to adversaries' ability to evade detection. This thesis studies the possibilities of applying machine learning techniques for detecting the attacks, focusing on improving the detection capabilities and reducing the number of false alarms.

The proposed detection approach is implemented in the Splunk platform, a tool commonly used in practice. Outputs of this thesis will help organizations improve security monitoring of their Active Directory deployments and security professionals to develop new detections for other attacks based on machine learning techniques.

The thesis is organized as follows: chapter 1 introduces the fundamental concepts of Active Directory, with emphasis on its security aspects and threats. It is followed by a detailed analysis of the selected attack techniques, provided in chapter 2. Chapter 3 encompasses research of machine learning approaches, their utilization in security monitoring, and machine learning support in Splunk technology. An approach is proposed based on the findings. Chapter 4 describes the realization process, evaluation of the machine learning methods, and the practical implementation. Finally, chapter 5 presents the obtained results.

This thesis continues the topic of my bachelor's thesis, focused on developing detection rules for attacks targeting Active Directory. As the efficiency of the developed rules varied, this thesis aims to improve the detection mechanisms by applying machine learning techniques and evaluate the results on data originating from a real Active Directory environment.

Goals

The main goal of this thesis is to study the possibilities of applying machine learning techniques for security monitoring of Active Directory and based on suitable algorithms develop a set of detection rules in Splunk technology.

The theoretical part aims to identify attacks targeting Active Directory whose detection using signature-based methods is not sufficient and could be improved using machine learning techniques.

Further, it aims to study machine learning approaches in relation to security monitoring and review the existing applications, particularly for detecting threats related to Active Directory. Next, analyze the options of using machine learning techniques in the Splunk platform, and based on the findings, choose algorithms feasible for realization.

The goal of the practical part is to propose and develop a monitoring solution for the selected attacks based on machine learning. An important part is identifying appropriate attributes from Windows security audit log and their transformation into features suitable for the determined algorithms. The algorithms will be utilized in detection rules, designed and implemented for use in the Splunk platform.

Finally, detection efficiency of the proposed solution shall be compared to a signature-based approach and its possible benefits or drawbacks assessed.

Chapter 1

Active Directory Security

Active Directory has become the cornerstone of many network environments, and its protection from security threats a necessity. This chapter introduces the basic concepts of Active Directory technology, its role in the authentication processes, and its native security auditing features. Further, adversary tactics and techniques targeting Active Directory are overviewed, together with the possibilities of their detection.

1.1 Active Directory Background

Active Directory (AD) is a directory service developed by Microsoft for Windows network environments. It is based on Lightweight Directory Access Protocol (LDAP), a standard protocol for directory services. AD forms a hierarchical structure that stores information about objects on the network, which typically include user accounts, computers, shared folders, printers, and many others. [1]

Active Directory is provided as a set of services that are part of the Microsoft Windows Server operating system (OS). As described in [2], AD services can be installed as multiple server roles, while the most important roles include:

Active Directory Domain Services (AD DS) comprise the core AD functionality for storing directory data and making it available to network users and administrators. AD DS provide a broad range of identity-related services, such as centralized identity management, authentication, authorization, single sign-on (SSO) capabilities, access control, or user rights management. AD DS also allow for centralized policy-based administration of the network environment.

Active Directory Federation Services (AD FS) extend the SSO functionality of AD DS to Internet-facing applications. Federated identity allows for consistent user experience while accessing the web-based applications of an organization, even when not on a corporate network.

Active Directory Lightweight Directory Services (AD LDS) represent an independent mode of AD without its infrastructure features. It functions as a standalone

application service that can be deployed alongside AD and operate independently. AD LDS offer a simplistic version of AD DS, providing directory services for applications that do not require full AD infrastructure.

AD DS are the most fundamental AD services[1]. A Windows server running AD DS role is called a *domain controller (DC)*. Domain controllers form the physical structure of Active Directory. They host all of the AD functionality and maintain the AD multi-master database, which is replicated between multiple DCs in the environment.

The logical structure of AD is built around the concept of *domains*, commonly referred to as Windows domains or AD domains. A domain represents an administrative and security boundary for the objects inside it. Domains can be organized into *domain trees* and those further into *forests*, building a hierarchical structure. On a smaller scale, objects inside domains can be organized into containers. The most common type of container is an *organizational unit (OU)*. Members of an OU may be objects such as users, groups, computers, or other OUs. Logical structure of AD is illustrated in figure 1.1. [1]

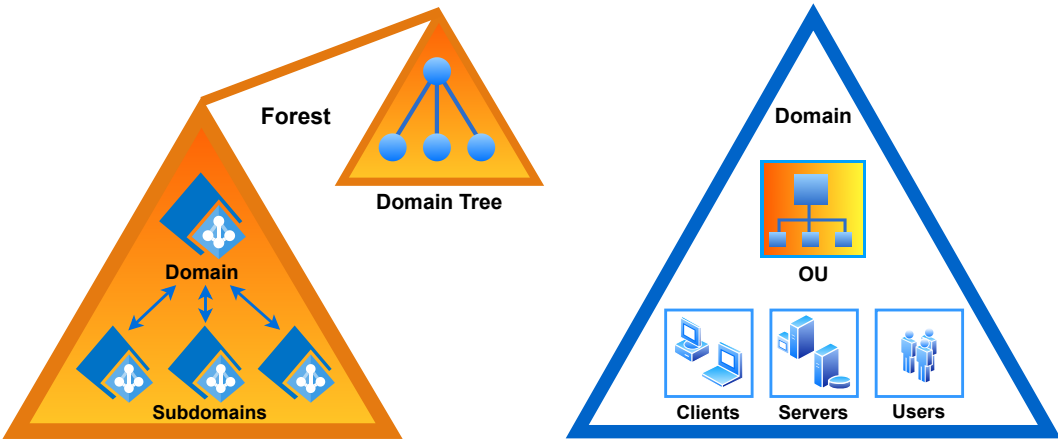

Figure 1.1: Logical structure of Active Directory

Administrators can control the behavior of AD objects via *Group Policy*. Group Policy allows managing various configurations of the objects, including their security settings. The logical structure of AD facilitates administration of the domain, as Group Policy can be applied to containers, such as OUs or domains, rather than individual objects. [1]

1.2 Authentication in Active Directory

Besides storing identity-related information, Active Directory serves as a foundation for authentication services in a domain environment. Authentication is a process for verifying the identity of an object or person. Its purpose is to validate that the party is genuine and

[1]In fact, AD DS are commonly understood under the sole term Active Directory, and hence these terms are not strictly distinguished throughout this thesis.

is truly who they claim to be. It is not to be confused with authorization, which is the act of determining the correct permissions and granting access to the requested resources. [3]

In Windows OS, any user, service, or computer that can initiate an action is a *security principal*. Security principals are uniquely identified by *security identifiers (SIDs)* and have accounts, which can be local to a computer or domain-based, stored in AD. Before a security principal can participate in a network domain, its account must be authenticated towards AD. The principal must provide some form of secret authentication data, such as a certificate or password, to authenticate. [4]

Following is the explanation of the authentication process in AD, based on Microsoft documentation [4]. For simplification, a user identity is assumed. To authenticate users, Windows implements *interactive logon* process. In order to log on, a user enters credentials, typically username and password, into the *Log On to Windows* dialog box. This is implemented by *Graphical Identification and Authentication (GINA)* component that is loaded by *Winlogon* process. Winlogon passes the credentials obtained from the dialog box to the *Local Security Authority (LSA)* service, as illustrated in the left part of figure 1.2. Apart from using a password, users may alternatively present their credentials by inserting a smart card or interacting with a biometric device.

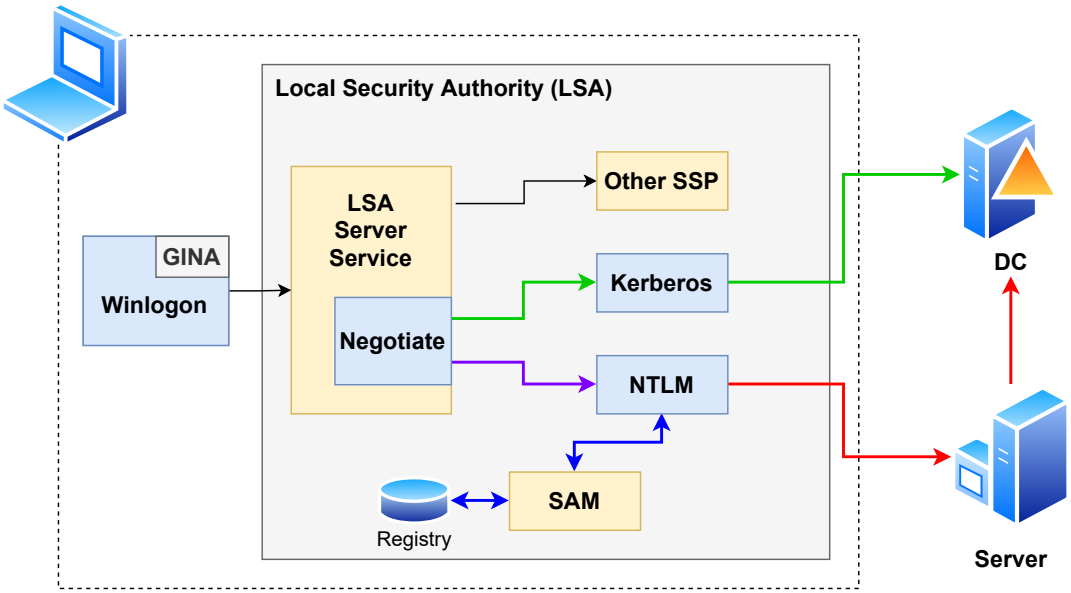

Figure 1.2: Simplified AD authentication overview

LSA subsystem may communicate with a remote authentication source (such as a DC). This happens through a protocol layer, in which access to different authentication protocols is provided via *Security Support Provider (SSP)* interface. The Windows OS implements a default set of authentication SSPs, including *Negotiate*, *Kerberos*, *NTLM*, *Secure channel*, and *Digest*.

Interactive logon can be initiated using either *local* or *domain* user account. Local user accounts are managed by *Security Accounts Manager (SAM)* and stored in *Registry*

database on the local computer. Local accounts are the default type of accounts on a Windows computer that has not joined an AD domain. These accounts allow users to access local resources; however, they are not sufficient for accessing and using domain resources.

Domain user accounts are stored in the AD database on DCs. A domain logon grants the user access to both local and domain resources. For a successful domain logon, it is required that both user and computer have an account in AD, and the computer is physically connected to the network. Kerberos or NTLM protocol is used to authenticate domain accounts.

Kerberos protocol provides greater security than NTLM, and therefore it is the preferred protocol to use within AD domain. Nevertheless, NTLM is still supported. Kerberos and NTLM SSPs are not to be used directly but via Negotiate security package instead that automatically selects between those two. Kerberos is selected by default unless it cannot be used by one of the systems involved in the authentication process. [5]

After interactive logon has taken place, *network logon* is used to confirm the user's identification to the network service the user is attempting to access. This is usually invisible to the user, as previously established credentials are reused. This way, integrated SSO functionality is provided with supported applications.

Figure 1.2 shows a simplified overview of the described authentication concepts. Different logon scenarios are illustrated: a) local account (blue path); b) domain account with NTLM (red path) and Kerberos (green path) protocols.

1.2.1 NTLM

NT LAN Manager (NTLM) authentication protocol is a family of protocols developed by Microsoft for use in Windows environments. These protocols are designed to provide authentication between clients and servers based on a challenge/response mechanism. NTLM has evolved throughout the history of Windows OS, and its current version, NTLMv2 has been used since Windows 2000. [6, 7]

Although NTLM authentication is replaced by Kerberos as the preferred authentication protocol, it is still supported and must be used for authentication with stand-alone systems, or systems configured as members of a workgroup [6]. Furthermore, NTLM authentication may also be used in scenarios when Kerberos authentication is not possible, such as if:

- one of the parties in authentication is not Kerberos-capable,
- the server has not joined an AD domain,
- Kerberos authentication is not configured properly,
- the implementation directly chooses to use NTLM.

In NTLM authentication, a resource server being accessed must take one of the following actions to verify the identity of a computer or user accessing it, depending on the authentication scenario:

1. Contact a domain authentication service on the DC if the authenticating account is a domain account.

2. Look up the account in the local account database, in case of a local account.

As the account information for domain accounts is maintained by the DC, only the DC can validate user credentials and complete the authentication sequence. The resource server uses Netlogon Remote Protocol to communicate with the DC for this purpose, which is also called *NTLM pass-through authentication*. [8]

NTLM authentication can be utilized during both interactive and network logon processes. Following is the description of a typical authentication sequence, based on Microsoft documentation [8, 9]. It is assumed that a domain user accesses a service on a resource server. The explanation is supported by the diagram in figure 1.3.

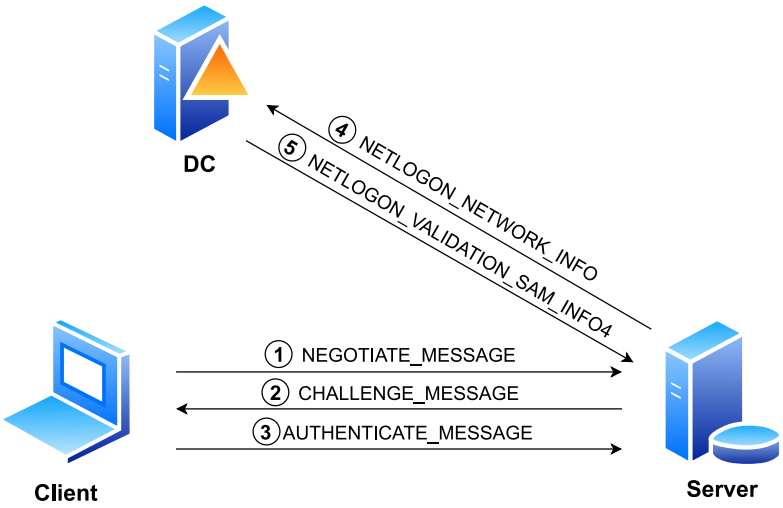

Figure 1.3: NTLM authentication

1. The user logs on to the client workstation by typing in the user name and password. The client computes an NTLM hash of the password and discards the actual password. To initiate the authentication, the client sends *NEGOTIATE_MESSAGE* to the server. Apart from NTLM options, this message includes the client's workstation name and the domain name. Based on the provided domain name, the server determines whether the client is eligible for local or domain authentication.

2. The server generates a random number (*nonce*) and sends it to the client in *CHALLENGE_MESSAGE*.

3. The client encrypts the challenge with the NTLM password hash and sends it in *AUTHENTICATE_MESSAGE* to the server. This message also includes the username of the authenticating account and the client's workstation name.

4. The server forwards the received response to the DC, including the challenge previously sent to the client, as *NETLOGON_NETWORK_INFO* message.

5. The DC uses the username to retrieve the hash of the user's password from AD database. It uses this hash to encrypt the challenge and compares it with the client's response. The result is returned in *NETLOGON_VALIDATION_SAM_INFO4* message to the server. If the verification is successful, the message contains the user's *Privilege Account Certificate (PAC)* with the authorization data. The server is then able to make authorization decisions.

1.2.2 Kerberos

Kerberos is a protocol that allows secure mutual authentication of principals communicating over an untrusted network. Kerberos protocol was initially developed at MIT for Project Athena and originally based on Needham-Schroeder's authentication protocol with modifications suggested by Denning and Sacco. One of the main advantages of the Kerberos protocol is that it enables SSO functionality. Today's version Kerberos v5 is specified in *RFC 4120*, replacing former *RFC 1510*. [10, 11]

Microsoft included Kerberos v5 in Windows 2000 (based on *RFC 1510*) with the aim to replace NTLM authentication in AD domains. In 2006, the protocol was updated to comply with *RFC 4120*. Microsoft's implementation of Kerberos introduces some differences and additional functionality beyond the *RFC* specification, including authorization, different implementation of SSP interface, or optional PAC validation. [10, 12]

Following paragraphs, based on *RFC 4120* [11] and Microsoft documentation [13], aim to provide a simplified explanation of the Kerberos protocol, focusing on its utilization for authentication in AD. The explanation is supported by the authentication diagram provided in figure 1.4.

The Kerberos protocol consists of three sub-protocols (or exchanges):

- *Authentication Service (AS)* exchange,
- *Ticket-Granting Service (TGS)* exchange,
- *Client/Server* exchange.

In Microsoft's implementation, the *Key Distribution Center (KDC)* is implemented as a domain service installed on the domain controller, which uses the AD as its account database, and performs two service functions: the Authentication Service and the Ticket-Granting Service. The AS and TGS exchanges therefore occur between a client and the DC, as shown in figure 1.4.

The AS exchange typically occurs at the initiation of a login session, such as when LSA service validates the user's domain credentials during interactive logon. Apart from authenticating itself, the client needs to obtain credentials for the TGS, which will subsequently be used in network logon when the client requests access to other servers.

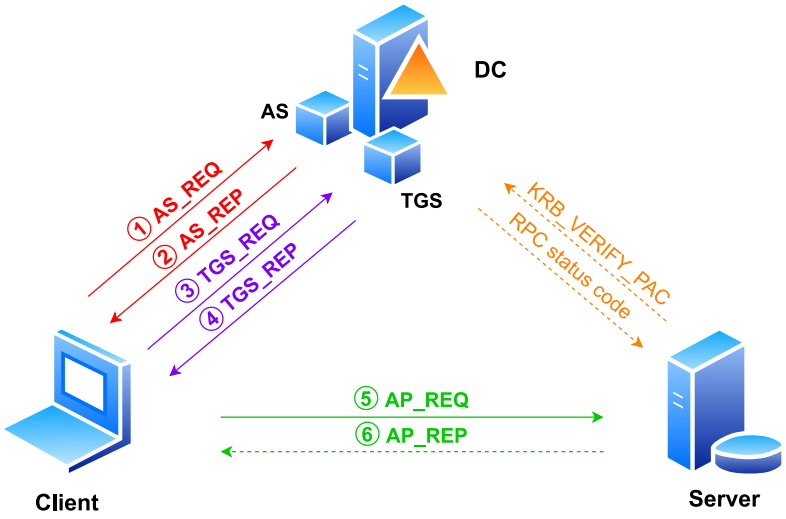

Figure 1.4: Kerberos authentication

1. Client sends *AS_REQ* message to the AS on KDC. This message contains an identification of the client, TGS, and pre-authentication data, which is a timestamp encrypted with the master key derived from the user's logon password.

2. When the KDC receives the *AS_REQ* message, it looks up the user in its account database (AD), decrypts the pre-authentication data, and evaluates the timestamp. If the timestamp is valid, the KDC can be assured that the client is genuine. KDC then creates a logon session key in two copies, one encrypted with the user's master key, and the other embedded into a *Ticket-Granting Ticket (TGT)* together with authorization data. TGT itself is encrypted with the KDC 's own master key, derived from its security principal – *KRBTGT* account. The created credentials are sent to the client in *AS_REP* message. The client extracts the TGT and the session key and stores them in its cache.

After AS exchange, the client holds a TGT that it may use to request a ticket from KDC in order to access a specific service. This happens in TGS exchange. TGT is reusable, and by default expires after 10 hours, after which it can be renewed. TGT contains a PAC which carries information about all the security groups in which the user is a member. TGT is encrypted and signed by *KRBTGT* account, so only KDC is able to decrypt it and read the data. [14]

3. Client sends *TGS_REQ* message to KDC. This message includes the identity of the requested service, an authenticator message encrypted with the user's logon session key, and the TGT.

4. KDC decrypts the TGT with its secret key and extracts the user's logon session key. The extracted session key is used to decrypt the authenticator message. If the

message passes the evaluation, the KDC invents a session key for the user to share with the target service. KDC encrypts one copy of the service session key with the user's logon session key. Another copy is embedded into a *Service-Granting Ticket (SGT)*, together with the user's authorization data, and encrypted with the service's master key. KDC replies to the client with *TGS_REP* message. The client extracts SGT from this message, decrypts the service session key with its logon session key, and stores these credentials in its cache.

If TGS exchange was successful, the client has a SGT which it presents for admission to a service in the Client/Server exchange.

5. Client sends *AP_REQ* message to the server hosting the target service. The message contains an authenticator that is encrypted with the service session key obtained from KDC, the SGT for use with the service, and a flag that indicates whether the client requests mutual authentication (optional).

6. Server decrypts the ticket, extracts the user's group membership information, the session key, and uses it to evaluate the authenticator. At this point, the server has enough information to make an authorization decision. If mutual authentication was requested, the server uses the session key to encrypt the time from the user's authenticator message and returns the result in *AP_REP* message.

During the Client/Server exchange, the server may optionally forward the PAC parsed from SGT to a DC in *KERB_VERIFY_PAC* message to validate that the user's group membership presented is accurate. The DC verifies the signature and returns the result to the server as a Remote Procedure Call (RPC) status code. After that, *AP_REP* message is sent to the client.

In the *TGS_REQ* message (step 3 above), the client requests access to a specific service instance. In Active Directory, service instances are uniquely identified by *Service Principal Names (SPNs)*. Before a service instance can be used with Kerberos authentication, its SPN must be registered on the account object that the service instance uses to log on. A given SPN can be registered with one account only. [15]

An SPN has the following format: `<svc_class>/<host>[:<port>[/<svc_name>]]`. The first two components are required. `<svc_class>` is a string that identifies the general class of service. There are well-known service class names [16], but the name can also be an arbitrary string unique to a service type. `<host>` is the name of the computer on which the service is running, typically identified by Fully Qualified Domain Name (FQDN). Port number may be used to differentiate between multiple instances of the same service class on a single computer. Finally, `<svc_name>` is used for replicable services. Examples of SPNs for common services might be `MSSQLSvc/db01.example.com:1433` or `TERMSERV/server1.example.com`. [15]

1.3 Security Monitoring of Active Directory

Active Directory, as a standard part of Windows Server OS, integrates into its native logging and auditing solution called *Windows Event Log*. Event Log is a collection of events, whereas an *event* represents a basic log or audit trail unit. The event logging service records software, hardware, and security events from the OS and applications. Event Log can be viewed via *Event Viewer* management console snap-in, command-line tools, or its application programming interface (API). [17]

Logging in an AD domain is decentralized by default, as every machine keeps its own Event Log. Security monitoring of AD environment requires collecting events from machines across the domain into a centralized solution. For this purpose, organizations typically utilize Security Information and Event Management (SIEM) software. SIEM products comprise multiple log management phases, including log collection, parsing, and correlation of events from various log sources. These products also commonly include features for security monitoring, such as alerting or incident management. [18]

Windows Event Log contains two main categories: *Windows Logs* and *Applications and Services Logs*. Windows Logs category is intended for events that apply to the entire system, while the latter category is suited for events from a single application or component. Windows Logs category is further structured into five channels: *Application*, *Security*, *Setup*, *System*, and *ForwardedEvents*. [19]

Security auditing settings for an AD domain can be controlled centrally by security audit policies. Windows defines two kinds of auditing policies: *Basic* and *Advanced security audit policies*.

Basic audit policy settings specify nine basic auditing categories. These were introduced in Windows 2000 and have been available in all versions of Windows released since. Advanced security audit policies were introduced in Windows Vista / Server 2008. Although both offer similar auditing subcategories, these two policies are not compatible and are applied differently. Advanced security audit policies are preferred, as they offer more granular control over the number and types of events audited and can be applied by using Group Policy. [20]

Advanced audit policy settings are split into categories and subcategories that allow monitoring only specific behaviors. Each of the categories typically generates one or more event types in Security Event Log and can be configured to audit success, failure, or both. Audit policy settings are defined for the following categories:

- Account Logon,
- Account Management,
- Detailed Tracking,
- DS Access,
- Logon/Logoff,
- Object Access,

- Policy Change,
- Privilege Use,
- System,
- Global Object Access Auditing.

Several categories provided by the Advanced security audit policies are tied closely to Active Directory. For instance, the categories Account Logon and Logon/Logoff track authentication and credential usage, Account Management monitors modifications to accounts and groups, and DS Access records changes and replication of the AD schema. Other categories, such as Detailed Tracking, Object Access, and Privilege Use, provide information about interactions with sensitive resources. [20]

Apart from the Security channel, other channels in Windows Event Log may also contain events relevant for security monitoring of AD. Relevant data may be recorded by applications that provide their events in *Applications and Services Logs*. A typical example is PowerShell, a scripting language and tool for automation and OS management. As a built-in feature of Windows OS, PowerShell can be easily utilized by adversaries to perform malicious actions. Hence, recent versions of PowerShell provide auditing features, such as *Module Logging* and *Script Block Logging*, that record execution activity into *Microsoft-Windows-PowerShell/Operational* Event Log channel. [21]

Windows Event Log files are stored in a proprietary binary *EVTX* format. This format can be rendered into a textual form, such as if displayed via Event Viewer, or translated into XML format, which is more suitable for its further parsing and processing by SIEM solutions. An example of event 4769 displayed via Event Viewer is shown in figure 1.5.

Events conform to a predefined *Event Schema* [22] that describes possible elements and their data types. The main elements of an event are `SystemProperties` and `EventData`. The logged information is contained in the leaf nodes of these elements, which can be understood as key-value pairs representing fields and their values.

`SystemProperties` node is presented consistently across different events, containing fields such as:

- `EventID`, a number identifying the particular event type,
- `Channel`, the source Windows Event Log channel,
- `TimeCreated`, the timestamp that determines when the event was logged,
- `Computer`, the name of the computer on which the event occurred.

`EventData` node contains data attributes specific for a particular event, like `IpAddress`, `TargetUserName`, `SubjectDomainName`, or `ServiceSid`. The set of attributes is different for various event IDs.

In general, event attributes have various data types, as they contain different kinds of information. Examples of value types commonly occurring in logged events are:

- **strings**:
  - names: `TargetUserName`, `SubjectDomainName`, `Computer`, `ProcessName`, ...
  - categories: `Channel`, `ObjectType`, ...
  - text: `CommandLine`, `Path`, `ScriptBlockText`, ...
- **numerics**:
  - identifiers: `EventID`, `LogonGuid`, `ProcessId`, `SubjectUserSid`, ...
  - codes: `LogonType`, `Status`, `TicketEncryptionType`, ...
  - other: `IpAddress`, `IpPort`, `KeyLength`, `RecordNumber`, ...

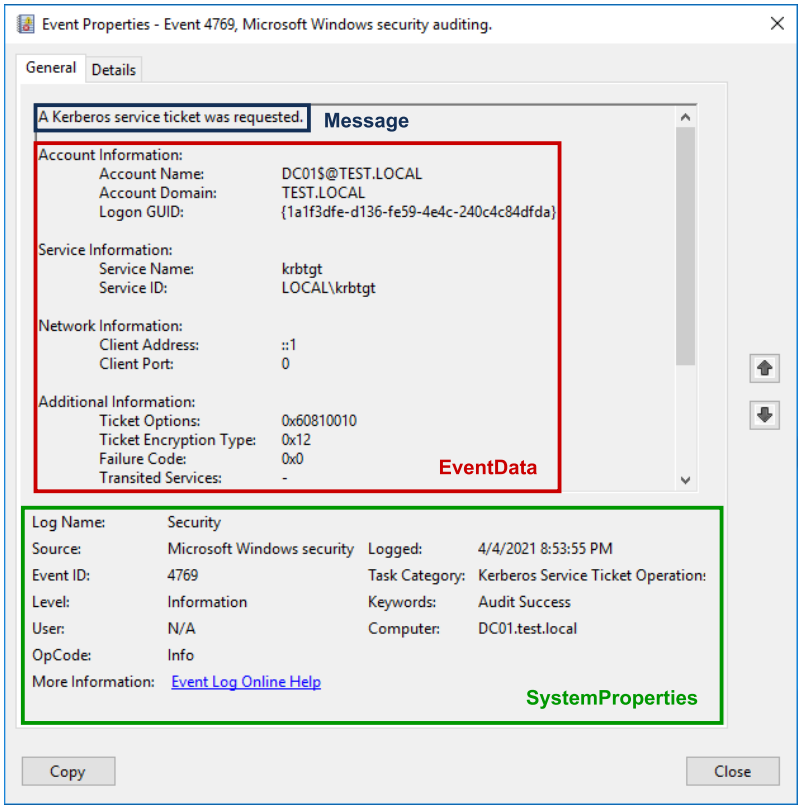

Figure 1.5: Event 4769 displayed via Event Viewer

Understanding the data types of the event attributes is essential for their representation as features in machine learning models. As visible from above, event properties commonly contain names, identifiers, or text fields. However, machine learning algorithms typically work with numerical features. Therefore, before the event data can be used with these algorithms, its preprocessing and feature extraction is necessary.

1.4 Active Directory Threats

To effectively defend against cyber security attacks in general, it is necessary to "know the enemy" – understand adversaries' goals and motivation, comprehend how attackers operate, and identify the methods and tools they use. Defenders need to know what to protect against before they can develop any countermeasures.

The complexity of this area justifies the use of different classifications and methodologies. Lots of analytic models and frameworks exist, while each of them has a slightly different goal, use cases, and focuses on a different perspective. However, various models are not necessarily incompatible, and in fact, many of them are complementary. The most widely used models for security monitoring include:

- Diamond Model of Intrusion Analysis [23],
- Cyber Kill Chain [24],
- MITRE ATT&CK Framework [25].

Since this thesis focuses on threats in the context of a particular technological environment – Active Directory, *MITRE ATT&CK Framework* has been chosen as the most appropriate baseline. ATT&CK provides a detailed description of adversary techniques, to the level of particular tools or procedures, and includes indicators that can be leveraged for their detection.

ATT&CK is a framework describing adversarial actions across their lifecycle, provided as a globally accessible knowledge base by The MITRE Corporation. The framework has two parts: *ATT&CK for Mobile*, focusing on mobile devices, and *ATT&CK for Enterprise*, covering enterprise environments. The latter part is relevant in the context of this thesis, as AD is an enterprise product. [25]

ATT&CK for Enterprise incorporates information about Tactics, Techniques, and Procedures (TTPs) adversaries use to compromise and operate within an enterprise network. The knowledge is organized in a matrix consisting of 14 *tactic* categories, while each category contains a list of *techniques* adversaries use to perform that tactic. Individual techniques provide technical descriptions, useful detection indicators, and potential mitigations. Some of the techniques are further split into *sub-techniques*. [26]

As of writing this thesis, the current version of ATT&CK for Enterprise (8.2) describes 178 techniques and 352 sub-techniques, covering different platforms that appear in enterprise environments. Some of the techniques apply to multiple platforms, but some are relevant to a specific platform (i.e., OS or application) only. The framework allows displaying techniques relevant to some major platforms; however, a specific matrix for Active Directory is not provided. Only matrices covering Windows OS and cloud Azure AD are available. [26]

However, not all Windows-related techniques apply to AD, and the matrix for Azure AD misses several crucial techniques relevant to on-premise AD environments. Therefore, the TTPs listed in ATT&CK Matrix for Enterprise have been reviewed to identify those applicable to Active Directory. As AD relates with a wide range of platforms, protocols,

and applications, in broader understanding, many different adversary techniques may be relevant to it. The selection focuses on adversarial techniques closely related to AD principal services – identity management and authentication.

The techniques and sub-techniques identified as relevant to AD are listed in table 1.1. Note that some of the techniques do not specify a sub-technique, and not all sub-techniques of a particular technique are necessarily relevant to AD. References to the appropriate tactics are mentioned, whereas a single technique may belong to multiple tactics.

Figure 1.6 illustrates the layout of the selected techniques across a simplified ATT&CK Matrix. The techniques relevant to AD are marked with red color. For brevity, tactics and techniques are displayed by their IDs only.

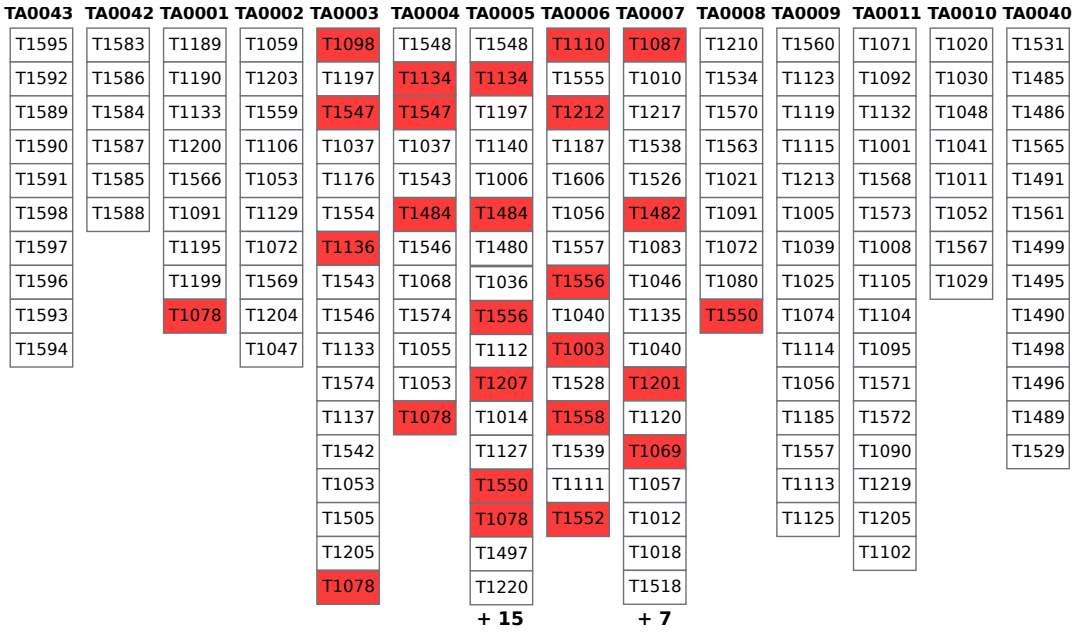

Figure 1.6: Techniques targeting AD visualized in ATT&CK Matrix

Noticeably, the selected techniques are concentrated in the middle part of the Matrix, consistently with the position of AD and its role in an enterprise network. As such, AD is an internal system, usually placed behind perimeter security protection, and thus plays a minor role in the initial attack phases, such as *Reconnaissance*, *Resource Development*, and *Initial Access*. Also, AD on its own is typically not the purpose and target of cyber attacks. It therefore relates to the latter tactics, *Collection*, *Command and Control*, *Exfiltration*, and *Impact*, only marginally.

Nonetheless, Active Directory may provide a path for compromising more interesting systems in the domain. As the central point for authentication and identity management, AD is significantly impacted by the tactics that adversaries leverage to move from the entry point of the environment to their goal objectives and target systems. In a simplified

Technique	Sub-technique	Tactic
Valid Accounts	Domain Accounts	Initial Access, Persistence, Privilege Escalation, Defense Evasion
Create Account	Domain Account	Persistence
Account Manipulation	-	Persistence
Boot or Logon Autostart Execution	Security Support Provider	Persistence, Privilege Escalation
Domain Policy Modification	Group Policy Modification	Privilege Escalation, Defense Evasion
	Domain Trust Modification	
Access Token Manipulation	SID-History Injection	Defense Evasion, Privilege Escalation
Rogue Domain Controller	-	Defense Evasion
Modify Authentication Process	Domain Controller Authentication	Defense Evasion, Credential Access
Brute Force	Password Guessing	Credential Access
	Password Cracking	
	Password Spraying	
	Credential Stuffing	
OS Credential Dumping	LSASS Memory	Credential Access
	Security Account Manager	
	NTDS	
	LSA Secrets	
	Cached Domain Credentials	
	DCSync	
Steal or Forge Kerberos Tickets	Golden Ticket	Credential Access
	Silver Ticket	
	Kerberoasting	
	AS-REP Roasting	
Unsecured Credentials	Group Policy Preferences	Credential Access
Account Discovery	Domain Account	Discovery
Password Policy Discovery	-	Discovery
Permission Groups Discovery	Domain Groups	Discovery
Domain Trust Discovery	-	Discovery
Use Alternate Authentication Material	Pass the Hash	Defense Evasion, Lateral Movement
	Pass the Ticket	

Table 1.1: ATT&CK tactics and techniques related to AD

view, centralized management of accounts and domain policies may allow for *Persistence*, *Privilege Escalation*, *Defense Evasion*, or *Discovery*, whereas the role of AD in authentication poses a risk of *Credential Access* and *Lateral Movement*.

The following subsections briefly describe the relevant techniques across adversary tactics, with emphasis on possibilities for their security monitoring and detection. The description is based on information provided by the ATT&CK Framework [27, 28].

1.4.1 Persistence

Persistence consists of techniques that adversaries use to keep access to the environment. In an AD domain, adversaries may retain their access by controlling domain accounts. These accounts may hold different privileges. In case adversaries have a sufficient level of access to AD, they may be able to manipulate existing accounts, such as modify their credentials, permissions, group membership, or even create new domain accounts.

Activities related to modifications of existing accounts or account creation are auditable via Windows events to a fair extent. Nonetheless, as valid domain accounts can be utilized in various tactics, robust monitoring of the related techniques should be in place.

1.4.2 Privilege Escalation

In Privilege Escalation, the adversary goal is to gain higher-level permissions to systems. In an AD environment, the ultimate goal is to obtain rights equivalent to *Domain Admins* or *Enterprise Admins*, which essentially mean administrator access to almost every system in the domain. There are many Privilege Escalation methods, while the most common take advantage of system weaknesses, misconfigurations, and vulnerabilities. However, there are also several techniques related to AD architecture.

In section 1.2, the SSP interface was described as a part of authentication process in a Windows OS. Adversaries may abuse SSPs to load a malicious dynamic link library (DLL) into LSA process on system boot. This DLL may then have access to encrypted and plaintext passwords that are stored in OS, such as those of domain users logged on. This is mitigated in newer versions of Windows, as the OS requires these DLLs to be signed by Microsoft. Furthermore, changes in Registry related to SSP can be audited.

Since domain configuration settings are principally tied to the AD environment, their alternation may have a significant impact. Related techniques include modifications of Group Policy Objects (GPOs) or changing trust settings for domains. Adversaries may also modify the settings, carry out malicious actions, and then revert the change to evade detection. Modification of domain policy can be tracked via relevant Windows events.

Security principals in AD are identified by a unique SID. These values are used in security descriptors and access tokens. An account can hold additional SIDs in the `SID-History` AD attribute, which allows account migration between domains. Adversaries having *Domain Admins* or equivalent rights may insert harvested or well-known SID values into this attribute and gain elevated access to domain resources, or bypass access controls. Therefore, any changes to the `SID-History` attribute of users should be monitored by account management events on DCs.

1.4.3 Defense Evasion

Adversaries naturally attempt to hide their actions and try to avoid being detected. Besides deliberately taking measures for subverting defenses, Defense Evasion may also be achieved as a part of other techniques.

Two interesting evasion techniques exist for AD that include patching and registering a rogue domain controller. Malware may be used to inject false credentials into the authentication process on a DC, and these credentials may be subsequently used to authenticate as a domain user. Alternatively, adversaries may use a tool capable of simulating a real DC, register it as a DC in Active Directory, and then inject and replicate changes for domain objects. By using this technique, adversaries may bypass system logging, as the rogue DC is not audited.

Apart from malware protection, monitoring of functions exported from authentication-related DLLs and interactions with LSA subsystem may be implemented. Possibilities of monitoring for a rogue DC include analyzing network traffic associated with AD replication, as well as monitoring related Windows events.

1.4.4 Credential Access

The objective of Credential Access is to gain knowledge of legitimate credentials, such as account names and passwords, which can be further used to access systems. AD, as the domain authentication foundation and account database, is susceptible to adversary techniques that attempt to guess, steal, forge, or dump credentials.

Brute Force techniques are typically used to guess unknown passwords systematically. An adversary may interact directly with the authentication service or perform an offline attack against previously acquired credential data, such as password hashes.

In the simplest form, an attacker would guess the password for an account by attempting common or generated passwords against it. This attack involves the risk of locking out accounts if account policies of the target AD environment are not considered. To avoid the lockouts, an adversary may use the *Password Spraying* technique, where a single or small list of commonly used passwords is attempted against many different accounts. Apart from common passwords, attackers may attempt to use credentials obtained from online breach dumps. The chance of credential overlap is not negligible, as users tend to use the same credentials for multiple services and platforms.

In case adversaries could obtain password hashes, they may try to recover plaintext passwords used to compute the hashes. This can be done either systematically or by using pre-computed tables. The advantage of this technique is that the cracking can be done offline, avoiding any interaction with the authentication service.

Detection of Brute Force techniques in an AD environment can be based on monitoring events that audit authentication attempts, as well as account lockouts. The principal sign of a brute force attack is a high number of logon failures. Cracking of password hashes is difficult to detect since it is generally done outside of the AD environment.

Another method of obtaining credentials is dumping them from legitimate storage places in the Windows OS:

- memory of the LSA Subsystem Service process,
- Security Accounts Manager database,
- *NTDS.dit*, the AD domain database file located on DCs,
- *LSA Secrets* storage in Registry.

The techniques of *Credential Dumping* typically require elevated privileges on *Administrator* or *SYSTEM* level. Besides interacting with credential storage directly, adversaries may use tools capable of abusing DC's API to simulate the replication process and pull credential data from a remote DC.

Possibilities of detecting Credential Dumping depend on the particular procedure and tools used by adversaries. In general, monitoring of suspicious program execution, processes, and their command-line arguments may be implemented. Also, suspicious interactions with sensitive objects, such as LSA subsystem and DC replication requests, should be monitored. Auditing of these actions may be possible via Security and PowerShell Event Log channels. The number of events can be narrowed by monitoring signatures of common credential dumping tools.

Broad usage of Kerberos protocol in AD authentication has led to the development of specific adversary techniques for stealing or forging Kerberos tickets. Stolen tickets may be prone to offline *Password Cracking* attacks, whereas an adversary capable of forging tickets may subvert the authentication process and gain access to protected resources.

In the *Kerberoasting* attack, adversaries may use a valid TGT to obtain a SGT for a specific service. The obtained SGT may be encrypted with a weak cipher suite and therefore be vulnerable to a Password Cracking attack. Adversaries may export the ticket and crack it offline, which would reveal the service account's password in question.

In a similar technique, called *AS-REP Roasting*, an adversary may obtain credentials of accounts with disabled Kerberos pre-authentication. In that case, TGT data obtained from a DC may be encrypted with an insecure algorithm, and the ticket prone to Password Cracking.

Detection of these techniques may be based upon looking for irregular patterns of activity, especially numerous requests, using weak encryption within a small time frame. Kerberos pre-authentication is enabled by default and should be disabled only if inevitable.

Adversaries who have obtained the password hash of a service account may create a forged Kerberos SGT, called *Silver Ticket*. This ticket is used to authenticate to the specific service, allowing access unless optional PAC validation occurs. Silver Ticket is presented directly to the server, there is no communication with the DC, and therefore the options for its detection are limited.

In the *Golden Ticket* technique, adversaries who have obtained password hash of *KRBTGT* account may forge Kerberos TGTs for any account in AD. A forged TGT allows requesting service tickets from TGS and accessing domain resources. However, dumping *KRBTGT* password hash requires privileged access to a DC.

Forged Kerberos tickets cannot be easily recognized from the logged authentication events. The events may be monitored for anomalies, such as malformed or blank fields,

occurrences of TGS requests without preceding TGTs issued, or the lifetime of TGT tickets differing from the domain setting. It also makes sense to monitor the activities in conjunction with prior Credential Dumping techniques.

1.4.5 Discovery

In Discovery tactics, adversaries aim to gain knowledge about systems and the network. Active Directory is indeed a helpful tool for this tactic, as it holds valuable information about the domain. Techniques specifically related to data stored in AD include discovering accounts and groups together with their permissions, password policies, or domain trusts. The obtained information may be especially helpful in applying subsequent techniques and targeting the attacks.

Detecting of Discovery activities may be challenging. Obtaining useful information is often possible using the native commands and utilities designed for managing AD. Discovery activities should be monitored together with other malicious actions that may arise based on the obtained information. Apart from that, signature-based monitoring of processes and command-line arguments, as well as PowerShell *Module Logging* and *Script-Block Logging*, may be utilized.

1.4.6 Lateral Movement

After Discovery, adversaries may systematically move across the environment, pivoting through the identified systems and accounts until they reach their target objective. One of the options applicable to an AD environment is using alternate authentication material to bypass access controls. Authentication processes in AD generate credential material, i.e., password hashes and Kerberos tickets, which may be stolen from the OS through Credential Access techniques. Adversaries may use this material to authenticate with other systems without knowing the actual credentials.

In the *Pass the Hash* technique, an adversary that has obtained a password hash of an account may participate in NTLM authentication and access remote systems without knowing the account's plaintext password. In the *Pass the Ticket* technique, an adversary obtains Kerberos tickets – TGT or SGT. The obtained SGT allows access to a particular resource, whereas a TGT may be used in TGS exchange to request access to any resource the user has privileges to access.

Detection of the techniques mentioned above may be challenging, as there are no specific indicators or events for the mentioned activities. The actions could be correlated with other suspicious activities that previously occurred on the accessed systems, as well as priorly executed Credential Access techniques. Another possibility is to review events related to logon and credential use for discrepancies, such as logon of privileged accounts from unusual machines.

Chapter 2

Selected Attack Techniques

Section 1.4 outlined multitude of adversary techniques targeting Active Directory. In my bachelor's thesis [29], I analyzed several of these techniques and developed a set of rules for their detection. The detection rules were analyzing Windows Event Log data collected from a virtual lab environment. The following techniques, especially of Credential Access and Lateral Movement tactics, were covered:

- Brute Force,
- OS Credential Dumping,
- Steal or Forge Kerberos Tickets,
- Use Alternate Authentication Material.

Detection rules define conditions based on known indicators of the attack techniques. These conditions are checked against relevant Windows events, and if matched, an alert is generated. This method is known as misuse detection, or signature-based approach [30].

However, in practice, signature-based detection may not always be sufficient. Since the rules are defined statically, they often contain various manually defined thresholds, constants, or signatures. These are difficult to determine in the creation process, but on the other hand, relatively easy for an attacker to overcome. Adversaries may modify their attack procedure to evade detection by limiting the number of attempts to fit under a numeric threshold or renaming the attack tools to avoid signatures.

Based on the outputs of the previous work and research of existing detection approaches, I have identified and selected two adversary techniques for which signature-based detection is not sufficient: Password Spraying and Kerberoasting. These techniques are traditionally detected using threshold alerting based on the number of attempts. Besides the risk of evading the detection by adversaries, the detection is particularly susceptible to producing lots of false alarms, as the logged events are similar to those generated by users during regular activities.

Both these attack techniques fall under Credential Access tactics, and their goal is to discover credentials of valid domain accounts. Specifics of their methodology make these

techniques different and more effective than the typical Brute Force attacks. Unlike many other Credential Access techniques, they do not require elevated privileges and hence can be executed earlier in the chain of AD compromise.

This chapter provides a detailed description of the selected attack techniques, followed by thorough research on existing methods for their detection. Different aspects and limitations of these approaches are discussed, justifying the motivation for developing a novel detection solution based on machine learning techniques.

2.1 Password Spraying

Password Spraying is one of the most effective Brute Force techniques. It can be thought of as an advanced password guessing attack, where an attacker flips the conventional strategy. Instead of attempting to log on to a single user account by trying many passwords, the attacker attempts a single password on many different accounts. This is also often referred to as horizontal vs. vertical approach. In the horizontal approach of Password Spraying, attackers avoid locking accounts, which is the basic prevention mechanism of password guessing attacks. [31]

Password Spraying attack is conceptually applicable to many different technologies. However, Active Directory is a valuable target due to its SSO functionality. By compromising a domain account, an attacker may obtain access to multiple systems, network resources, and applications. In Mitre ATT&CK Matrix, the Valid Accounts technique appears under four different tactics. Gaining a valid AD domain account may open the path for an attacker to apply these tactics. [32]

Password failures are auditable through events recording logon activity. Detection of account lockouts is not applicable, as adversaries deliberately limit the number of attempts. Nevertheless, the monitoring logic can be flipped similarly as the logic of the attack: instead of detecting numerous logon failures for one target account, focusing on failed attempts for many target accounts. [31]

However, the detection becomes problematic if adversaries limit the rate of the attempts. A few failed logins per user or source can be easily lost in the noise of standard logon patterns in the AD domain. Such an attack looks like an isolated failed login from the perspective of logged events.

2.1.1 Attack Description

Password guessing attacks, and credential brute force techniques in general, tend to work because users create weak passwords. Besides the notorious choices, like “123456”, people would use names of their relatives, birth dates, favorite sports teams, or a company name as their password [33].

Organizations attempt to solve this problem by introducing password policies that enforce the complexity of the passwords. In Windows AD environments, there are *Password Policy* settings that can be enforced via Group Policy [34]. The options available include the following:

- Minimum password length,
- Password must meet complexity requirements,
- Minimum/Maximum password age,
- Enforce password history.

By combining the above options, administrators may enforce a password policy that would allow only passwords of a sufficient length, require usage of multiple character sets (letters, digits, special characters, ...), require users to change their password regularly, and prevent password reuse. However, the effect of such policies on the end users may be the opposite. Users tired of inventing new unique passwords every few months would create a pattern that is easy to remember (e.g., "Spring2021!", "Summer2021!", ...), circumventing the purpose of the policy.

In a Password Spraying attack, it may be enough for an attacker to select one common pattern and apply it to all users in AD. The probability that there will be at least one person using it among hundreds or thousands of users in an organization is not negligible. The chance for success is higher than attempting an extensive list of passwords on a single randomly selected user.

Before adversaries can start guessing passwords, they must be aware of user account names available in the target environment. If they do not possess this information already, it can be often acquired using open source intelligence resources. A simple web search may be enough, as people may inadvertently leak their account names in online posts, on social networks, or use the same username on multiple platforms. Usernames can also be commonly reconstructed from the naming schema for emails, such as `first.last@company.org`. Adversaries already having access to an internal system connected to AD may simply query the DC to obtain a list of available accounts. This can be done for example by invoking `Get-ADUser` cmdlet in PowerShell [35].

Having access to an internal system connected to AD is not a strict prerequisite for executing the Password Spraying attack. Organizations commonly use AD authentication with different applications through AD FS that enable users to access the applications even when they are not on a corporate network. Adversaries can take advantage of the federated endpoints available on the Internet and use these endpoints to carry out Password Spraying. [36]

Since manually typing passwords into authentication dialog boxes is not particularly efficient, adversaries automate this activity using scripts. These may be written in PowerShell, such as *DomainPasswordSpray* [37], or other languages. An advantage of using PowerShell is its default availability in Windows OS, refraining from the necessity to install additional dependencies. Nevertheless, plenty of other tools capable of Password Spraying are available. Some of them further automate the attack by automatically enumerating available user accounts. Example execution of a successful Password Spraying attack is demonstrated in figure 2.1.

During an attack, adversaries may attempt multiple passwords for one account instead of a single one. The only constraint is to limit the number of attempts per account, so

```
>> Invoke-DomainPasswordSpray -UserList .\users.txt -Password "Passw0rd!"
[*] Using .\users.txt as userlist to spray with
[*] Warning: Users will not be checked for lockout threshold.
[*] The domain password policy observation window is set to 5 minutes.
[*] Setting a 5 minute wait in between sprays.

Confirm Password Spray
Are you sure you want to perform a password spray against 10 accounts?
[Y] Yes  [N] No  [?] Help (default is "Y"): Y
[*] Password spraying has begun with  1  passwords
[*] This might take a while depending on the total number of users
[*] Now trying password Passw0rd! against 10 users. Current time is 5:50 PM
[*] Writing successes to
[*] SUCCESS! User:john.doe Password:Passw0rd!
[*] Password spraying is complete
```

Figure 2.1: Successful Password Spraying of a user account

the account is not locked. In AD, the relevant options are controlled via *Account Lockout Policy* [34]. This policy allows the following settings:

- Account lockout threshold,
- Account lockout duration,
- Reset account lockout after.

Account lockout threshold determines the number of failed login attempts that will cause a user account to be locked. After that, the account must be manually unlocked by an administrator or can be unlocked automatically after some time elapsed. This period can be specified by *Account lockout duration* setting. The third option, *Reset account lockout after* controls the number of minutes that must elapse from the time a user fails to log on before the failed logon counter is zeroed. [34]

Configuring these policy settings involves some trade-offs, and the values must be considered individually for every AD environment. Keeping the settings too loose may open space for effective Brute Force attacks, while setting them too strict poses the risk of accidental user lockouts, as well as Denial of Service (DoS) attacks. [34]

The implications for a Password Spraying attack are that an adversary aware of the *Account Lockout Policy* in the target AD environment may adjust the attack to maximize its efficiency without the risk of locking out accounts. Supposing an attacker already has access to the domain, both *Password Policy* and *Account Lockout Policy* configuration may be obtained using PowerShell, as illustrated in figure 2.2. An adversary may then change the attack strategy accordingly. Moreover, some advanced attack tools, such as *DomainPasswordSpray* [37], are even capable of obtaining this information and adapting the attack automatically. [35]

```
PS C:\Users\Administrator> Get-ADDefaultDomainPasswordPolicy


ComplexityEnabled           : True
DistinguishedName           : DC=test,DC=local
LockoutDuration             : 00:05:00
LockoutObservationWindow    : 00:05:00
LockoutThreshold            : 10
MaxPasswordAge              : 42.00:00:00
MinPasswordAge              : 1.00:00:00
MinPasswordLength           : 7
objectClass                 : {domainDNS}
objectGuid                  : cb2f8b2f-5713-4867-b0ce-f72689256bf2
PasswordHistoryCount        : 0
ReversibleEncryptionEnabled : False
```

Figure 2.2: Account policy information obtained via PowerShell

2.1.2 Detection

To detect Password Spraying, it is first necessary to determine relevant audit events and identify possible indicators in them. Auditing of authentication and credential usage in AD is vastly covered by native events that generate into Windows Event Log. Based on the documentation of Advanced Audit Policy Configuration [34], the audit policy settings related to logon attempts are controlled by its following subcategories:

- Account Logon:
  - Audit Credential Validation,
  - Audit Kerberos Authentication Service,
  - Audit Other Account Logon Events.
- Logon/Logoff:
  - Audit Account Lockout,
  - Audit Logoff,
  - Audit Logon,
  - Audit Other Logon/Logoff Events,
  - Audit Special Logon.

In my previous work [29], I have already reviewed these options, together with possible events that are generated by the particular policy subcategories. For the Password Spraying scenario, the events of interest are mostly logon failures, as the attempted password would be wrong for many users. The events identified as relevant for Password Spraying detection are specified in table 2.1.

Microsoft documentation [34] describes the occurrence of the respective event IDs as follows:

Event ID	Message
4625	An account failed to log on.
4648	A logon was attempted using explicit credentials.
4771	Kerberos pre-authentication failed.
4776	The domain controller attempted to validate the credentials for an account.

Table 2.1: Events relevant for Password Spraying detection

Event 4625 generates on the computer where a logon attempt was made, which may be on domain controllers, member servers, and workstations.

Event 4648 generates when a process attempts an account logon by explicitly specifying that account's credentials.

Event 4771 generates for Kerberos authentication, every time the KDC fails to issue a TGT, which also occurs in case a wrong password was provided. This event generates only on domain controllers.

Event 4776 generates every time credential validation occurs using NTLM authentication. This event occurs only on the computer that is authoritative for the provided credentials, which for domain accounts is the domain controller.

All the events listed in table 2.1, except for event 4648, provide a field containing the status code of the reason why the logon attempt failed. Based on this code, the events allow filtering to only those indicating logon attempts with a misspelled or wrong password. List of these status codes is provided in table 2.2.

Event ID	Field	Value
4625	`SubStatus`	`0xC000006A`
4771	`Status`	`0x18`
4776	`Status`	`0xC000006A`

Table 2.2: Status codes indicating wrong password

As explained in section 1.2, the main protocols used for authentication in AD are Kerberos and NTLM. Besides the overall configuration of the AD environment and capabilities of the authenticating parties, the choice of the protocol depends also on the service accessed and the particular application attempting to authenticate. The distinction between usage of these protocols in various logon scenarios is important, as they result in different types of audited events. This must be reflected in the detection logic, as the particular method used in a Password Spraying attack eventually impacts the event IDs logged.

Sean Metcalf published research on this topic [35], and illustrated the difference in the logged events on an example by accessing a Server Message Block (SMB) share on

a DC vs. accessing a LDAP service. In the case of an SMB share, the authentication was performed using NTLM and events 4625 were logged. In the case of an LDAP service, Kerberos protocol took place, which resulted in only events 4771 being logged. Events 4648 were logged on the workstation where Password Spraying was performed.

Event 4625 defines the `AuthenticationPackage` field, with possible values including `NTLM`, `Kerberos`, or `Negotiate`, suggesting that the event should be logged invariantly of the authentication protocol used [38]. However, based on the above, this does not seem to be the case for failed Kerberos pre-authentication.

To further research auditing of different events in various logon scenarios, I conducted a series of experiments in a lab AD environment. This environment consisted of a DC, member server, and a client workstation, and had the appropriate auditing configured. Several different authentication actions were performed that resulted in a failed logon of a single domain account due to a bad password. The actions taken included execution of three different Password Spraying attack tools from the client workstation. Each of these tools was used for a different authentication objective:

- *Spray*, targeting a SMB share on a server using NTLM authentication [39],
- *DomainPasswordSpray*, using .NET `DirectoryEntry` class to authenticate towards a domain controller over NTLM [37],
- *Kerbrute*, using Kerberos pre-authentication to request a TGT from the DC [40].

Apart from executing the listed attack tools, two more casual actions were performed. An invalid password was typed into the *Log On to Windows* dialog box during an interactive logon of the client. Another action represented an unsuccessful attempt to mount a network share located on the member server via `net use` command. Table 2.3 describes the events logged for the performed actions.

Action	**Protocol**	**Events logged**			
		4625	4648	4771	4776
Log On to Windows dialog	Kerberos	Client	-	DC	-
SMB share via `net use`	Kerberos	-	-	DC	-
Spray	NTLM	Server	-	-	DC
DomainPasswordSpray	NTLM	DC	Client	-	DC
Kerbrute	Kerberos	-	-	DC	-

Table 2.3: Events logged for different bad password scenarios

Kerberos authentication with bad passwords manifests in invalid pre-authentication data supplied by the client in AS exchange, which results in event 4771 always being logged on a DC. During an interactive logon, associated failure event 4625 is logged on a client workstation. However, this does not seem to occur during network logon.

For NTLM authentication, the event 4776 is logged on a DC. This event occurs together with event 4625, which generates on the computer where the logon attempt was made.

For network logon, this may be either DC or the server. In the case of a service accessed on the server, NTLM pass-through authentication takes place, and the server is in charge of the authentication process. Event 4648 appears on the client workstation, but only in some cases.

To conclude the observations, detection of Password Spraying can be based on events 4771 for Kerberos authentication and either event 4776 or 4625 for NTLM authentication. Event 4776 offers the advantage of seeing all attempts on a DC, whereas 4625 can also be logged on member servers. However, event 4625 is richer in the provided details, as 4776 shows only name of the computer from which the authentication attempt was performed. Information about the destination computer and network details are not present. Also, event 4776 does not generate when a domain account logs on locally to a DC. [41]

Once the appropriate logging is configured, and relevant events are identified, these may be used to build detection rules. The most straightforward approach is to create threshold rules based on the number of detected events within a defined timeframe. For instance, Metcalf [35] proposes configuring alerts for the following:

- more than 50 events 4625 within 1 minute,
- more than 50 events 4771 within 1 minute,
- more than 100 events 4648 on workstations within 1 minute.

In my previous work, I have proposed two Splunk rules for detecting Password Spraying based on similar principles. The rules are based on DC events 4771 and 4776, for Kerberos (listing 2.1) and NTLM (listing 2.2) authentication respectively, using the appropriate status codes for bad passwords. These rules are configured to trigger an alert if there are more than five events detected and less than five minutes elapsed between the individual failures. [29]

```
source=XmlWinEventLog:Security EventID=4771 Status=0x18
| transaction IpAddress maxpause=5m maxevents=-1
| where eventcount > 5
```

Listing 2.1: Threshold Rule detecting Kerberos Password Spraying

```
source=XmlWinEventLog:Security EventID=4776
   Status=0xC000006A
| transaction Workstation maxpause=5m maxevents=-1
| where eventcount > 5
```

Listing 2.2: Threshold Rule detecting NTLM Password Spraying

By comparing the approaches, it is visible that the threshold values set for the number of events, and the timespan considered, are different. The reason for this is, as also stated

by Metcalf [35], that there are no universally applicable values. The thresholds need to be tuned for a particular AD environment. Moreover, it may not be easy to determine the correct values, as the size of an environment, configuration of *Account Lockout Policy*, and the typical behavior of users must be considered.

Apart from that, statically defined thresholds may be considered a weak element of the detection. Adversaries may simply slower the attack by making fewer attempts over a more extended time period. Such an attack would be missed by the threshold rules, but on the other hand, it may still return valid results for an attacker if it ran undetected over several weeks or months.

Other options for detecting Password Spraying seem to be limited. Metcalf proposes monitoring users in an AD environment via PowerShell script utilizing `Get-ADUser` cmdlet for displaying `badPasswordTime` and `badPwdCount` attributes. This can be used to monitor for anomalies, such as if bad password attempts are within a short time of each other. [35]

Microsoft Defender for Identity[2] defines several Brute Force attack alerts that may trigger also for Password Spraying attacks:

- Suspected Brute Force attack (Kerberos, NTLM) – external ID 2023,
- Suspected Brute Force attack (LDAP) – external ID 2004,
- Suspected Brute Force attack (SMB) – external ID 2033.

These alerts should be triggered when many authentication failures occur using Kerberos, NTLM, or specifically LDAP and SMB are targeted. Based on their description, the alerts should trigger for both horizontal and vertical Brute Force attacks, including Password Spraying. [42]

Microsoft has recently published another solution for Password Spraying detection. It is based on a supervised machine learning system looking for the same password hashes being attempted against users across many AD tenants around the world. The evaluated data incorporates IP reputation, unfamiliar sign-in properties, and other deviations in account behavior. However, this solution is applicable to cloud Azure AD environments only. [43]

As suggested by the Microsoft's approaches, more effective detection of Password Spraying can be based on baselining, learning the behavior of users in the domain, and finding deviations. Therefore, I decided to research the possibility of improving the detection by applying machine learning techniques. Machine learning could identify sources of failed logon attempts that are not consistent with the common login patterns observed in the domain, and may represent Password Spraying attacks. The detection method in this thesis is focused on a single-tenant AD environment and utilizes only native AD auditing capabilities.

[2]More details about Microsoft Defender for Identity are provided in section 3.2.

2.2 Kerberoasting

Kerberoasting attack was first presented by Tim Medin in 2014 [44]. Although several years have passed, this attack technique is still actual and tends to be effective due to underrated administration of AD environments. Kerberoasting has introduced a possibility to extract credentials of remote service accounts. The obtained credentials may allow adversaries to utilize other attack tactics, especially Persistence, Privilege Escalation, and Lateral Movement. [45, 46]

In the Kerberoasting attack, adversaries do not exploit any particular vulnerability; they exploit poor administration of service accounts. Authentication towards services commonly occurs while users access resources in the domain. Detecting Kerberoasting may be challenging, as it is difficult to accurately distinguish malicious activity from users' legitimate behavior.

Sean Metcalf researched this topic and proposed several ways to approach Kerberoasting detection [45]. In my previous work, I studied these methods, and based on them, developed a set of detection rules in Splunk [29]. In the subsequent research [47], the developed rules were evaluated and compared in various modifications on log data originating from a lab environment. Moreover, detection capabilities and implementation considerations of these methods were discussed.

As the results have shown, various methods perform differently. Non-standard approaches seem to be efficient but carry on implementation overhead, and thus their utilization may not be feasible in every AD environment. On the other hand, traditional signature-based rules tend to have a higher false alarm ratio, whose reducing is desired.

2.2.1 Attack Description

Kerberoasting attack takes advantage of how Kerberos protocol is utilized for authentication in AD. It is targeted especially towards service accounts, which is for several reasons. Firstly, credentials of these accounts are used in Kerberos authentication protocol by design, although not in plaintext forms. Secondly, service accounts are often configured with elevated permissions and may be members of privileged AD groups. And lastly, password management of these accounts is often insufficient. The same strict password policy that is usually applied to users may not be applied to service accounts. Consequently, these accounts may be set with weak passwords that are not changed regularly. [45]

The advantage of Kerberoasting is that the attack does not require elevated domain or local privileges. An attacker only needs to control a valid regular account within an AD domain, or to have knowledge of such an account's credentials. This account is used to request one or more service tickets. Alternatively, having the ability to sniff traffic within a domain might also be sufficient, as the service ticket may be sniffed from Kerberos network communication. [46]

Kerberos protocol is the default authentication mechanism in AD, which provides seamless SSO access to services with network logon. This is ensured through the use of service tickets that are issued by DCs. Section 1.2 described the authentication process

under normal circumstances. Following is the description of how the process is abused during the Kerberoasting attack, based on [45] and [46].

A user that desires to access network resources must be first authenticated towards the AD domain itself. That is, the user must hold a valid Ticket-Granting Ticket, obtained from a DC in Authentication Service exchange, which is depicted as steps 1 and 2 in figure 2.3. This reflects the said requirement for a valid account. In case TGT is not available, an attacker who knows the account's credentials may initiate the AS exchange and obtain the TGT.

An authenticated user may subsequently participate in TGS exchange, and request a Service-Granting Ticket from a DC. As mentioned in section 1.2, the target service is identified by its Service Principal Name, which is presented to the DC in *TGS_REQ* message alongside with the client's TGT. The DC looks up the provided SPN in AD and responds with *TGS_REP* message. This message contains a SGT, encrypted with the service's master key.

Under normal circumstances, the client caches the received SGT and presents it to the server hosting the target service. However, in the Kerberoasting attack, the attacker does not access the service. Instead, they export the service ticket from memory. The ticket can be exported without *Administrator* rights, as it is cached by a user-space process. This can be done by different tools, such as *Mimikatz* [48], which is demonstrated in figure 2.4. There is no communication with the target server, as the SGT is not used for service access at all, as depicted by the diagram in figure 2.3.

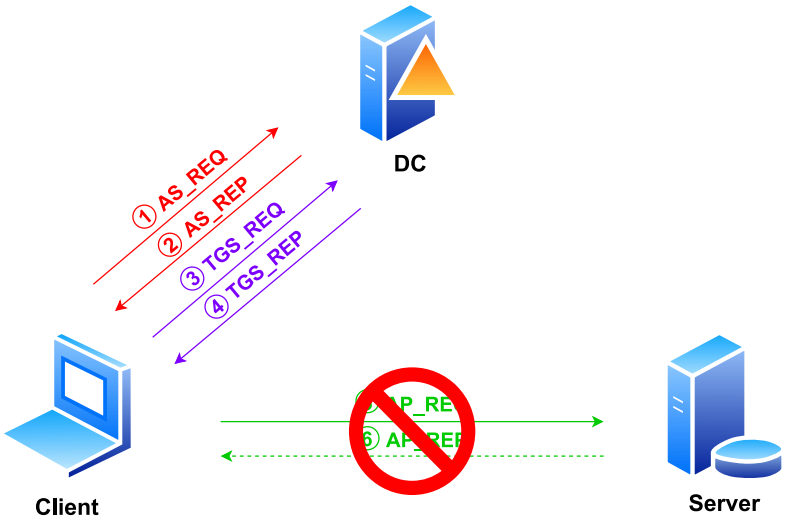

Figure 2.3: Kerberoasting attack diagram

After the SGT is exported, the attacker can attempt to open it by trying different password hashes. When the ticket is successfully opened, the password of the service account is discovered in plaintext. Cracking of the ticket can be done completely offline, on an attacker's machine outside of the AD domain.

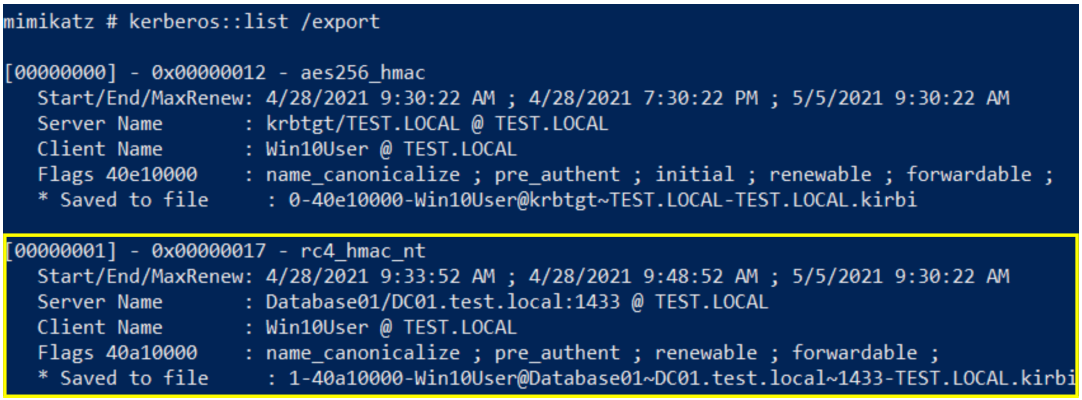

Figure 2.4: SGT exported using *Mimikatz* tool

The reason why all of this is achievable originates in the method how a DC derives the service's master key, which is used to encrypt the SGT. This key is derived from the service account password, and the exact algorithm depends on the encryption type used. Table 2.4 enumerates encryption types implemented in Windows OS. [12]

Code	Cipher suite
0x1	*DES-CBC-CRC*
0x3	*DES-CBC-MD5*
0x11	*AES128-CTS-HMAC-SHA1-96*
0x12	*AES256-CTS-HMAC-SHA1-96*
0x17	*RC4-HMAC-MD5*
0x18	*RC4-HMAC-EXP*

Table 2.4: Encryption types used with Kerberos

For compatibility reasons, the key used for *RC4-HMAC* encryption types is equivalent to Windows NTLM password hash, as denoted in *RFC 4757* [49]. This implies that if *RC4-HMAC* cipher suites are used, the service ticket is encrypted directly with the NTLM password hash of the service account. Brute force cracking of the ticket is possible by trying pre-computed hashes of different passwords. It is also computationally feasible, as cracking of RC4 password hashes is much faster than corresponding AES hashes [50].

Cipher suites types based on Advanced Encryption Standard (AES) encryption have been set as the default starting from Windows Vista / Server 2008, replacing previous default RC4 cipher suites. The cipher suites based on Data Encryption Standard (DES) have been disabled in Windows 7 / Server 2008 R2 OS versions. These changes were made to comply with *RFC 4120* [11]. Although RC4-based encryption types are no longer used by default in modern versions of Windows OS, these are not disabled and may still be used. Organizations often restrain from disabling these encryption types due to backward compatibility and possible authentication errors that may arise if such configuration is applied domain-wide. [50]

As documented in [51], the encryption type of SGT is selected as the strongest common type supported by both the target service and the DC. Client capabilities are not considered, as it never decrypts the service ticket. The encryption type is determined according to `msDS-SupportedEncryptionTypes` attribute, which lists supported encryption types of a service account. AES is automatically added to this attribute in newer Windows versions.

However, based on William Schroeder's research [50], this property is only set by default on computer accounts, not user accounts. The default behavior for user accounts is to use RC4 cipher suites unless the option *This account supports AES encryption* has been selected in the account properties in AD, or use of RC4 is disabled at the domain level.

Figure 2.5 demonstrates the described behavior on a *TGS_REP* Kerberos message captured with *Wireshark* tool in a lab environment. A service ticket was requested for the `Database01/DC01.test.local:1433` SPN, manually registered in AD with a user account. The encryption type of the obtained SGT was *RC4-HMAC-MD5*, although the OS version of the DC was Windows Server 2016.

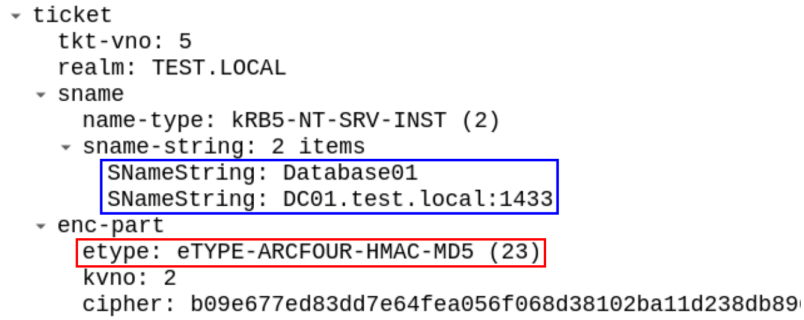

Figure 2.5: SGT in a captured *TGS_REP* message

SPNs are registered in AD with service accounts. A service account may be either a user or a computer account. Computer accounts have long and complex passwords, created and managed by the Windows system itself, that cannot be cracked in reasonable time [45]. User accounts are created and registered with SPNs manually. Their passwords are managed by administrators, unless configured with *Group Managed Service Accounts* service, in which the system manages the passwords similarly as with computer accounts [52]. With manual password management, there is a risk that service accounts will be configured with a weak password, which may not be changed regularly or even set never to expire.

To summarize the observations outlined in the previous paragraphs, the following is assumed. Adversaries performing the Kerberoasting attack will primarily target manually registered SPNs, mapped to user accounts. These accounts could be set up with weak passwords and will likely default to RC4 encryption of the issued service tickets. In that case, those may be vulnerable to brute force password hash cracking, which can be done offline and in a feasible time.

Before Kerberoasting can be executed, an attacker must be aware of SPNs available in the target environment. AD allows every authenticated user to query for accounts with registered SPNs, so a client may identify the correct SPN of a service instance that it desires to access. This way, an attacker may query AD to enumerate all service accounts, which is also called *SPN Scanning*. [45]

Apart from the account name itself, this technique may also reveal other valuable information for the attacker. That is due to SPN format described in section 1.2, as it includes the target host, port number, and service type, which may in relation with the list of well-known SPNs [16] also reveal the purpose of a service account. As further described in [45], while SPN Scanning, attackers may filter not only based on service type but also for accounts that are members of highly privileged groups, having elevated permissions. These accounts may allow an attacker to achieve Privilege Escalation.

In a simplified summary, Kerberoasting attack may be considered as consisting of three relatively distinct steps:

1. SPN Scanning for available services.
2. Requesting service tickets for a particular SPN.
3. Exporting the obtained ticket from memory and cracking it.

There are many tools capable of performing the Kerberoasting attack or some of its steps. The technique has also become part of many offensive frameworks, including *PowerSploit*, *Empire*, or *Impacket* [46]. Some of the tools, such as *Rubeus* [53], allow setting different options for targeting the attack, including filtering of SPNs, specifying desired encryption types, or enumerating only accounts whose password was last changed before the specified date. Furthermore, SPN Scanning and subsequent requesting of service tickets can also be performed by built-in PowerShell commands [45]. Exporting of the service tickets from memory can be achieved using tools similar to *Mimikatz*, as shown in figure 2.4. Commonly used cracking tools, like *John the Ripper* and *Hashcat*, can be employed to crack the exported ticket [46].

2.2.2 Detection

Research of various approaches for detecting Kerberoasting identified several relevant concepts which are to be elucidated further:

1. Monitoring based on indicators present in the event *4769: A Kerberos service ticket was requested*, audited on domain controllers.
2. Detecting usage of attack tools or invocation of suspicious commands, built on top of PowerShell and *Process Creation* events collected from workstations.
3. Implementing service account honeypot and monitoring its access.
4. Using Microsoft Defender for Identity tool and its pre-defined alerts related to Kerberoasting.

Starting from the bottom of the list, the first option is to utilize Microsoft Defender for Identity[3], if available in the target AD environment. Based on the documentation of the pre-defined alerts [42], there are two of them related to the Kerberoasting attack:

- Security principal reconnaissance (LDAP) – external ID 2038,
- Suspected Kerberos SPN exposure – external ID 2410.

Alert ID 2038 aims to identify suspicious LDAP enumeration queries that may represent SPN Scanning activity. The tool initially profiles and learns behaviors of legitimate users. After the learning phase, alerts should be generated for suspicious activities. The alert ID 2410 should catch usage of Kerberoasting attack tools, such as *PowerSploit* or *Rubeus*, and does not need any learning period. [42]

Moving further, Metcalf proposed an effective method for detecting Kerberoasting based on honeypot service account [45]. To implement this method, a fake account is created in the AD environment and is associated with an SPN. Subsequently, this account is monitored for any service ticket requests. Since there is no legitimate reason to request tickets for this honeypot service by normal users, the detected activity will likely be malicious. Additionally, it is possible to make such an account more attractive for attackers by setting its `AdminCount` attribute or making the account member of privileged groups, creating an illusion that the account has elevated rights in AD.

This approach was among the methods tested in our research [47], where it proved to be effective and had no false alarms. Although this method requires some additional configuration in the environment, it is not so complicated to implement in overall. However, the chance that honeypot monitoring will miss an attack cannot be ruled out in reality. There is a possibility that an attacker would not request service tickets for all SPNs available in the environment or would avoid requesting tickets for accounts that have suspiciously high privileges. Therefore, it may be necessary to supplement this method with other detection approaches.

As stated in the previous section, Kerberoasting, or its part, may be performed by using different attack tools. Attackers may also utilize PowerShell to run these tools or directly use it to perform SPN Scanning and request service tickets through commands from the AD module [45]. It would be possible to implement signature-based monitoring for these activities, such as detecting names of known attack tools or detecting invocation of suspicious PowerShell commands. Collecting events logging creation of new processes, such as *4688: A new process has been created*, or PowerShell events for *Module Logging* (4103) and *Script Block Logging* (4104), would be necessary. Alternatively, we proposed a detection rule that performs a full-text search in the PowerShell events, matching strings containing names of service accounts provided in a list [47].

These approaches have several limitations. Firstly, some signature-based detection methods can be easily evaded by attackers. For example, if adversaries rename the attack tools used, a detection rule searching for tools' names will not catch the activity. Similarly,

[3]More details about Microsoft Defender for Identity are provided in section 3.2.

detection of suspicious PowerShell commands based on string matching can be eluded by obfuscation techniques.

Another downside of the mentioned approach arises from the fact that the Kerberoasting attack would typically be executed on a client computer. To detect tools and commands executed by users, collecting relevant logs from workstations is necessary. However, this log data may not always be available. Workstations are typically the most numerous group of assets in organizations, producing significant amounts of log data, which reflects into licensing costs of the monitoring solutions. Therefore, workstation monitoring is often not implemented due to financial reasons.

The second step of Kerberoasting involves requesting a SGT for the chosen SPN from a DC, which is the only part of the attack that can be audited on the DC. This can be done by enabling the *Audit Kerberos Service Ticket Operations* subcategory of *Advanced security audit policies*. As I have already reviewed in [29], the only event from this auditing subcategory bringing value for Kerberoasting detection is the event *4769: A Kerberos service ticket was requested*, which generates every time a DC gets a request for SGT [54].

As requests for service tickets regularly occur during the authentication process, events 4769 are usually logged in high volumes. Detection of Kerberoasting based on monitoring of these events was researched by Sean Metcalf [45], who named indicators that can be leveraged to narrow the number of events and identify possibly malicious actions:

- excessive requests for different services, with a small-time difference between each other, from the same user,
- service tickets issued with a weak encryption type (such as RC4).

We utilized these indicators to develop several detection rules and evaluated their performance in our previous research [47]. The best results were obtained by combining both indicators, looking for an excessive number of service ticket requests in a short time from the same user, using weak cipher suites. Splunk implementation of this detection rule is displayed in listing 2.3. The rule would trigger an alert if there is a sequence of requests from one IP address, requesting more than five services, and the maximum of 5 minutes elapsed between the requests.

```
1 source=WinEventLog:Security EventID=4769
    TicketEncryptionType IN (0x1, 0x3, 0x17, 0x18)
2 | regex ServiceName != "\$$"
3 | transaction IpAddress maxpause=5m maxevents=-1
4 | eval services=mvcount(ServiceName)
5 | where services > 5
```

Listing 2.3: Threshold Rule detecting Kerberoasting

The results have shown that although this method narrows the volume of events, false detections still occur. Especially in environments where older encryption types cannot

be completely disabled due to compatibility reasons, service tickets encrypted with these cipher suites may appear regularly.

Detection rules similar to that in listing 2.3 require setting a threshold value for the number of requested service tickets and for the timespan considered. The key is to define the correct values for these thresholds, as they greatly influence the number of false alerts and the detection sensitivity of the rule. The optimal values may be different for every AD environment.

In case some sources occasionally request a higher count of service tickets, these would trigger false alarms. This situation can be solved by whitelisting – creating a list of known sources behaving as such and excluding them from the detection rule. However, this approach poses two problems. First, it may not be easy to determine sources that should belong to the list and maintain it over time. Second, a host listed in a whitelist may get compromised by an attacker and subsequently be an actual source of Kerberoasting attack.

Moreover, advanced attack tools, such as *Rubeus*, allow an attacker to set a time pause between individual requests for service tickets [53]. By expanding the attack timeframe, attackers may evade detection by rules that monitor only a short time period.

To effectively detect Kerberoasting, it is necessary to find irregular patterns of activity in users' behavior. An example would be a higher number of service ticket requests from a user who does not usually access such services. [46]

Machine learning techniques could identify anomalies in the behavior of users accessing services in an AD domain and thus help to detect Kerberoasting more accurately. The normal behavior should be identified by considering more features and properties, as opposed to only service count considered in a threshold rule. Additionally, this approach could also eliminate the need for maintaining extensive whitelists of legitimate sources.

Chapter 3

Machine Learning & Security Monitoring

Machine learning (ML) nowadays experiences increasing popularity and helps to solve different problems in various real-world applications, not excluding computer security. This chapter provides a brief introduction to ML algorithms and identifies methods suitable for security monitoring, with respect to the existing research in this area. Further, the Splunk tool is introduced, with a focus on its ML support. Consequently, an approach utilizing the researched concepts is formulated.

3.1 Machine Learning Algorithms

Machine learning algorithms are commonly organized into a taxonomy that differentiates between multiple categories based on the type of "learning", purpose, and the desired outcome of the algorithms. There are slight variations in the categories different sources [55, 56, 57] identify, but the following four basic groups are typically recognized:

Supervised learning generates a function for mapping inputs to outputs based on labeled data, containing example input-output pairs. The inferred function is then used for mapping unseen data samples. Typical tasks of supervised learning are

- classification – if there is a discrete number of possible outputs (categories),
- regression – for continuous output values.

Unsupervised learning attempts to model the inputs, find structure, groupings, or patterns in data without labels. The model is used to derive meaningful insights or assign labels to the data. Common applications of unsupervised learning include

- clustering – grouping inputs based on their similarities,
- association – discovering relationships between inputs.

Semi-supervised learning combines the above. It aims to extend the knowledge learned on a small amount of labeled data to a larger amount of unlabeled data and perform a task of otherwise supervised or unsupervised learning.

Reinforcment learning uses observations gathered from interaction with an environment to guide the learning algorithm in a sequential fashion. Unlike in supervised learning, there are no labeled examples, and opposed to unsupervised learning, the model can perceive feedback.

For each of the categories, there is a number of known algorithms and techniques. Those usually originate in different mathematical principles that determine how a model is created. ML algorithms are controlled by *hyperparameters* – adjustable parameters whose values affect the learning process and performance of the resulting models. [58, 59]

Various algorithms have different characteristics, advantages, and disadvantages. The choice of a particular type of learning, and subsequently a concrete algorithm, depends on various factors, but mainly on the task to be solved with machine learning, and the kind of input data. Different problems or real-world applications may have specific requirements that limit the possible options.

Security monitoring represents the task of analyzing vast amounts of log data and finding suspicious activities representing threats. It is based on the premise that attacks will manifest in the form of different characteristics of the logged events. In general, two approaches are commonly identified [30, 59] for this problem:

Misuse detection, also known as signature-based detection, models normal and abnormal behavior from known attacks. Patterns (signatures) of known attacks are defined based on past activities, and this knowledge is used to detect attacks in new data. Misuse detection can usually detect known attacks, though false alarms and missed detections are not uncommon, but has the disadvantage of not being capable to detect new attacks.

Anomaly detection models normal behavior and looks for deviations. The new behavior is identified as anomalous if it is sufficiently different from known normal behaviors. Anomaly detection may discover unknown attacks or attacks that cannot be reliably described by signatures. On the other hand, since all anomalies are not necessarily attacks, anomaly detection can be prone to a higher number of false alarms.

Both anomaly and misuse detection tasks do not have to be necessarily solved by machine learning. Especially for misuse detection, a common approach is to embrace the attack signatures into static conditions that function as detection rules. For anomaly detection, statistical approaches are frequently utilized. However, ML techniques are applicable to both these tasks, as visualized in figure 3.1. [30]

The following subsections describe ML techniques applicable for misuse and anomaly detection, together with a brief description of algorithms utilized in this thesis. The motivation for selecting particular algorithms is further clarified in section 3.4. The methods

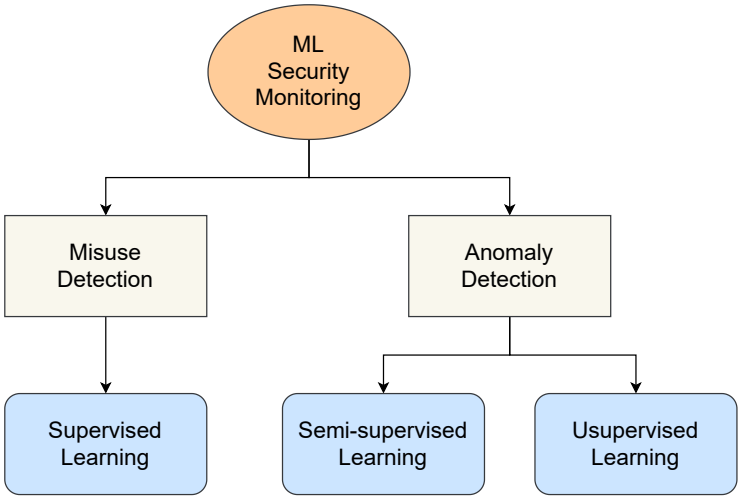

Figure 3.1: Machine learning in security monitoring

are presented with a focus on illustrating their basic concepts and the hyperparameters available for tuning. This section does not aim to provide a comprehensive explanation of the theory and mathematical principles behind the algorithms, for which references to the relevant literature are provided.

3.1.1 Misuse Detection

Misuse detection aims to identify characteristics of known attacks. Learning the difference between attacks and regular events can be understood as a binary classification problem. In binary classification, the outputs may acquire values of two categories only. In the field of security monitoring, the observed activity is either malicious or benign. [30]

Binary classification is a typical task of supervised learning. Variety of supervised ML algorithms are suitable for binary classification problems, some of the most common are:

- Decision Tree,
- Random Forest,
- Support Vector Machine,
- k-Nearest Neighbor (k-NN),
- Naive Bayes.

Random Forest and Support Vector Machine were two supervised algorithms utilized in this thesis for misuse detection.

3.1.1.1 Random Forest

Random Forest is an ensemble algorithm combining bagging and decision trees, introduced by Breiman [60]. *Bagging*, or bootstrap aggregating, is a technique that creates subsets of the training set by its uniform random sampling with replacement. Random Forest is constructed as a combination of tree predictors, where each tree depends on one of the independent samples. The prediction is typically made by the majority vote of the trees or by averaging their outputs.

A *decision tree* represents a discrete model for the classification process, performed by traversing the tree structure from the root to one of the leaf nodes. Each node on the path corresponds to a specific test of an attribute value, while the result of each test directs the output to one of the next nodes in the tree. Leaf nodes provide the classification of a specific instance. Different algorithms exist for constructing the trees, which is typically performed in a top-down, greedy fashion. An important part of the construction process is selecting an attribute to be tested at a splitting node, which can be performed according to different criteria. [61]

Classification based on individual decision trees may be sensitive to changes and tends to overfit the input data. Random Forests are composed of multiple decision trees, allowing the individual trees to be more shallow. Furthermore, random sampling of the training data leads to diverse decision trees. These characteristics make the Random Forest algorithm more robust and less prone to overfitting. [58, 60]

Apart from the number of estimators in the forest, the most important hyperparameters of Random Forest include the maximum depth of the individual trees and the number of randomly selected features at each candidate split in the learning process. [58]

3.1.1.2 Support Vector Machine

Support Vector Machine (SVM) is a technique principally based on the capability of separating data in a multidimensional space. SVM attempts to construct an optimal decision *hyperplane*, separating different data classes with a margin as wide as possible. The *margin* is defined as the distance between the hyperplane and the closest data point. The separating hyperplane depends on the data points that lie on (or within) the margin, called *support vectors*. [59]

SVM was historically developed in three major steps. The original idea of linearly separating points by a hyperplane was supplied by the use of *kernel functions*. Different kernel functions allow SVM to construct the optimal hyperplane in situations where a non-linear curve can separate the data more efficiently. In such a case, the kernel function transforms the dataset into a higher dimensional space making it possible to perform the linear separation, which is also called the kernel trick. [59, 62]

Soft margins were another addition to the SVM algorithm. In situations where the classes are not completely separable, SVM constructs a hyperplane that will maximize the margin and minimize the number of instances that fall off the margin. The misclassified instances are penalized by a user-controlled regularization parameter. [58, 61]

Consequently, important hyperparameters of SVM algorithm include the regularization parameter and the choice of a kernel. Commonly used SVM kernel functions are linear, polynomial, sigmoid, or radial basis function (RBF). Some of the kernel functions can be controlled by additional parameters, such as the degree of the polynomial kernel function or RBF kernel coefficient. [58]

3.1.2 Anomaly Detection

The goal of ML algorithms in anomaly detection is to learn the normal behavior. The premise is that the profile of the normal behavior is significantly different from that of the anomalous behavior. The learned models are then used to identify deviating events that are possibly representing malicious actions. Additionally, anomaly detection can identify new attacks or attacks with unusual characteristics. [59]

Utilizing anomaly detection in security monitoring introduces several challenges. While data corresponding to normal behavior is usually available, labeled data representing anomalies is not. This limits supervised techniques, as they need labeled samples of both types of behavior. Moreover, constant changes in the network environment may also change normal behavior patterns, resulting in a deteriorated performance of the learned models. [30]

To overcome these limitations, unsupervised and semi-supervised ML methods are usually employed for anomaly detection. These methods are applied to input data without prior knowledge about the data labels or patterns of attacks. As described by [63], the two approaches slightly differ in assumptions about the input data:

Unsupervised, also called *outlier detection*, assumes the training data contains outliers, which are defined as examples that are far from the others. The model fits dense areas in the data, ignoring deviant observations.

Semi-supervised, also called *novelty detection*, expects that the training data is not polluted by outliers. The model profiles normal behavior and is used to decide whether a new observation is an outlier (novelty).

ML algorithms for anomaly detection are frequently based on distance-based or density-based clustering, methods estimating the distribution of the normal observations, and since recursive partitioning can be represented by a tree structure, also tree-based techniques [30, 58]. The following algorithms were utilized in this thesis:

- One-class SVM,
- Local Outlier Factor,
- Isolation Forest.

3.1.2.1 One-class SVM

One-class SVM is a semi-supervised algorithm introduced by Schölkopf et al. [64], developed as an extension of SVM for the purpose of novelty detection. One-class classification differs from standard SVM classification in the sense that it learns data from one class only, the normal class, while there are no or few samples from the abnormal class. One-class SVM is useful in situations where it is desirable to detect novelty, but it is not necessary to estimate a full density model of the data.

One-class SVM algorithm estimates a distribution that encompasses the majority of the input data and constructs a hyperplane separating all data from the origin of the feature space, such that its distance to the origin is maximal. For a new observation, it is determined which side of the hyperplane it falls on and hence classified whether it is a regular or a novel observation.

Similarly as with SVM, its one-class variant supports different kernel functions allowing to use the algorithm with linearly non-separable data. In addition to the choice of a kernel, One-class SVM introduces parameter ν. This parameter represents an upper bound of the fraction of outliers and a lower bound on the fraction of support vectors. Informally, ν corresponds to the ratio of anomalies detected in the dataset. [30, 64]

3.1.2.2 Local Outlier Factor

Local Outlier Factor (LOF) is an unsupervised algorithm introduced by Breunig et al. [65], applicable to outlier detection. LOF algorithm is related to the concept of density-based clustering. The locality principle is applied in the calculation of the *density* of a sample, as it is considered with respect to the densities of its local neighborhoods. The outlier factor represents a degree that quantifies how isolated a particular data point is from the surrounding neighborhood.

The density is obtained for every data point by estimating its distance from k-nearest neighbors, where the distance is computed according to the chosen metric. A normal instance is expected to have a local density similar to that of its neighbors, while an abnormal instance is expected to have a much lower density. Samples that have a significantly lower density than their neighbors are considered outliers. [63]

The advantage of LOF is that due to the local approach, it is able to identify outliers that would not be identified in a global perspective. The focus on locality can be directly influenced by setting the number of neighbors considered for the LOF calculation. Other important hyperparameters are related to the computation of the nearest neighbors: the algorithm itself, its parameters, and the metric used for measuring the distance. The number of detected outliers can be controlled by a threshold on the outlier score. [58, 65]

3.1.2.3 Isolation Forest

Liu et al. [66] proposed a different method for anomaly detection, called *Isolation Forest*. Unlike other methods, Isolation Forest does not model normal examples but instead explicitly isolates anomalies. The concept of *isolation* is understood as a separation of

an anomalous instance from the rest of the instances. Another difference of Isolation Forest is that no distance or density measures are utilized to detect anomalies.

The algorithm builds an ensemble of trees induced from the data. Similarly as in Random Forest, sub-sampling is used to construct the individual trees. Observations are isolated by randomly selecting a split between the maximum and minimum values of a randomly selected feature. [58, 66]

Isolation Forest works in two phases. In the first phase, isolation trees are built using sub-samples of the input data. In the second phase, the tested instances are passed through isolation trees to obtain an anomaly score for each instance. The score provides a ranking that reflects the degree of abnormality. Anomalies are more susceptible to isolation, more likely to be separated early, and hence have short average path lengths of the tree structure. The data points can be sorted according to their path lengths (anomaly scores), and the anomalies be identified as the points ranked at the top of the list. [66]

The Isolation Forest algorithm can be controlled by a few key hyperparameters. These include the number of trees in the ensemble, the number of samples used to train each tree, and the threshold on the anomaly scores. [58]

As suggested by the authors, the Isolation Forest method works well for datasets with many irrelevant attributes and in situations where the training set does not contain any anomalies. Additionally, the algorithm has linear time complexity and low memory requirements. [66]

3.2 Applications of ML in Security Monitoring

The idea of utilizing ML to solve the challenges of security monitoring is not new and has been researched heavily over the past years. This is confirmed by numerous publications proposing to enhance detection capabilities of security monitoring solutions and SIEM systems, utilizing both misuse and anomaly detection methods. ML techniques have been applied to malware detection [67], improvement of network intrusion detection systems (IDSs) [68], or detection of particular attack categories, such as DoS attacks [69].

Although research focusing specifically on AD technology is less prevalent, several works attempting to utilize machine learning for discovering anomalous behavior, and detecting attack techniques targeting AD environments, are available.

Hsieh et al. [70] proposed an AD insider threat detection framework built on accounts' behavior sequences. In this framework, a user account's activity is understood as a time-series, and annotated Markov probability model is built for each account in the domain. The state annotations are described using Windows event IDs, while some of the events are co-considered with additional fields. The Markov models should describe the personal tendency for each user to generate specific sequences of event codes.

Every behavioral model is accompanied by a reference probability that depicts how well the model fits the used training dataset and the likelihood that the user produces corresponding logs during routine activity. Anomalies are determined according to a condition based on this reference probability and a user-defined threshold parameter. By

testing this approach in an organization with 95 employees, the authors were able to obtain the best performance of about 66.6% recall and 99.0% precision[4].

Goldstein et al. [71] suggest enhancing SIEM systems with unsupervised algorithms to detect suspicious authentication attempts or unusual user account activities in an AD environment. The advocated approach is based on a global k-NN algorithm, which is applicable without the need for any prior training of the system.

In the experiments, the algorithm was used to find anomalies in Windows authentication logs originating from 57 Windows servers, mostly DCs. Relevant events were provided by the *Logon/Logoff* and *Account Logon* audit policy categories. Several data views were created based on different aggregations per user and time unit. Examples of detections by the system included unusual user activity in non-business hours, anomalous activities of privileged users, and misconfigurations of the environment.

A ML method for detecting advanced attacks against AD was proposed by Matsuda et al. [72]. The research focused on detecting techniques that require Domain Administrator privilege, using events related to processes collected only from DCs. Particularly events *4674: An operation was attempted on a privileged object* and *4688: A new process has been created* were utilized. As those events do not contain information about IP addresses, those were extracted from correlated events *4769: A Kerberos service ticket was requested.*

The experiments covered three algorithms for outlier detection: One-Class SVM, Isolation Forest, and Local Outlier Factor. These were evaluated against a specific attack scenario that included gaining Domain Administrator privilege through a specific vulnerability and a subsequent Golden Ticket attack. The best detection results were obtained using the One-class SVM algorithm. Authors recommend using both event IDs for detection but train the models separately to mitigate false negatives.

Uppströmer and Råberg [73] utilized supervised machine learning for detecting Lateral Movement in AD. The considered attack techniques included AD enumeration, Pass the Hash and Pass the Ticket. Several classifiers were compared, namely Decision Tree, Random Forest, k-NN, SVM, and Multi-layer Perceptron. A semi-synthetic dataset based on different Windows event IDs was used for evaluation.

In the obtained results, the tested classifiers performed differently in terms of recall and precision. Differences were spotted particularly between SVM and Random Forest approaches; SVM is to be preferred in situations where the amount of false positives is essential to be low, while Random Forest is better if a low amount of false negatives is preferred.

A slightly different approach for detecting Lateral Movement in AD was utilized by Meijerink [74]. In his thesis, two anomaly-based methods were compared: HDBSCAN, a method based on clustering, and Principal Component-based Classification, a statistical method based on Principal Component Analysis. The methods were applied to a realistic dataset, and the detection was focused on Windows event *4624: An account was successfully logged on.* The results showed that clustering generally performed better but with a relatively high false-positive ratio, which should be reduced further.

[4]For a detailed explanation of evaluation metrics for ML algorithms see section 4.1.5.

Machine learning can also be found in commercial security monitoring solutions. Microsoft offers a proprietary cloud-based solution called Microsoft Defender for Identity. This tool claims to monitor user behavior and activities, with ML-based analytics across on-premise AD environment. Particularly it monitors DCs by capturing and parsing network traffic and leveraging Windows events. The data is then sent to a cloud service, where it is analyzed for attacks and threats using profiling, deterministic detection, machine learning, and behavioral algorithms. [42]

As mentioned in chapter 2, Defender for Identity defines alerts for monitoring suspicious activities in AD. However, this tool is offered as a commercial product requiring additional licensing and is not provided in an AD environment by default. The tool was not available for the experiments in the context of this thesis, and therefore its efficiency cannot be compared to the other detection approaches. The advertised detections cannot be reviewed objectively, as the details of their implementation are not publicly available.

3.3 Splunk Technology

Splunk is a commercial software product acknowledged as a machine data intelligence platform. It allows searching, analyzing, and visualizing machine-generated data using Search Processing Language (SPL) via a web-based interface. Monitoring log data, including security-related, represents a typical use case of the Splunk platform. Indeed, Splunk has been recognized as a Leader in the Gartner Magic Quadrant report for SIEM solutions for several consecutive years. [75, 76]

This thesis uses Splunk technology for the development and implementation of the ML-based detection rules, mainly as it:

- is a state-of-art platform, commonly adopted by organizations as a SIEM solution,
- supports parsing and processing Windows Event Log data,
- offers possibilities to build custom machine learning solutions,
- allows for comparison of traditional and ML-based approaches on the same data within the same platform.

This section provides a brief overview of the Splunk platform, its capabilities for security monitoring, as well as the options for using machine learning techniques. I have already covered some concepts of Splunk technology in my previous work, where I utilized SPL to develop a set of detection rules [29]. Ultimately, additional information can be found in the official Splunk documentation [75].

3.3.1 Splunk for Security Monitoring

The main functionality of Splunk is data processing, which comprises the following tiers:

1. Collecting and ingesting data from systems, applications, databases, network devices, files, or other sources.

2. Parsing, transforming, and indexing the collected data.

3. Running searches on the indexed data.

Searches can be saved and scheduled to run recurrently, producing reports or populating dashboards. Both historical and real-time searches can be set up to generate alerts. Alerts can be configured to trigger an action, e.g., send an email or run a custom script, if results meet the specified conditions. The alerting feature is particularly beneficial for security monitoring, as the search query can look for suspicious activities and generate an alert if discovered. [75]

Splunk software is offered as multiple products. The functionality mentioned above represents the core of the platform, called *Splunk Enterprise*. Splunk Enterprise can be deployed to an on-premises or custom cloud organizational infrastructure. Splunk also comes as a managed cloud-based service called *Splunk Cloud*. The products differ in licensing and pricing. Splunk Enterprise licenses are typically ingest-based, measured by daily data ingestion, or infrastructure-based, measured by allocation of virtual CPUs. Limited trial, developer, and free licenses also exist. [77]

Splunk Enterprise can be used as a single instance or as a distributed deployment, depending on the size and needs of an organization [75]. The platform consists of three types of processing components which represent the data processing tiers:

Forwarders reside on the machines generating the data, where they consume the data and forward it to indexers.

Indexers index, transform and store the incoming data, and also search it in response to requests from search heads.

Search heads interact with users, accept search requests and return results back to users.

Splunk functionality can be further extended by *apps* and *add-ons*. Apps generally provide user interfaces or additional functionalities to analyze and display knowledge around data sources, while add-ons typically help gather, normalize, and enrich data sources. Apps and add-ons are developed by the Splunk team, third parties, or as a community effort. Some of them are provided as premium solutions underneath additional licensing, while others may be used free of charge. [78]

Processing of Windows Event Log data within Splunk requires utilization of *Splunk Add-on for Microsoft Windows*. This add-on performs index-time and search-time extraction of knowledge from Windows events and ensures their correct parsing into fields. Different Windows Event Log channels are exposed as distinct source types, for example, `XmlWinEventLog:Security`. Source types are typically used in SPL queries to narrow the searched events. The add-on renders Windows Event Log events in XML format by default. [79]

Splunk organizes the indexed data in the form of events. *Event* is a single data entry with an associated timestamp that can span one or multiple lines. A straightforward example is a Windows event, which maps directly to a single Splunk event. From the

search perspective, an event represents the smallest searchable unit of data. Each event has its attributes, represented as key-value pairs called *fields*. Using fields, it is possible to distinguish between specific events based on their values. [80]

Events can be queried using Splunk's Search Processing Language. SPL allows data searching, filtering, manipulation, insertion, and deletion. SPL syntax defines the following basic language constructs:

- **commands**: `search`, `stats`, `eval`, `where`, ...
- **functions**: `count`, `sum`, `if`, `match`, ...
- **arguments**: `span`, `field`, `maxevents`, ...
- **keywords**: `AND`, `OR`, `IN`, `BY`, `AS`, ...

Search commands have functions and arguments associated with them. Those specify how the commands act on results and which fields they act on. Keywords allow expressing relationships between clauses. Commands are separated using a pipe character, while results from each command are passed as input to the next command, forming a search pipeline. [81]

Chapter 2 already contained several examples of SPL queries. To explain the described concepts on an example, assume the query in listing 2.1. The search retrieves Windows events from the Security channel and selects only those with specific `EventID` and `Status` values[5]. The results are further processed by `transaction` command, whose operation is specified by its `maxpause` and `maxevents` arguments. Finally, results are filtered by the condition using `where` command.

3.3.2 Machine Learning in Splunk

As a contemporary data intelligence platform, Splunk complies with the ongoing trends in the field and provides a solution for using machine learning with the processed data. Support for ML processes is ensured by *Splunk Machine Learning Toolkit (MLTK)* app that acts as an extension to the Splunk platform. The app provides a guided framework for ML workflows but also allows creating custom ML outcomes. [82]

Splunk MLTK functionality depends on *Python for Scientific Computing* add-on. This add-on contains a Python interpreter bundled with the necessary scientific and machine learning libraries, such as *Numpy*, *Scipy*, *Pandas*, and *Scikit-learn*. Splunk MLTK exposes SPL commands that provide an interface for using the ML algorithms defined in these Python libraries within the Splunk platform. [82, 83]

The essential SPL commands for interacting with ML models are `fit` and `apply`. The `fit` command is used to train a ML model based on a selected algorithm. The trained model can be saved as a knowledge object by the `INTO` keyword, and used later with different data using the `apply` command. Both commands apply the model to the timely search results according to their placement in the search pipeline. [82]

[5] Note that there is an implied `AND` between the field-specifying clauses, unless stated otherwise.

As Splunk ingests various types of machine data, having different formats, the data may require preprocessing before it is used with ML algorithms. Possible problems include missing values, non-numerical data, values in differing scales, or string values. For this purpose, MLTK provides several algorithms for scaling or imputing missing data points. Additionally, `fit` and `apply` commands automatically perform some data preparation actions. Any fields and events containing null values or non-numeric fields with more than 100 distinct values are discarded. Further, the remaining non-numeric fields are converted into dummy variables using one-hot encoding. [82]

ML algorithms available in Splunk MLTK cover different ML areas and use-cases:

- **anomaly detection**: Density Function, Local Outlier Factor, One-class SVM
- **classification**: Random Forest Classifier, Logistic Regression, SVC, ...
- **clustering**: DBSCAN, G-means, K-means, Spectral Clustering, ...
- **regression**: Decision Tree Regressor, Lasso, Linear Regression, ...
- **time series analysis**: ARIMA, State Space Forecast

Additionally, supplemental algorithms for feature extraction, data preprocessing, cross-validation, or computation of data characteristics are available. [82]

Although most algorithms in Splunk MLTK are based on implementations from the Python library *Scikit-learn*, not all algorithms from this library are available for use in Splunk. Moreover, particular algorithms do not necessarily expose all possible hyperparameters to be set via SPL. In that case, the default values are used.

In addition to the available algorithms, Splunk MLTK provides an ML-SPL API that allows to import custom algorithms based on the supported Python libraries. This interface is leveraged also by *Splunk MLTK Algorithms on GitHub* app. This app is developed as a community effort and provides implementation of algorithms that can be used with MLTK in addition to the default set. [82, 84]

3.4 Proposed Approach

This thesis introduces monitoring rules utilizing machine learning techniques for detection of two particular attack techniques targeting AD, described in chapter 2. Both Password Spraying and Kerberoasting are "traditionally" detected using threshold rules, which is not sufficient, as those can be easily evaded by attackers or may report a high number of false alarms.

Section 3.2 contained review of existing efforts attempting to utilize ML for detecting AD threats. So far, the majority of research has focused on Lateral Movement tactic or advanced techniques that require higher privileges in AD environment. Credential Access techniques are covered only marginally, as a part of generic approaches detecting anomalies from a variety of logged Windows events.

From the monitoring perspective, detection of both Password Spraying and Kerberoasting is challenging, especially due to the following common traits:

1. Although traces of the attack activities are present in the audited events:
   - a failed logon attempt due to a wrong password in Password Spraying,
   - a request for a service ticket in Kerberoasting,

   the attacks are difficult to spot as they coincide with logs of legitimate user behavior:
   - a user mistypes a password while logging-in,
   - a user accesses a remote network service.
2. The attacks are challenging to detect if adversaries spread the attack over a longer time, reduce the frequency of attempts, and adapt their strategy to the properties of a particular AD environment.
3. Both activities appear as usage of a valid domain user account.

To detect the attacks more reliably, it is required to differentiate between normal and abnormal user behavior. In principle, that is consistent with the purpose of anomaly detection, as presented in section 3.1. Therefore, anomaly detection could be a suitable method for this task. Two basic anomaly detection approaches were identified: unsupervised (outlier detection) and semi-supervised (novelty detection).

Supervised approaches are typically used for misuse detection through learning patterns of attacks. In addition to signature-based detection rules, supervised ML algorithms could learn attack patterns that cannot be easily captured as static conditions. However, the use of supervised methods for attack detection implies multiple constraints that should be considered:

1. Supervised learning requires labeled examples of both benign and malicious events. Samples of attacks are typically not available, and therefore must be synthetically generated or obtained otherwise.
2. The task of attack detection is inherently imbalanced. There are significantly more benign events than malicious events, both during training and production use of the model.
3. Supervised methods may be susceptible to overfitting.

This thesis utilizes all three approaches (unsupervised, semi-supervised, and supervised) and aims to find the most suitable method for detecting the selected attacks. The experiments primarily focus on anomaly detection methods due to their seeming relevance to the task and the potential disadvantages of the supervised approaches.

Two representative algorithms were chosen for each category. To an extent, the choice was constrained by the list of algorithms available in Splunk MLTK; however, emphasis was put on covering diverse algorithm types. The chosen algorithms include tree-based, SVM-based, as well as density-based methods. In the supervised category, the classifiers were chosen based on similarity to the studied anomaly detection algorithms, in order to

ML Algorithm	Type	Available in Splunk MLTK
Local Outlier Factor	unsupervised	Yes
Isolation Forest	unsupervised	Yes*
One-class SVM	semi-supervised	Yes
Local Outlier Factor	semi-supervised	No
Support Vector Classifier (SVC)	supervised	Yes
Random Forest Classifier (RFC)	supervised	Yes

Table 3.1: ML algorithms used in this thesis

compare possible benefits of one or the other approach. Table 3.1 summarizes the ML algorithms that were used in the experiments.

Although Splunk MLTK implements Local Outlier Factor as an unsupervised method only, *Scikit-learn* also includes its semi-supervised variant, which can be enabled by setting the parameter `novelty=True` [63]. Despite its unavailability in MLTK, this variant was also included in the experiments for comparison, as there would be only a single semi-supervised method otherwise.

Isolation Forest is not directly available in Splunk MLTK. Nevertheless, this algorithm can be easily added to Splunk by installing *Splunk MLTK Algorithms on GitHub* app, as described in the previous section.

Chapter 4

Realization

This chapter comprises the practical realization of the proposed approach. The implementation process is thoroughly explained, including the overall methodology used, preprocessing of data, feature engineering, experimenting with different configurations, and analyzing the outputs. The performance of various approaches is evaluated and compared to the traditional detection rules. For each of the attacks, the best approach based on the obtained results is implemented as a Splunk search, prepared to be used for real-time security monitoring of AD environments.

4.1 Methodology

One of the goals of this thesis is to verify the suitability of different ML techniques for use in security monitoring of AD and compare them to a signature-based approach. To fulfill this goal, and to ensure the validity of outputs, it is important to define a consistent workflow for the experiments and their evaluation.

There are already well-established methodologies and process models available for use in ML and data mining (DM) projects. Such models provide high-level control of the project structure and help to ensure that none of the crucial steps are missed.

One of the most popular and broadly adopted models is Cross-Industry Standard Process for Data Mining (CRISP-DM), proposed by a consortium of four commercial companies [85]. CRISP-DM provides an overview of the life cycle of a DM project. It defines phases of a project, their respective tasks, and the relationships between these tasks. In practice, it is used as a standard process framework for designing, creating, testing, and deploying ML solutions. The model describes six phases of a DM project, the sequence of which is not strict, as indicated in the process diagram in figure 4.1 [86].

As CRISP-DM is a robust and well-proven methodology, it was chosen as the baseline for implementing the proposed approach. The following sections describe methods and procedures utilized to fulfill the goal of this thesis consistently with the CRISP-DM framework.

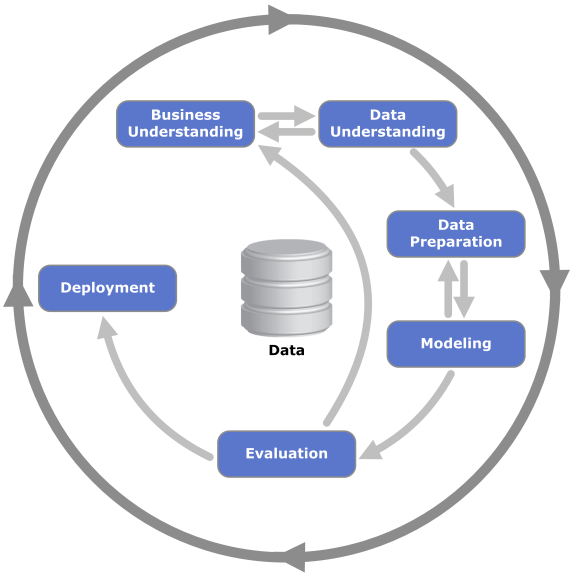

Figure 4.1: Phases of CRISP-DM model

4.1.1 Business Understanding

The main aspect of the first phase is to define the problem that needs to be solved using ML and the goal that should be accomplished. Another important step is to design a project plan that would lead to the achievement of the defined goals, including selecting the tools and techniques, and identify possible constraints. [85]

In the case of this thesis, the ultimate goal is to identify suitable ML techniques for detection of AD attacks and implement them as Splunk detection rules utilizable for security monitoring of AD. An assessment criteria for the developed solution is the comparison of its performance to a signature-based detection approach.

Most of the "business analysis" of the problem was already described in the previous chapters, from analysis of known attacks targeting AD, through the weak points of the currently used detection approaches, to technological aspects of Splunk and Splunk MLTK. As described in chapter 3.3, implementation of ML algorithms in Splunk MLTK is based on Python library *Scikit-learn* [58], however only subset of the algorithms is available for use in the toolkit.

Another important part of the workflow is the data preparation step. Though there definitely are SPL functions allowing for different data transformations, using conventional programming or scripting language may be more flexible and convenient for this task.

Due to the limitations mentioned above, a different technology was used for the data preparation and modeling phases. A straightforward option was to use Python together with *Jupyter*, *Scikit-learn* and other libraries, such as *Pandas*, *NumPy*, and *Matplotlib*; technology stack very popular in the area of data science, as confirmed by 2020 Kaggle Machine Learning & Data Science Survey [87]. These tools allowed automating the processes, performing more effective feature engineering, broader experiments, and comprehensive

analysis of different approaches. Arrangement in the form of Jupyter notebooks allowed for graphical visualizations of the outputs.

The choice of technologies was also supported by the fact that the Splunk platform uses Python, and Splunk MLTK provides an interface to the *Scikit-learn* implementation of the ML algorithms [82]. This ensures that the results are equivalent as if made in Splunk directly, provided that the same data, features, and hyperparameters of the models were used in both cases.

4.1.2 Data Understanding

The data understanding phase encompasses data collection and further activities for recognizing properties and discovering insights into the data. Eventually, possible problems should be identified in this phase. [85]

In the context of this thesis, "data" means Windows Event Log data collected from systems in an AD domain. The log data used in this thesis represents traffic originating from an AD environment of a real organization. This environment has hundreds of daily users, and its AD infrastructure contains several DCs with relevant logging policy configured. Such an environment generates thousands of various Windows events per day. These are collected, parsed, and indexed by a Splunk instance that makes the log data available for querying.

For the needs of ML analysis, a static portion of the log data was extracted, covering a continuous time frame of several weeks. The data was filtered to contain only relevant event IDs for detecting the selected attacks. The extracted data was considered benign, containing only records capturing legitimate traffic.

Next, it was necessary to gain log data representing malicious activities. As it was not feasible to execute the actual attacks in the live AD environment to gather samples, malicious log data was synthetically generated. However, great emphasis was put into the generating process to ensure that the generated data looks similar to the collected events and fits well with the benign samples.

The process of generating malicious events started by executing real tools, capable of performing the studied attacks, in a lab environment that had appropriate logging configured. The tools were executed with different configurations to simulate real attackers. Relevant events were monitored, focusing on specific information and attributes contained in them that represented signs of attack execution. Afterwards, this knowledge was used to create more malicious samples, with several fields modified to fit the properties of the real AD environment.

Further details regarding the data retrieval and the process of generating malicious events are provided in the following sections for each of the attack techniques covered.

4.1.3 Data Preparation

The data preparation phase comprises all the activities required to build a dataset in a format suitable for subsequent use with ML models. These may include selecting, cleaning, transforming, and reformatting data, as well as the process of creating features,

called *feature engineering*. Data preparation is usually considered the most extensive and time-consuming among the CRISP-DM phases. [85]

As early stated, data preparation and ML experiments were performed outside of Splunk, and thus it was necessary to export the data. Relevant log data was searched in Splunk, elementally filtered, and preprocessed into a tabular form by aggregating on specific attributes. This way, the amounts of irrelevant data were limited, and the records could be exported from Splunk in CSV format. This was done for both benign and malicious events.

The exported CSV files were loaded into a Jupyter notebook and further manipulated using Python. Multiple statistical and other custom functions were used on the data to construct features from the values. The majority of fields in Windows events cannot be used directly as features in ML models, as those usually require numeric features. Moreover, the events are limited in the number of available attributes and the textual nature of their values. Therefore, great emphasis was put on the feature engineering step. The set of extracted features was different for each scenario, depending on the attributes contained in the events and the nature of the particular attack techniques.

Once the data was preprocessed and suitably formatted for use in ML models, it was necessary to split the available data into training, validation, and test datasets. However, since the applied algorithms are of different kinds (unsupervised, semi-supervised, and supervised), this step was treated differently for each category.

Firstly, all available data was split into learning and test dataset. A previously unseen portion of data was required to perform an unbiased performance comparison of the created ML models. This applies to all the three ML approaches, as well as the threshold detection rules. The test dataset was kept aside and not used in any part of the modeling phase.

Secondly, the learning dataset was split into training and validation subsets. The training dataset was used to fit the models of supervised and semi-supervised methods. Specifically, in this thesis, the semi-supervised methods are used for novelty detection, which implies that their training dataset must not be polluted by outliers [63]. The training dataset for semi-supervised methods was therefore cleaned of malicious examples. For both supervised and semi-supervised methods, the fitted models were successively applied onto the validation dataset, which was then used for evaluation and tuning of the models' hyperparameters.

Unsupervised methods do not require training a model, so there is no need for a separate training dataset. However, since the dataset used in this thesis was labeled, it was employed to tune hyperparameters of the unsupervised models. The whole learning dataset was used for this purpose. Figure 4.2 illustrates the explained data splitting process for different ML approaches.

4.1.4 Modeling

In the modeling phase, various ML algorithms are applied to the prepared data, and hyperparameters of these algorithms are calibrated to optimal values. The quality and validity of the built models must be assessed based on relevant metrics. [85]

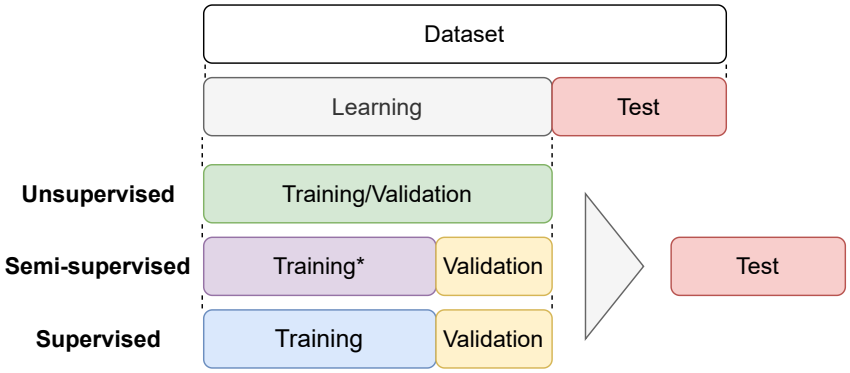

Figure 4.2: Dataset splitting process

After selecting the particular algorithms to be used, the following steps were performed for each approach:

1. *feature selection*,
2. *hyperparameter tuning*.

The first step aims to select the most informative attributes and remove possibly redundant or irrelevant features that may be present in the dataset. The primary motivation is not to reduce the dimensionality of the dataset[6], but to avoid computing unnecessary features, with only a little or no impact on the resulting model. As the developed models would be part of rules deployed for real-time security monitoring, it is essential to reduce the steps that are computationally costly.

A wrapper method was utilized for feature selection. *Wrapper methods* are multivariate methods that consider subsets of features at a time and use a classifier to assess the performance [56]. Several different subsets of features were formed based on various preconditions. These were supplied to the chosen ML methods and assessed based on a scoring metric on the validation dataset. The best combination of features was identified for each approach and used in further experiments.

The second step, hyperparameter tuning, or hyperparameter optimization, is the process of finding the configuration of hyperparameters that results in the best performance of a ML model [59]. To determine the best values of hyperparameters for a particular model, an exhaustive approach was used. A predefined *parameter grid*, containing number of different values for hyperparameters, was exhaustively searched while all parameter combinations were generated. Performance of estimators set-up with these different configurations was measured on the validation dataset, compared based on the chosen scoring metric, and the best combination for every algorithm was retained.

Both the steps described above evaluate the performance of the used configurations (feature subsets and hyperparameters) based on a chosen evaluation metric. In general,

[6]In the case of attributes contained in Windows events, the situation is inherently the opposite – there are not many features available.

many metrics can be used to measure the performance of an estimator, and the selection depends on the goals and objectives examined. While in the modeling phase, assessment is done in direct relation to the model itself, metrics are also used in the evaluation phase, where overall results are taken into account.

4.1.5 Evaluation

In the evaluation phase, the models that have been already built are evaluated on test data. The key is to determine whether they achieve the business objective, and a decision is to be made on the final use of the results. The evaluation methods and criteria are usually dependent on a particular use case scenario. [85]

The business objective of this thesis is to improve detection capabilities in comparison with traditional signature-based rules. Therefore, the performance of the designed ML models was measured on the test dataset and compared to the performance of a traditional rule aiming to detect the same attack technique. The main focus was to estimate the improvement possibly gained by the proposed ML approach. To properly perform this step, a choice of a suitable metric was required.

Security monitoring deals with a binary classification task – the activity is either malicious or benign. Ultimately, the desired state is to be alerted on every attack, but simultaneously receive as few alerts as possible. In case an alert appears, the underlying activity detected should be malicious with the highest certainty possible.

A commonly used concept for evaluating classifiers is the *confusion matrix*. Confusion matrix offers a simple tabular summary of predictions made by a ML model, allowing for a simple evaluation of its performance. In binary classification, the matrix has two dimensions that compare the predicted and actual values for the two classes. [56]

The terminology originating from the concept of the confusion matrix is directly applicable to the area of security monitoring and is, as such, commonly used in practice [30]. Figure 4.3 illustrates its use for evaluating attack detection that can be understood as follows. An observation is considered:

- **true positive (TP)** if there is a real attack that triggers an alarm,
- **false positive (FP)** if an alarm is produced, but there is no attack (type I error),
- **false negative (FN)** if there is an attack, but no alarm triggers (type II error),
- **true negative (TN)** if there is no attack and no alarm is produced.

While the tabular format containing nominal numbers of predictions can be descriptive, there are cases where a single number representing the score of a model is desired. Eventually, it is easier to compare the performance of various models, or different configurations of a model, based on a single number in the interval from 0 to 1.

A wide range of metrics for scoring binary classification models exists, while these metrics are typically derived from the concepts of confusion matrix [56, 88]. Several of them are outlined below, with an emphasis on their usage in this thesis.

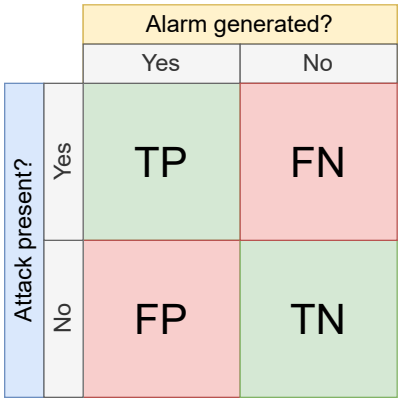

Figure 4.3: Confusion matrix for attack detection

Accuracy (ACC) reflects how many observations was the model able to classify correctly:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}.$$

Accuracy is not a very suitable metric for evaluation of imbalanced problems, which the task of attack detection unfortunately is[7].

Precision, also called **positive predictive value (PPV)** measures how many observations predicted as positive are in fact positive:

$$PPV = \frac{TP}{TP + FP}.$$

Higher precision means that higher ratio of alarms is representing an actual attack.

Recall, also called **true positive rate (TPR)** indicates how many observations out of all positive observations were classified as positive:

$$TPR = \frac{TP}{TP + FN}.$$

In attack detection, it tells us how many attacks were caught, or recalled, from all malicious events.

F-score combines precision and recall into one metric. In general form, it uses a positive real parameter β, which represents number of times recall is considered as important as precision:

$$F_\beta = (1 + \beta^2)\frac{PPV \times TPR}{\beta^2 \times PPV + TPR}.$$

[7] A model simply classifying all observations as benign would achieve high accuracy, but at the same time would be hardly useful for security monitoring.

For $\beta = 1$, F_1-score represents harmonic mean of precision and recall. Other common values are $\beta = 2$ (F_2) and $\beta = 0.5$ ($F_{0.5}$) that bias towards recall and precision, respectively.

While designing mechanisms for attack detection, the aim is to minimize both type I error (number of FPs) and type II error (number of FNs). However, in practice, the impact of these errors may not be equal. Simply put, an undetected attack against AD may result in a domain compromise, whereas a FP alert may be "only" worth waste of time and effort spent on investigating the alarm by a security analyst. Therefore, F_2-score, which reckons recall twice as important as precision, was used as the decision metric for comparing models' configurations in the modeling phase. The main tool used for the final assessment of the approaches on test data was the confusion matrix.

4.1.6 Deployment

The final phase of CRISP-DM framework encompasses finalizing the designed solution. It covers all the steps necessary for its successful deployment and the actions ensuring that the model can fulfill the business goal it was designed for. [85]

In this thesis, deployment means providing Splunk SPL searches implementing the selected approaches. The provided queries should extract all the necessary features from the indexed events by using suitable SPL commands and transform them to the format required by the chosen ML models. The models would be fitted and applied as a part of the queries. For real-time security monitoring of an AD environment, the searches should be saved and scheduled to run regularly on a Splunk instance.

4.2 Password Spraying

One of the candidate scenarios in which ML approach could help improve detection is Password Spraying. Its detection is typically based on monitoring a high number of login failures per source and requires determining a threshold for alerting. In an attempt to avoid detection, adversaries may spread a Password Spraying attack over a long period of time. In that case, only a few failures would be observed, and an alarm would not trigger. By utilizing ML techniques, it might be possible to detect authentication failures that are not necessarily noisy in amount but deviate from normal activity in the domain.

4.2.1 Data Preparation

To detect Password Spraying, it is necessary to collect events logging authentication failures from hosts in the AD environment. Based on the research described in section 2.1, events *4771: Kerberos pre-authentication failed* were collected from all domain controllers, and events *4625: An account failed to log on* from all DCs and also member servers. Events 4625 were preferred for NTLM authentication over events 4776 due to the higher level of details they provide. To identify only failures caused by bad passwords, both events can be filtered for specific status codes, as listed in table 2.2.

The most important fields contained in these events are the attributes identifying the target user account name and the source from which the authentication attempt is coming. Only a few attributes are provided consistently in both events: `Computer` field, which records the hostname of the computer processing the authentication request, and information about the target account provided in the field `TargetUserName`.

With the source, however, the situation is more complicated. Event 4771 records only network information – IP address of the client host as `IpAddress` field. Event 4625 contains both IP address (`IpAddress`) and hostname (`WorkstationName`) of the client host. However, in practice, the `IpAddress` field is often empty (filled with a dash symbol) in this event, and cannot be relied on. Since there was the need to unify the information about the source for purposes of further aggregation, the following approach was used for the event 4625: the value of `IpAddress` field was checked to contain a valid IP address; otherwise, the value in `WorkstationName` was used to identify the source. Note that this type of aggregation would not be possible with events 4776, as those miss information about the IP addresses.

For the Password Spraying attack, many failure attempts towards different accounts from a single source are expected. Such an attack may span from a few seconds or minutes, to several hours or even days. To have a chance to detect attacks spanning extensive time periods, it is essential to have an overall view of the behavior for a more extended time unit. Hence, the events were split into 1-day bins and aggregated per unique source. As a result, one record represented all authentication failures from a single source that occurred during one day.

4.2.1.1 Feature Engineering

The next step was to extract all the possible features across the records. The objective was to create as many features as possible and supply them to the feature selection phase. However, this process was significantly limited by the number of attributes contained in the events. The features created were centered around three main concepts: count of values, properties of user accounts attempted, and time differences between the attempts.

Features evidently included are the total count of attempts (`cnt`) and the distinct count of user accounts (`user_dc`). Further, a distinct count of computers (`computer_dc`) responding to the authentication attempts was calculated. As described in section 2.1, logging of a particular event ID depends on the accessed service. Count of different event types may thus hint at the attack tools used. This is captured in features `evt_4625_cnt` and `evt_4771_cnt`.

Next, it is interesting to consider the strategy of a potential Password Spraying attack. An attacker would probably target many accounts, where one or more passwords would be attempted for these accounts. However, there is usually an account lockout policy configured in AD environments. Attackers could systematically avoid the risk of locking out accounts by limiting the number of attempts per user. To reflect this, multiple features deal with count of failures per user account: `user_max`, `user_avg`, `user_mod`, and `user_mod_cnt`.

Typical AD environments involve different kinds of accounts. Apart from accounts used by regular users, there may be accounts assigned to services (such as databases or web applications), computer accounts, or accounts dedicated to running scripts or scheduled tasks. Organizations commonly utilize a naming convention that allows distinguishing between various account types easily.

Adversaries may be aware of, or may deduce, the naming convention in place. In that case, they might target only a specific type of accounts by the Password Spraying attack. For example, only personal accounts, as people tend to create weak passwords. Therefore, differentiating between the types of targeted user accounts could also be beneficial. For this purpose, the features `user_pers_dc`, `user_np_dc`, `user_sys_dc`, and `user_other_dc` were created.

To evade detection, attackers may spread a Password Spraying attack over a longer period of time. This can be achieved very easily by modifying the attack script to include a pause between the attempts. The assumption is that such an attack would introduce regularities in the timestamps of the logged events, e.g., failure events generated repeatedly every few seconds or minutes. Consequently, several features that reflect time differences between the logged events were created (`tdelta_*`). These include sum, average, maximum, minimum, and mode of time differences.

Finally, several features characterizing counts were also represented as ratios. The full list of 24 features created and subsequently considered in the feature selection phase is listed in table 4.1.

To transform data from the events into the desired format and extract all the features, it is necessary to use various evaluation functions. While for some features, the approach is quite straightforward, more complicated SPL constructs are needed for others. Listing 4.1 displays the Splunk search used to retrieve the relevant events. This query already uses `stats` command with different functions to perform the aggregation and create some of the features. At the end of the listed search, a condition filters out records containing failure attempts towards a single account, as failures related to one account only are not of interest for Password Spraying detection.

Auxiliary fields `user_cnt` and `orig_time` present in the query were needed for further extraction of features. The remaining features from table 4.1 were created using Python during the analysis. Construction of the subset of features used in the final solution in SPL form is visible as a part of the final searches in appendix C.

4.2.1.2 Dataset

To prepare the dataset, the search 4.1 was used to extract relevant log data from 12 weeks. After filtering, aggregating, and splitting into 1-day bins, the resulting dataset contained 11923 records. This data captured only legitimate activity of an AD domain and was presumed not to include any traces of Password Spraying.

As the next step, events representing malicious activity were generated. A variety of possible attack scenarios was covered by creating different sequences of events, anticipating both simple and more advanced attacks.

Feature	Description
cnt	Total count of events
computer_dc	Distinct count of authenticating computers
evt_4625_cnt	Count of events 4625
evt_4771_cnt	Count of events 4771
tdelta_sum	Sum of all time differences
tdelta_avg	Average time difference
tdelta_max	Maximal time difference
tdelta_min	Minimal time difference
tdelta_mod	Time difference recurring most often (mode)
tdelta_mod_[cnt|rat]	Count (ratio) of occurrences of the mode time difference
user_dc	Distinct count of users
user_max	Maximum count of failures per user
user_avg	Average count of failures per user
user_mod	Mode count of failures per user
user_mod_[cnt|rat]	Count (ratio) of the occurrences of mode failures per user
user_pers_[dc|rat]	Distinct count (ratio) of personal accounts
user_np_[dc|rat]	Distinct count (ratio) of non-personal accounts
user_sys_[dc|rat]	Distinct count (ratio) of system accounts
user_other_[dc|rat]	Distinct count (ratio) of other accounts

Table 4.1: Password Spraying: ML features considered

The following elements were taken into consideration:

- one or more different DCs or servers processing the authentication requests and logging the relevant events,
- activity logged in event ID 4625 or event ID 4771,
- composition of target account types (personal, non-personal, ...),
- number of passwords tried per user,
- overall count of accounts attempted,
- time difference between the attempts.

The research of attack tools in section 2.1 implied that using most of the tools will result in event 4625 being logged. However, some tools are capable of utilizing Kerberos authentication, resulting in event 4771 being logged. Adversaries may take advantage of this, as fewer organizations may monitor for events 4771 in the context of Password Spraying [35]. An attacker may hypothetically even combine both methods to lower the probability of being detected. Figure 4.4 illustrates composition of event IDs in the collected and generated datasets.

```
source=XmlWinEventLog:Security ((EventID=4625
    SubStatus=0xC000006A) OR (EventID=4771 Status=0x18))
| eval orig_time=_time
| eval src=if(cidrmatch("0.0.0.0/0", IpAddress), IpAddress,
    WorkstationName)
| rename TargetUserName AS user
| bin _time span=1d
| eventstats count AS cnt BY user, src, _time
| streamstats count AS event_no
| stats values(eval(cnt."-".user)) AS user_cnt,
        values(eval(orig_time."-".event_no)) AS orig_time,
        count AS cnt, dc(user) AS user_dc,
        dc(Computer) AS computer_dc,
        count(eval(EventID="4625")) AS evt_4625_cnt,
        count(eval(EventID="4771")) AS evt_4771_cnt
        BY _time, src
| where user_dc > 1
```

Listing 4.1: Password Spraying: Search for data retrieval

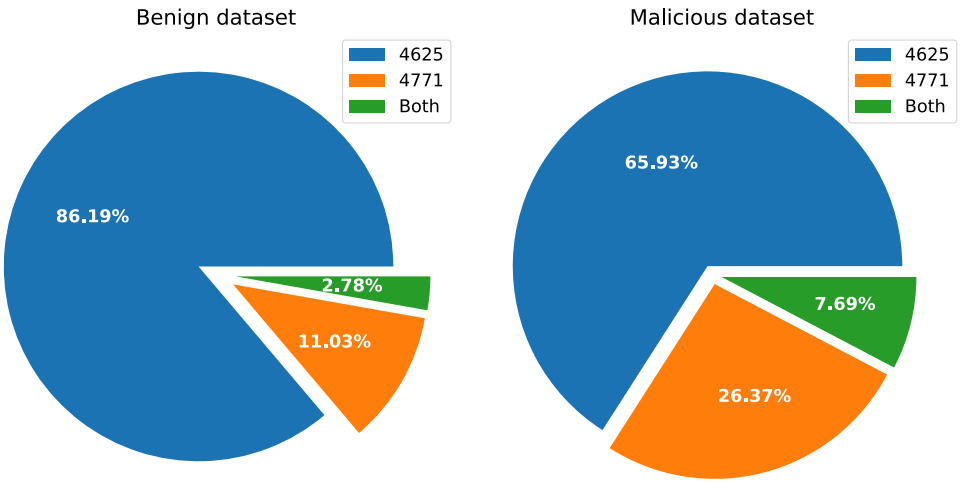

Figure 4.4: Representation of event IDs in the datasets

Other important differences between the benign and malicious datasets are noticeable in the count of targeted user accounts. From figure 4.5 it is visible that in the malicious dataset, there are records where hundreds of distinct user accounts are attempted from a single source. These cover the situations where an attacker would sweep all or a large group of accounts in the environment.

Following the principles illustrated in figure 4.2, the dataset was split into subsets

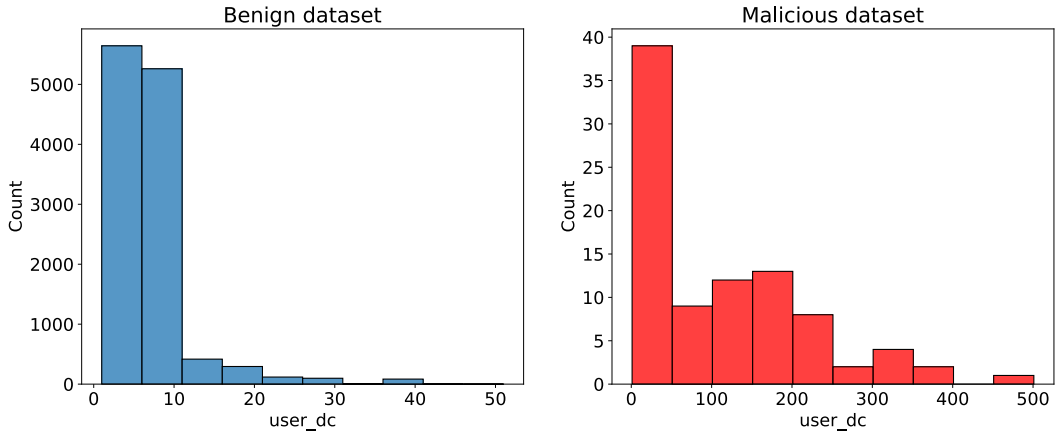

Figure 4.5: Distinct count of targeted user accounts

required for the modeling phase. Initially, the available data was split into learning and test subsets, with the ratio of 2 : 1 in size. The test subset was put aside and not used until the final comparison of the created models. Shuffling was applied to the data during the splitting process to ensure that malicious samples are distributed evenly. Results of the split are visible in table 4.2.

For unsupervised methods, the learning dataset was used as-is. For supervised and semi-supervised methods, it was further split into training and validation subsets. The size of the validation subset was determined to a third of the learning dataset. For the semi-supervised methods, any samples representing malicious data were deleted from the training subset, as novelty detection requires the data not to be polluted.

	Malicious samples	
Dataset	Count	Percent
Learning	59	0.73%
Test	32	0.81%

Table 4.2: Password Spraying: Distribution of malicious samples

More visualizations of the properties of the used datasets are visible in the analysis report attached on the enclosed CD.

4.2.2 Modeling

The previous section outlined some differences between normal and malicious activities observable in the datasets. Machine learning methods could be able to identify these differences and thus detect events related to Password Spraying attack.

The prepared datasets were used to conduct experiments with the ML algorithms listed in table 3.1. The goal was to identify the best ML model, its configuration, and the most descriptive set of features for detecting the attack.

4.2.2.1 Feature Selection

The purpose of feature selection was to determine the best subset from all the features listed in table 4.1. Since several features were created based on the same attributes extracted from Windows events, there was a possibility that some of them might have been correlated or redundant, not adding much valuable information to the models.

It is inevitable that a sequence of SPL commands performing feature extraction will be present in the final detection rule. As the query will be searching hundreds or thousands of events, it is desirable to design it effectively. Computing numerous features may be unnecessarily resource-intensive.

As already mentioned, a wrapper method was utilized for feature selection. Not every possible subset was considered, but several prototypical feature vectors were created instead, based on various statistical and "logical" properties. The created feature vectors are presented below, in descending order based on count of features contained in them:

- `all` – all features that were initially created (as listed in table 4.1),
- `no_user_type` – all features, except for those related to different account types,
- `not_correlated` – highly correlated features removed,
- `best_10` – the best 10 features selected based on χ^2 statistic test,
- `important` – important features from the perspective of the attack principles,
- `cnt_only` – containing only `cnt` and `user_dc` attributes.

Statistical approach was used to create feature vectors `not_correlated` and `best_10`. Highly correlated features were determined by computing the Pearson correlation coefficient for each feature pair. Then filtering was applied based on the threshold for an absolute value of the coefficient. A feature from the pair was removed if the value was higher than 0.7. For the vector `best_10`, χ^2 statistic was computed between each feature and the target class. Based on the score, ten best features were selected. This method should have removed features that were the most likely to be independent of the target class and therefore irrelevant for classification.

The feature vector `no_user_type` covers a situation where differentiating between account types is not possible. This may be the case if there is no standardized naming convention in use, or the naming does not indicate any useful information about the type of accounts.

In the description of Password Spraying in section 2.1, several indicators of this attack were mentioned. The feature vector `important` was created to include only a few features considered the most important based on these indicators. The vector `cnt_only` contained only two basic features that would be likely used in a traditional threshold rule and was included for the purpose of comparison. Table 4.3 lists the features used in some of the feature vectors; the full list of features included in each particular vector is provided in the analysis report attached on the enclosed CD.

Feature / Vector	cnt_only	important	best_10
cnt	•	•	•
user_dc	•	•	•
user_mod_cnt		•	•
user_avg		•	
user_pers_dc			•
tdelta_mod_cnt		•	•
tdelta_avg			•
tdelta_sum			•
tdelta_max			•
evt_4625_cnt			•
evt_4771_cnt			•

Table 4.3: Password Spraying: Feature vectors

The wrapper method for feature selection was utilized as follows:

1. Default values of hyperparameters were determined for each of the ML algorithms used (as listed in table 3.1).
2. Each of the algorithms was fitted on the training dataset, using every feature vector.
3. Scoring was performed on the validation dataset.
4. Best vector was selected for every algorithm according to achieved F_2-score.

Figure 4.6 illustrates the obtained F_2-score values after applying different feature vectors to all the selected ML algorithms. Particular score values were not decisive at this point; the results were only used to decide which feature vectors the algorithms would utilize in further experiments. If multiple vectors achieved a similar score, computational complexity of the feature extraction was considered. Following choices of feature vectors were made according to the results:

- `important` for Isolation Forest, One-class SVM and Support Vector Classifier,
- `best_10` for Random Forest Classifier and both Local Outlier Factor variants.

4.2.2.2 Hyperparameter Tuning

In the hyperparameter tuning step, the optimal values for hyperparameters of all the selected algorithms were determined. A parameter grid, representing a subset of the hyperparameter space, was manually specified for each of the algorithms. This space was then exhaustively searched in a grid search, meaning that every possible combination of hyperparameters was generated. A model was fitted and evaluated for every combination.

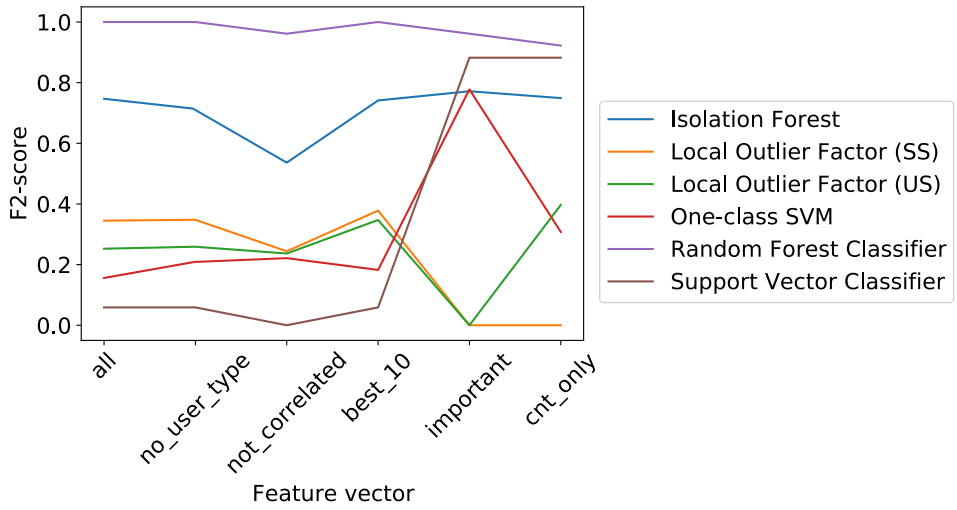

Figure 4.6: Password Spraying: F_2-score of feature vectors

Similarly as with feature selection, scoring was performed on the validation dataset, and F_2-score was used for evaluating and selecting the best configuration of every algorithm.

To make the comparison possible, the Threshold Rules detecting Password Spraying, presented in section 2.1, were adapted to the format used in the datasets. Using events 4625 instead of 4776 and aggregating the events per source made it possible to merge the rules into a single one. The adapted version of the rule is visible in listing 4.2.

```
source=XmlWinEventLog:Security ((EventID=4625
    SubStatus=0xC000006A) OR (EventID=4771 Status=0x18))
| rename TargetUserName AS user
| bin _time span=1d
| stats count AS cnt, dc(user) AS user_dc BY _time, src
| where user_dc > 1 AND cnt > 40
```

Listing 4.2: Adapted Threshold Rule detecting Password Spraying

As every record in the dataset represented an aggregation of failed authentication attempts throughout one day, the numeric threshold values for filtering on the number of these attempts (`cnt`), and distinct count of targeted users (`user_dc`), must have been changed accordingly. In a sense, different possible threshold values used with these attributes could also be considered "hyperparameters" of the Threshold Rule model. Therefore, the rule could participate in a similar process as unsupervised ML algorithms: the hyperparameters were tuned on the learning dataset, and the best values were selected according to F_2-score metric. The rule with chosen values was then be applied to the test dataset and compared with the other models. The parameter grid for Threshold Rule contained two fields, `cnt` and `user_dc`, with two sets of possible threshold values for them.

Table 4.4 displays values for hyperparameters of different models selected as the best in the grid search. These were subsequently used during the evaluation of the models on the test dataset. Hyperparameters that are not listed were used with their default values according to the *Scikit-learn* implementation. The contamination parameter for LOF and Isolation Forest models, as well as the ν parameter for One-class SVM, were unified to the value of 0.01, expecting approximately 1% of malicious samples in the test dataset. Full parameter grids used for hyperparameter tuning are provided in the analysis reports attached on the enclosed CD.

Model	Hyperparameter	Value
Threshold Rule	`user_dc`	1
	`cnt`	40
Isolation Forest	`n_estimators`	20
	`max_features`	0.8
	`contamination`	0.01
Local Outlier Factor (unsupervised)	`n_neighbors`	50
	`leaf_size`	10
	`p`	1
	`contamination`	0.01
One-class SVM	`kernel`	rbf
	`gamma`	0.0075
	`nu`	0.01
Local Outlier Factor (semi-supervised)	`n_neighbors`	20
	`leaf_size`	30
	`p`	1
	`contamination`	0.01
Support Vector Classifier	`gamma`	0.0075
	`C`	1.0
Random Forest Classifier	`n_estimators`	10
	`max_features`	0.8

Table 4.4: Password Spraying: Hyperparameters of ML models

4.2.3 Evaluation

The best feature vectors and the optimal hyperparameter values selected for each algorithm were used to create ML models. Supervised and semi-supervised models were fitted on the training dataset. All the models were then applied to the test dataset, where their performance was measured. Comparison of the obtained results is displayed in table 4.5.

The test dataset contained 32 malicious events out of 3965 total events. The Threshold Rule was able to identify 23 attacks correctly and yielded 12 false alarms. The number of FP detections is not too high, considering the test dataset covers traffic of several weeks.

	Thresh. Rule	Isolation Forest	LOF (uns.)	OC SVM	LOF (semi.)	SVC	RFC
TP	*23*	25	17	**29**	19	25	**29**
FP	*12*	**8**	23	39	33	**0**	**0**
FN	*9*	7	15	**3**	13	7	**3**
TN	*3921*	3925	3910	3894	3900	**3933**	**3933**
Precision	*0.657*	0.758	0.425	0.426	0.365	**1.000**	**1.000**
Recall	*0.719*	0.781	0.531	**0.906**	0.594	0.781	**0.906**
F_1**-score**	*0.687*	0.769	0.472	0.580	0.452	0.877	**0.951**
F_2**-score**	*0.706*	0.776	0.506	0.740	0.528	0.817	**0.924**

Table 4.5: Password Spraying: Comparison of ML algorithms

However, the rule missed 9 actual attacks, which could potentially mean compromised user accounts.

Isolation Forest performed slightly better than the Threshold Rule in all the metrics. It was able to detect more attacks and at the same time reduce the number of FPs. LOF models performed worse in terms of both error types. One-class SVM obtained the highest recall score, meaning that it was able to identify most of the attacks, with only 3 false negatives, however, at the cost of the highest number of FP detections among all the methods.

Supervised methods resulted in no false positives. Support Vector Classifier identified two attacks more than the Threshold Rule. Random Forest Classifier appears to be the best method among all, with the highest score values of all measured metrics and no FPs.

Considering different ML approaches, semi-supervised methods had significantly more false alarms than other models, whereas supervised methods had none. By comparing ML algorithm types, tree-based algorithms (Isolation Forest and RFC) performed well in overall and seem to be suitable for this task. On the contrary, both LOF methods had high number of incorrect predictions and do not appear to be very suitable.

4.2.4 Deployment

Based on the obtained results, two approaches for detecting Password Spraying were implemented in Splunk, utilizing Isolation Forest and Random Forest Classifier algorithms. Both these methods performed better in comparison to the Threshold Rule.

These two ML algorithms are unlike in nature. Not only do they represent different ML approaches, but they were also used with different feature vectors in the experiments. As every AD environment is individual, there may be constraints favoring one solution over the other in practice.

Isolation Forest is an unsupervised method and thus does not require a model to be trained and saved. The algorithm is applied directly to new data. In experiments, this algorithm was used with `important` feature vector that contained five features only and did not require distinction of different user types. To deploy this approach, it is necessary

to use the data preparation search provided in listing C.1 in conjunction with the snippet in listing 4.3, performing outlier detection based on the Isolation Forest algorithm. A new field is created in the search results, `isOutlier`, marking events evaluated as outliers. These are the malicious events for which an alarm should be created.

```
1 ...
2 | fit IsolationForest cnt, user_dc, user_mod_cnt, user_avg,
    tdelta_mod_cnt n_estimators=20 max_features=0.8
    contamination=0.01
3 | search isOutlier = 1
```

Listing 4.3: Password Spraying: Detection with Isolation Forest

Random Forest Classifier was used with the feature vector `best_10` that contains more features. Therefore, it requires an extended data preparation search, which is provided in listing C.2. Furthermore, RFC is a supervised method, and therefore must be treated differently. At first, a model must be trained on training data, which can be achieved by running the search in listing 4.4. Note that this search requires a labeled dataset, with the target field `isMalicious` set appropriately for every record. The value `isMalicious=-1` is used to represent benign events, whereas `isMalicious=1` represents attacks. The trained model is saved as `PS_RFC_model`.

```
1 ...
2 | fit RandomForestClassifier isMalicious
3    FROM cnt, user_dc, user_mod_cnt, user_pers_dc,
    tdelta_mod_cnt, tdelta_avg, tdelta_sum, tdelta_max,
    evt_4625_cnt, evt_4771_cnt
4    INTO PS_RFC_model n_estimators=10 max_features=0.8
```

Listing 4.4: Password Spraying: Training RFC model

The next step is to apply this model to new data. This is done by the search presented in listing 4.5. Both searches must be used together with the data preparation search provided in listing C.2. After the model is applied, the predicted value is contained in the field `predicted(isMalicious)` that depicts whether an event is predicted to be malicious.

```
1 ...
2 | apply PS_RFC_model
3 | search "predicted(isMalicious)" = 1
```

Listing 4.5: Password Spraying: Detection with RFC model

To deploy the developed searches, relevant Windows events must be ingested and correctly parsed by a Splunk instance. It is necessary to collect events 4771 and 4625

from all domain controllers, and events 4625 also from any other member servers that may perform NTLM pass-through authentication.

To use the developed searches for real-time detection of Password Spraying, it is necessary to run them by an appropriate schedule. Optimal schedules may be individual for every AD environment; however, several factors should be noted.

It is important to select a proper timespan of covered events. The unsupervised search based on Isolation Forest must be supplied with enough events to grasp the normal activity of the environment. Recommended timespan is no shorter than one week. The search based on Random Forest Classifier requires enough data (both benign and malicious) to be trained on. However, after the model is trained, the timespan requirement for its usage is not as strict as for Isolation Forest.

4.3 Kerberoasting

Kerberoasting attack is challenging to detect, as it does not significantly differ from usual domain activity from the perspective of logged events. Its detection can be based on identifying irregular patterns of activity, which is, however, not easy to capture in the form of conditions in a detection rule. Although in section 2.2 there were mentioned several non-traditional detection techniques, their implementation may not be feasible in every AD environment.

In our recently published research, we proposed applying ML techniques to address the issues of signature-based approaches and improve their capabilities for detecting Kerberoasting [89]. We utilized two algorithms for anomaly detection – Local Outlier Factor and One-class SVM. In our testing, the method based on One-class SVM algorithm was effective, as it was able to reduce the number of FP detections, and at the same time also to decrease the number of false negatives in comparison with a threshold rule.

Continuing this research, I have decided to further explore ML approach for Kerberoasting detection by utilizing more algorithms, extending the experiments, and evaluating the performance on more log data. In this thesis, I have included supervised ML methods, which were previously not considered, additional feature vectors, and more samples of malicious events, covering multiple different attack scenarios.

4.3.1 Data Preparation

Kerberos service ticket requests are audited in events *4769: A Kerberos service ticket was requested* on domain controllers. These events generate in high amounts, as creating of SGTs is a very frequent occurrence in an AD environment. Therefore, it is necessary to narrow the event selection to service tickets issued using weaker cryptographic suites. This is possible by filtering the events on `TicketEncryptionType` attribute to values containing DES encryption (`0x1`, `0x3`) and RC4 encryption (`0x17`, `0x18`). Other possible codes are listed in table 2.4. This filtering would remove most of the events, as most traffic should default to AES encryption. However, the number of events may still be significant, especially in environments where RC4-based cipher suites are not disabled.

After selecting the events, it is necessary to extract information interesting for Kerberoasting detection. Event 4769 records network and account information identifying the source requesting the service ticket and the target service for which a ticket was requested. The format and possible values contained in the events may vary, and its understanding is essential for correct filtering and extraction of features for ML models.

Network information is present in the `IpAddress` field that contains an IPv6 or IPv4 address of the host making the request. This field may also contain the value `::1` representing *localhost* [54]. Events containing this value can be filtered out, as it effectively means that the service ticket request was made by DC itself. It does not make sense to execute Kerberoasting for an attacker having access to a DC already.

Information about the user account requesting the service ticket is logged in the `TargetUserName` field. This attribute contains the account name that requested the ticket, in the User Principal Name (UPN) syntax: `user_name@FQDN`. Although this field is in the UPN format, it does not contain the value of `UserPrincipalName` AD attribute, but it is built from the `SamAccountName` attribute of the user account and the domain name instead. [54]

Service information can be found in the attributes `ServiceName` and `ServiceSid`. `ServiceName` contains the name of the account or computer for which the service ticket was requested. It is the name of an account registered with an SPN, not the SPN value itself. SID value of this account will be typically present in the `ServiceSid` attribute. [54]

As explained in section 2.2, adversaries would be interested in obtaining service tickets for SPNs registered with manually created user accounts, as those may have a weak password, and the service tickets will likely be issued using RC4-based encryption types. Events capturing ticket requests targeting services mapped to computer accounts (ending with `$` character) can be excluded from the search. This was implemented using `regex` SPL command, as visible in listing 4.6.

In the proposed approach, ML techniques are utilized to differentiate between usual and unusual behavior. To assess the behavior of a particular source, an observation window of one day was determined. After filtering, the events 4769 were aggregated by their source and the period of one day. Consequently, one data row represented the service ticket requests made by a unique combination of username and IP address per one day.

The one-day period was preferred over shorter periods (i.e., several hours) due to the broader scope it can represent. The number of requests may significantly differ over the day (business vs. out-of-business hours). Also, an attacker may request fewer tickets at a time, prolonging the activity over more extended time instead of requesting a bunch of tickets at once.

The Splunk query used to retrieve events 4769 and prepare the dataset is listed in listing 4.6. This query performs filtering and aggregation of the events per source, as mentioned above. Apart from that, it also prepares the basis for feature extraction described in the following subsection.

At the end of the search presented in listing 4.6, there is a condition `svc_dc > 2`, filtering events for those having distinct service count higher than two. This was used to reduce further the number of data rows considered, excluding sources requesting only one

```
source=XmlWinEventLog:Security EventID=4769
    TicketEncryptionType IN (0x1, 0x3, 0x17, 0x18)
| eval orig_time=_time
| search IpAddress != "::1"
| regex ServiceName != "\$$"
| bin _time span=1d
| streamstats count AS event_no
| stats values(eval(orig_time."-".event_no)) AS orig_time,
        values(ServiceName) AS ServiceName,
        dc(ServiceName) AS svc_dc, count AS cnt,
        dc(Computer) AS dc_dc
        BY _time, IpAddress, TargetUserName
| where svc_dc > 2
```

Listing 4.6: Kerberoasting: Search for data retrieval

or two distinct service tickets per day. For the Kerberoasting attack, more requests made by an attacker during the day are expected.

4.3.1.1 Feature Engineering

The main focus of the feature engineering phase was to extract features that could describe the (ab)normality of the logged activity. From the perspective of Kerberoasting detection, the primary indicators are weak encryption types (already filtered), count of requested service tickets, types of services involved, and types of user accounts making the requests. The number of features that may be extracted is vastly limited by the attributes contained in the event 4769.

The count of requested services was represented in three different features: `cnt` containing the total count of service requests from one source, `svc_dc` containing the count of distinct services, and `svc_cnt_rat` representing the ratio between those two. This feature aims to capture the following idea. For an attacker, it is sufficient to request a single ticket for each service. In contrast, a legitimate source can request multiple tickets towards the same service over time, especially if *Maximum lifetime for service ticket* policy setting is set to lower values [34].

To define normal behavior, it is necessary to determine which sources typically access which services. Events 4769 include the name of a user account and the account mapped to the requested service. The number of possible values in these attributes may be high, especially in larger AD environments with many users. Considering every account separately would lead to a high-dimensional dataset. Also, encoding of account names would be difficult to implement, as the number of possible values is not necessarily finite and may change over time.

Instead of treating every account separately, it might be beneficial to distinguish between various types of accounts. The process of defining these types depends entirely on

conventions used in a particular AD environment. Furthermore, differentiation between the types is impossible without external information about the accounts.

Similarly as with Password Spraying detection, the most straightforward approach is to benefit from naming convention, if utilized, and if its definition allows obtaining information about accounts from their names. For data used in this thesis, it was possible to use this approach for both source user accounts and service accounts.

For user account name, it was possible to determine whether the account is assigned to a person (`user_is_pers`), is a non-personal account, used by some application, script, or scheduled task (`user_is_np`), or is a computer account (`user_is_sys`). These features were binary, containing values 0 or 1.

Regarding services, it was possible to identify two big groups of service accounts that provided the type of service associated with them based on their names: database-related and application-related services. For each data point, it was calculated how many tickets of each service type were requested. If the requested service did not fall under any of these categories, it was marked as "other". Features `svc_app_dc`, `svc_sql_dc` and `svc_other_dc` represent distinct count of requested services of each type. These features were also included in the form of ratio to the total count (`svc_dc`).

Another attribute representing the source host is `IpAddress`. Even though an IP address may be understood as a binary or decimal number, it does not represent an ordinal value. A typical approach of expressing IP addresses is obtaining location information and representing it as GPS coordinates or a regional code. However, this is not applicable in the Kerberoasting scenario, as the IP addresses in events 4769 are expected to be from private IP ranges.

The space of possible values in `IpAddress` field is significant. This is a similar situation as with account names, where an approach would be to rely on conventions of the environment. In case there is systematic network segmentation in place, where different IP subnets are assigned to different logical network segments, IP addresses can be labeled based on their segment. This would create a finite number of groups that may reveal valuable information about the type of source computer. However, this information was not available for the data used in this thesis, and thus this approach was not tested.

Additionally, several features that reflect time differences between the logged events were created as `tdelta_*` fields. Those might help identify attacks that span an extended period, where an attacker requests one or few service tickets at a time.

Ultimately, table 4.6 lists all 20 features created for detecting Kerberoasting. Many features listed in this table represent user or service types. To extract these features from account names, the naming convention was interpreted in the form of regular expressions. These were used in the Jupyter notebooks during the analysis, as well as in the final SPL searches provided in appendix C.

4.3.1.2 Dataset

The dataset for ML analysis was generated using Splunk search provided in listing 4.6, which was used to collect events 4769 from DCs in an AD domain for the time of six weeks. The search performed necessary event selection, filtration, and aggregation as described

Feature	Description
cnt	Count of service requests
svc_dc	Distinct count of requested services
svc_cnt_rat	Ratio of requests count to distinct service count
dc_dc	Distinct count of domain controllers
svc_app_dc	Distinct count of application services
svc_sql_dc	Distinct count of database services
svc_other_dc	Distinct count of other services
svc_app_rat	Ratio of application services
svc_sql_rat	Ratio of database services
svc_other_rat	Ratio of other services
user_is_pers	Is user account personal?
user_is_np	Is user account non-personal?
user_is_sys	Is user account system?
tdelta_sum	Sum of all time differences
tdelta_avg	Average time difference
tdelta_max	Maximal time difference
tdelta_min	Minimal time difference
tdelta_mod	Time difference recurring most often
tdelta_mod_cnt	Number of occurrences of the most recurring time difference
tdelta_mod_rat	Ratio of the most recurring time difference to all differences

Table 4.6: Kerberoasting: ML features considered

above, resulting in 9398 data rows. Each row represented service ticket requests made by a particular source during one day. The events in this dataset were considered benign, expressing normal activity in the domain.

Next, events representing malicious activity were generated. As described in section 2.2, Kerberoasting attack execution usually results in events 4769 being logged with `TicketEncryptionType=0x17`. This value was fixed in the generated events; other attributes were designed to cover different possible attack scenarios. The following factors were considered in the generating process:

- the number of service tickets requested,
- one or more DCs responding to the requests and logging the events,
- types of user accounts making the requests,
- types of services for which tickets were requested,
- time difference between the requests.

In the "noisiest" variant of Kerberoasting, an attacker would request service tickets for all, or a large subset of SPNs available in the domain, increasing the probability

that at least one service ticket would be cracked successfully. In that case, numerous requests for different services from a single source are expected. However, more advanced attackers may target only a specific group of SPNs, e.g., service accounts registered with user accounts or accounts that are members of high-privileged groups. Therefore, it is also expected to observe attacks where a lower number of tickets is requested. Figure 4.7 illustrates that both these scenarios are present in the generated dataset and compares the request counts to the benign dataset.

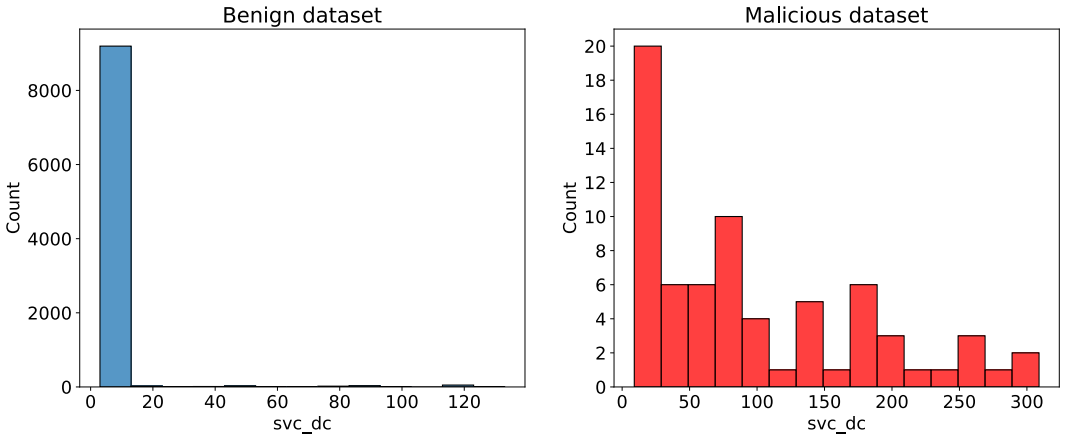

Figure 4.7: Count of requested services in the datasets

In the analysis, three categories of services were considered: database, application, and other. The distribution of these types in the generated malicious events was similar to that in the legitimate dataset, slightly biased towards database and application services. There is a higher probability that service accounts of database and application services were registered with a user account.

The type of an account making the requests was the next property considered. To request a service ticket, an attacker must hold a valid domain account. This would be most probably a personal account that was previously compromised. Another option is that an attacker could exploit a vulnerability in a script or application running under a non-personal account, and can request service tickets on behalf of this account. Situations in which an attacker could request service tickets under a computer account were not considered. Representation of user types in the datasets is expressed in figure 4.8.

Following the same principles as during the analysis of Password Spraying, the dataset was split into learning and test subset, leaving one-third of the available data for evaluation. The learning set was further split in the same ratio 2 : 1 into training and validation subsets for use with supervised and semi-supervised methods. Malicious samples were evenly spread across the datasets, which is confirmed by table 4.7.

More visualizations of the dataset properties are available in the analysis report attached on the enclosed CD.

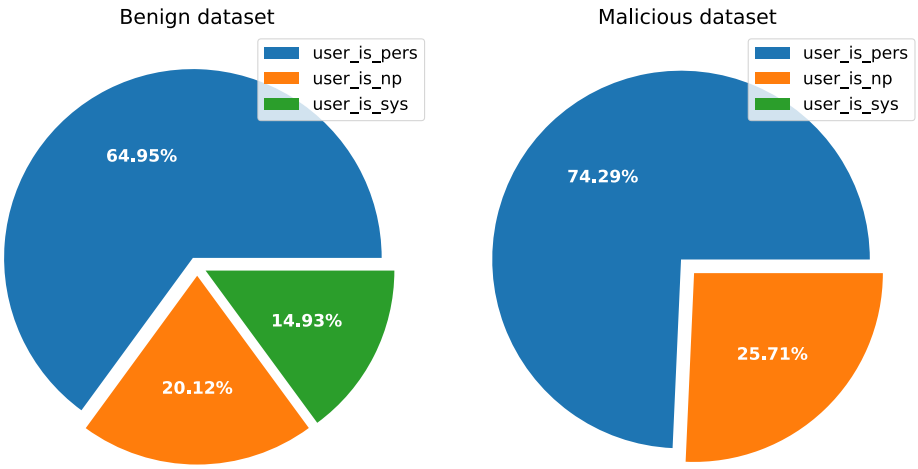

Figure 4.8: Representation of user types in the datasets

	Malicious samples	
Dataset	Count	Percent
Learning	48	0.76%
Test	22	0.70%

Table 4.7: Kerberoasting: Distribution of malicious samples

4.3.2 Modeling

ML algorithms listed in table 3.1 were applied to the prepared datasets. Extensive experiments were performed to find the most suitable set of features and configuration of hyperparameters for each of the models. The goal was to identify which ML algorithm is more suitable for detection of Kerberoasting attack, and how they would perform in comparison to the Threshold Rule.

4.3.2.1 Feature Selection

The list of all extracted features presented in table 4.6 is not very long, moreover, many of the features are based on similar concepts. In this scenario, a wrapper method for feature selection was utilized to compare impact of different subsets of features, formed based on their properties. Following sets were created:

- `C` (count): `svc_dc`,
- `CR` (count ratio): `svc_cnt_rat`,
- `S` (services): `svc_app_dc`, `svc_sql_dc`, `svc_other_dc`,
- `SR` (services ratio): `svc_app_rat`, `svc_sql_rat`, `svc_other_rat`,

- `U` (user): `user_is_pers`, `user_is_np`, `user_is_sys`,
- `T` (time): `tdelta_avg`, `tdelta_mod_cnt`, `tdelta_sum`.

Further, these feature subsets were unionized to form more extensive feature vectors. Every possible combination of the set was not considered, as varying importance of particular features for the given problem was taken into account. For example, set `C` was present in all feature vectors, as a distinct count of services is a principal indicator for detecting Kerberoasting.

Apart from the method described above, a statistical approach was also applied to feature selection. Correlation heatmap calculated using Pearson correlation coefficient (available in the report attached on the enclosed CD) has shown that there were not many highly correlated features. Therefore, filtration based on correlation was not applied. However, selecting ten best features based on scores computed by χ^2 statistic test was performed, and the selected features were used as the feature vector `best_10`.

Once the feature vectors had been formed, the wrapper method was applied as follows:

1. Default values of hyperparameters were determined for each of the ML algorithms.
2. Each of the algorithms was fitted on the training dataset, using every feature vector.
3. Scoring was performed on the validation dataset.
4. Best vector was selected for every algorithm according to achieved F_2-score.

Figure 4.9 illustrates the obtained F_2-score values after applying different feature vectors to all the selected ML algorithms. Feature sets `CR` and `SR` were present only in several vectors, as those did not seem to improve the score in general. This can be noted by comparing `C + SR` to `C + S`, and `C + CR` to `C`. Also, the performance of feature vector `best_10` was relatively poor for all algorithms in comparison to other feature vectors.

Feature vector `C + S + U`, taking into account both user and service types appears to be the most suitable for One-class SVM and SVC methods. However, the performance seems to degrade if combined with the features representing time differences. Generally, the addition of time-based features does not appear to improve the performance of the models.

Interestingly, both LOF methods obtained the highest F_2-score when used with `C` subset only, containing a single feature `svc_dc`. Especially for the semi-supervised version, the difference is significant in comparison to other feature vectors.

Table 4.8 lists the feature vectors used with different algorithms in further experiments. The choice was made based on the gained F_2-score illustrated in figure 4.9. In situations where multiple vectors achieved similar score, the selection was based on implementation overhead that the required feature extractions would induce.

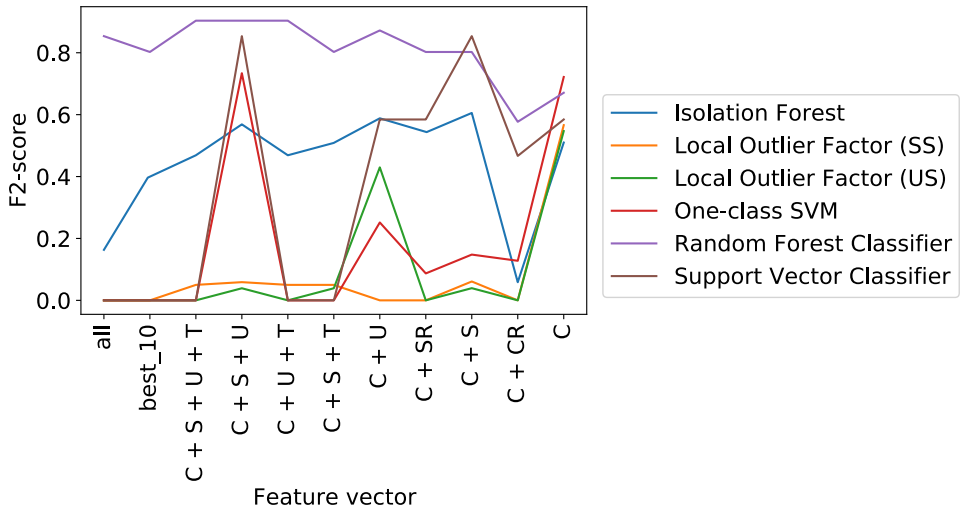

Figure 4.9: Kerberoasting: F_2-score of feature vectors

Algorithm	Feature vector
Isolation Forest	`C + S`
LOF (unsupervised)	`C`
One-class SVM	`C + S + U`
LOF (semi-supervised)	`C`
Support Vector Classifier	`C + S + U`
Random Forest Classifier	`C + S + U`

Table 4.8: Kerberoasting: Feature vectors

4.3.2.2 Hyperparameter Tuning

Once the most suitable feature vector had been selected for each of the algorithms, grid search was performed to find optimal values of hyperparameters for every model. Scoring was performed on the validation dataset, and F_2-score was used for evaluating and selecting the best configuration of every algorithm.

The Threshold Rule for detecting the Kerberoasting attack presented in listing 2.3 was modified to match the format of fields used in this analysis. Similarly as in the Password Spraying scenario, the rule was involved in the hyperparameter tuning process. Different possible threshold values for filtering the count of distinct services requested (`svc_dc`) were considered as its "hyperparameter" values. The parameter grid for this field contained values ranging from 5 to 50, with an increment of 5. The adapted version of the Threshold Rule for Kerberoasting detection is presented in listing 4.7. The rule triggers an alarm for sources that request more than ten distinct services during one day.

Table 4.9 provides values for hyperparameters selected as the optimal and subsequently used for evaluation on the test dataset. Hyperparameters that are not listed were used

```
source=XmlWinEventLog:Security EventID=4769
    TicketEncryptionType IN (0x1, 0x3, 0x17, 0x18)
| search IpAddress != "::1"
| regex ServiceName != "\$$"
| bin _time span=1d
| stats dc(ServiceName) AS svc_dc
        BY _time, IpAddress, TargetUserName
| where svc_dc > 10
```

Listing 4.7: Adapted Threshold Rule detecting Kerberoasting

with their default values according to their *Scikit-learn* implementation. Values of the contamination parameter for LOF and Isolation Forest models, and the ν parameter for One-class SVM were unified to 0.01, expecting approximately 1% of malicious samples in the test dataset. Full parameter grids that were used for hyperparameter tuning are provided in the report attached on the enclosed CD.

Model	Hyperparameter	Value
Threshold Rule	`svc_dc`	10
Isolation Forest	`n_estimators`	50
	`max_features`	1.0
	`contamination`	0.01
Local Outlier Factor (unsupervised)	`n_neighbors`	20
	`leaf_size`	20
	`p`	1
	`contamination`	0.01
One-class SVM	`kernel`	rbf
	`gamma`	0.01
	`nu`	0.01
Local Outlier Factor (semi-supervised)	`n_neighbors`	10
	`leaf_size`	10
	`p`	1
	`contamination`	0.01
Support Vector Classifier	`gamma`	0.005
	`C`	0.75
Random Forest Classifier	`n_estimators`	20
	`max_features`	0.8

Table 4.9: Kerberoasting: Hyperparameters of ML models

4.3.3 Evaluation

The optimized ML models were applied to the test dataset, where their performance was evaluated and compared. The test dataset that was put aside earlier in the data preparation phase contained 3125 data rows, out of which 22 represented attacks. Comparison of the achieved results is presented in table 4.10.

	Thresh. Rule	**Isolation Forest**	**LOF (uns.)**	**OC SVM**	**LOF (semi.)**	**SVC**	**RFC**
TP	*20*	10	7	**21**	16	13	18
FP	*73*	18	24	48	40	**0**	**0**
FN	*2*	12	15	**1**	6	9	4
TN	*3030*	3085	3079	3055	3063	**3103**	**3103**
Precision	*0.215*	0.357	0.226	0.304	0.286	**1.000**	**1.000**
Recall	*0.909*	0.455	0.318	**0.955**	0.727	0.591	0.818
$\boldsymbol{F_1}$**-score**	*0.348*	0.400	0.264	0.462	0.410	0.743	**0.900**
$\boldsymbol{F_2}$**-score**	*0.552*	0.431	0.294	0.669	0.556	0.644	**0.849**

Table 4.10: Kerberoasting: Comparison of ML algorithms

As visible from the table 4.9, the threshold used in the static rule was set relatively low, only to 10 distinct services. Thanks to this, the rule missed only two attacks. On the other hand, it also resulted in 73 false alarms, which is quite a high number, considering that the test dataset encompasses domain activity of approximately two weeks only. In terms of false positives, all ML approaches emitted less FP detections than the Threshold Rule. However, at the same time, most of them also missed more attacks.

The One-class SVM model was able to overcome the Threshold Rule in all markers, successfully detecting one more attack and concurrently decreasing the number of FP detections by more than 30%. The detection capabilities of this approach can be visible from the achieved recall, which was highest among all the algorithms.

Supervised algorithms had no FP detections and thus 100% precision. However, the number of false negatives was slightly higher, especially for SVC. Random Forest Classifier was the second best among the overall results. It obtained the best F-scores, though it missed two more attacks than the Threshold Rule. For other models, the measured ratio of missed attacks was more than 25%, which was considered too high for a reliable attack detection method, outweighing the advantage of fewer false alarms.

4.3.4 Deployment

Based on the obtained results, the One-class SVM and Random Forest Classifier methods were used to implement Splunk rules for detecting the Kerberoasting attack. In the performed analysis, both these algorithms were utilized with the feature vector `C + S + U` that consisted of seven features, including distinct count of services requested, their types, and type of user account involved. Most of these features rely on information

extracted from the naming convention used in the AD environment. To deploy these methods successfully, this information must be available or provided by other means.

Listing C.3 provides a preparation search necessary to extract the required features from the event data. The extractions are mostly implemented by `eval` commands, utilizing regular expressions to extract account types from their names. The actual regular expressions are replaced by `<REGEX>` placeholders, as those would be different for every AD environment. It is also important to note the UPN format of `TargetUserName` field in event 4769. This needs to be reflected in the design of the regular expression extracting the user type, or the domain suffix needs to be removed beforehand. The latter option was implemented in the developed search.

One-class SVM is a semi-supervised method that requires training of a model. However, this model should be trained on benign data only, and therefore this approach does not require prior generating of any attack data. This is advantageous for deployment, as such data might not be available.

The data preparation search, provided in listing C.3, must be used both during training and applying the One-class SVM model. Search snippet in listing 4.8 demonstrates training of the model. As visible, all necessary features are supplied to the algorithm[8], its hyperparameters are provided according to table 4.9, and the trained model is saved as `KRB_OCSVM_model`.

```
1 ...
2 | fit OneClassSVM svc_* user_* kernel="rbf" nu=0.01
    gamma=0.01 INTO KRB_OCSVM_model
```

Listing 4.8: Kerberoasting: Training One-class SVM model

Search snippet 4.9 shows how a saved model named `KRB_OCSVM_model` would be applied to detect Kerberoasting. The results are filtered for rows matching condition `isNormal = -1`, selecting rows evaluated as anomalous observations by the model. Note that the condition is different from the implementation of LOF and Isolation Forest algorithms that mark anomalies as `isOutlier = 1`.

```
1 ...
2 | apply KRB_OCSVM_model
3 | search isNormal = -1
```

Listing 4.9: Kerberoasting: Detection with One-class SVM model

On the other hand, the supervised approach based on Random Forest Classifier requires labeled data for training. This data must contain both benign and malicious samples. Labeling of the dataset can be done by using `eval isMalicious=-1` for benign and `eval isMalicious=1` for malicious samples. The search snippet for training the RFC model is

[8]The features are supplied to the model by wildcard notation due to their opportune naming.

provided in listing 4.10. This snippet should be appended to the data preparation search provided in listing C.3 and the related data labeling logic. The trained model is saved as `KRB_RFC_model`.

```
1 ...
2 | fit RandomForestClassifier isMalicious FROM svc_* user_*
    INTO KRB_RFC_model n_estimators=20 max_features=0.8
```

Listing 4.10: Kerberoasting: Training RFC model

The search snippet in listing 4.11 is used for detection with the saved RFC model. The results are filtered for those predicted as malicious.

```
1 ...
2 | apply KRB_RFC_model
3 | search "predicted(isMalicious)" = 1
```

Listing 4.11: Kerberoasting: Detection with RFC model

Both presented approaches fit and save a model. To deploy these methods for real-time detection of Kerberoasting, the models should be trained on data from an extended time span (several weeks) using searches 4.8 and 4.10. Once the models are trained and saved, the searches 4.9 and 4.11 should be scheduled for running once a day for detection.

Chapter 5

Results

This chapter presents the obtained results. Performance of the proposed ML solutions is compared to the traditional detection rules, and the advantages and disadvantages of each approach are discussed. Further, contributions of this thesis are summarized, and possible directions of future work are outlined.

5.1 Comparison

Table 5.1 recapitulates ML methods identified as the most suitable for detecting the selected attacks. Based on the results obtained on the test dataset, there does not appear to be a single superior method, and none of the learning types proved to be particularly outstanding. A noticeable difference was observed in terms of FP detections between supervised and semi-supervised algorithms. Semi-supervised methods produced more false alarms, whereas the supervised methods almost none. Variations in the results are present at the level of particular algorithms, emphasizing the importance of analyzing a concrete ML algorithm in relation to a particular detection scenario.

Scenario	Algorithm	Type
Password Spraying	Isolation Forest	unsupervised
	Random Forest	supervised
Kerberoasting	One-class SVM	semi-supervised
	Random Forest	supervised

Table 5.1: Implemented ML algorithms

Figure 5.1 presents results of Password Spraying detection. The Threshold Rule missed a significant number of attacks. The unsupervised method based on Isolation Forest had slightly better characteristics, as it detected two more malicious samples and reduced the number of FPs. Random Forest improved the detection significantly. False alarms were completely eliminated, and the number of recalled attacks increased by more than 25% in comparison with the Threshold Rule.

Figure 5.1: Password Spraying: Comparison of results

Figure 5.2 presents results of Kerberoasting detection. The main issue of the Threshold Rule was a significant number of false alarms. The best results were achieved by the One-class SVM algorithm, which was the only method that outperformed the Threshold Rule in all the measured indicators. The number of FP detections was decreased by more than 30%. The obtained results unambiguously confirmed the suitability of this algorithm for Kerberoasting detection, as already identified in our previous research [89].

Figure 5.2: Kerberoasting: Comparison of results

Additional experiments conducted in this thesis identified that the supervised Random Forest is also possibly suitable for this task. This approach missed slightly more attacks

than the Threshold Rule, but on the other hand, it significantly reduced the number of FP results.

The results obtained for both assessed attack techniques imply that detection based on machine learning can

- *improve detection capabilities*,
- *reduce the number of false alarms*,

in comparison with traditional threshold rules for the selected attack scenarios.

5.2 Discussion

Active Directory deployments differ in the number of users, services used, configurations, and policies applied. All these factors affect what activity is considered normal and abnormal in a given context. Therefore, all the presented detection methods must be adapted to the specifics of a particular environment upon deployment.

This is also applicable to threshold rules, which require setting a threshold value for alerting. The threshold value directly influences the number of false alarms the rule produces, as well as its detection sensitivity. For example, a higher threshold value for the number of failed authentication attempts would cause fewer alerts to be generated, but on the other hand, could miss actual attacks.

ML techniques add additional layers of complexity to the deployment process. Customization of the proposed detection rules must start already in the feature extraction step by defining means of distinguishing between account types. The most important adaptation step is then hyperparameter tuning, performed during training of the ML models. The quality of these steps may significantly impact the performance of the resulting ML detection rules.

Since AD environments constantly evolve, the patterns of normal behavior may change over time. As a result, the saved ML models may no longer capture the live state precisely, and their performance may degrade. In such a situation, the model can be retrained on newer log data, or the hyperparameter optimization phase may be repeated. However, the necessity of such actions increases the maintenance complexity of the monitoring solution.

The utilized algorithms listed in table 5.1 cover different ML methods. For each attack scenario, two algorithms of distinct properties were implemented.

The advantage of unsupervised methods is that training a model and the existence of malicious samples is not necessary. On the other hand, the searches applying these methods must run on a substantial amount of data to sufficiently baseline the regular activity. Since the records are aggregated per day, a sequence of logs covering at least several days is necessary. The number of searched events may be significant, and thus the detection rule may be very resource-intensive. On the contrary, (semi-)supervised methods require data from extensive periods only during the training phase. Once the model is saved, it can be applied to smaller portions of log data.

Figure 5.3 illustrates this subject by comparing run times[9] of the searches. Three different rules detecting Password Spraying were run on a Splunk instance, while multiple measurements of their run times were taken. The unsupervised Isolation Forest method was applied to 1-week data, while Random Forest and Threshold Rule were applied on data from one day. Processing a vast number of events by Isolation Forest resulted in a notable difference in the average run times of the detection rules.

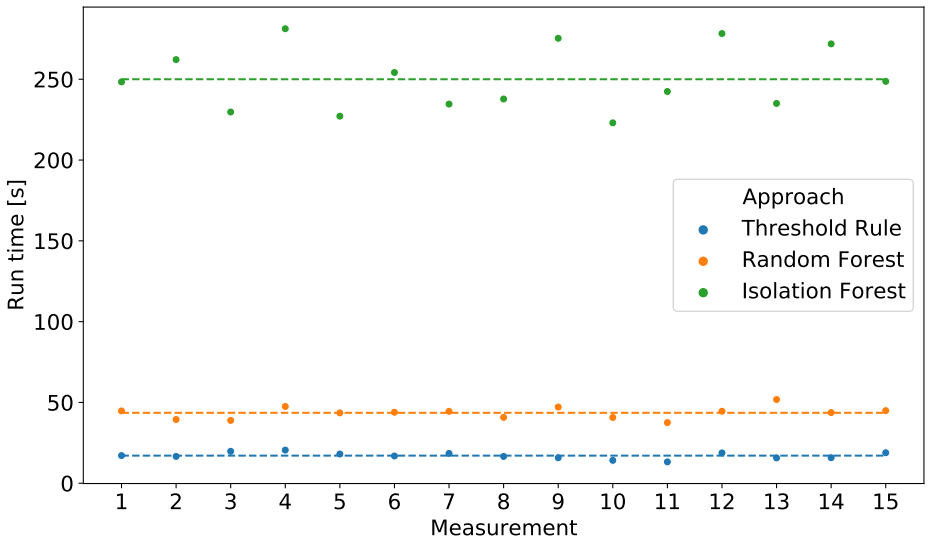

Figure 5.3: Run time of different searches

Despite the initial focus on anomaly detection approaches, supervised methods, especially Random Forest, achieved competitive results in both scenarios. Since no false positives were returned on the test dataset, possible overfitting of the supervised models must be considered.

A general disadvantage of supervised techniques is the need for labeled data containing samples of both benign and malicious logs. Attack events are generally not available and difficult to obtain, as executing the attacks in a live AD environment is time-consuming and may not always be feasible.

Alternatively, the malicious samples may be synthetically generated, similarly as in this thesis. However, generating malicious samples may induce systematic errors in the ML process. There is a risk that the generated events would not capture the attack activity precisely as if executed by an attacker. Learning on insufficient samples may impact the detection capabilities of the resulting supervised ML models.

Semi-supervised methods balance the mentioned disadvantages. Training of models eliminates long run times of unsupervised methods, and opposed to supervised methods, these models do not require malicious samples. A disadvantage of the semi-supervised

[9]The comparison serves illustrative purposes only since explicit details about the number of processed events and properties of the Splunk instance are not provided.

approach seems to be a higher number of false alarms. Although One-class SVM proved to be effective for Kerberoasting detection, it does not appear to be suitable for detecting Password Spraying.

5.3 Contributions

The practical outputs of this thesis comprise the following:

1. Developed *detection rules* detecting Password Spraying and Kerberoasting attack techniques in AD environments. These rules are based on several ML algorithms and are provided in the form of SPL queries. The queries may be deployed to a Splunk instance and used for real-time security monitoring according to instructions provided in sections 4.2.4 and 4.3.4.

2. *Jupyter notebooks* used during the data preparation, modeling, and evaluation phases of the developed solution. The notebooks contain Python source code for manipulating the data, conducting the experiments with different ML models, and generating the outputs. The notebooks are provided as an attachment to this thesis on the enclosed CD.

3. *Reports* presenting outputs from the attached notebooks. The reports contain additional graphical visualizations, configuration listings, and other outputs beyond the scope of the thesis' text. The reports are attached to this thesis on the enclosed CD.

Since log data used for the analysis originated from an authentic AD environment, this data is not provided as a part of this thesis for security reasons. AD audit data is considered sensitive, as Windows events contain usernames, hostnames, IP addresses, and other attributes that might reveal information about objects in the domain or its configuration.

The outputs offered by this thesis aim to help security professionals implement novel ML-based detection mechanisms in AD environments. An actual implementation for detecting two specific attack techniques is provided and can be deployed in Splunk, a tool commonly adopted by organizations as a SIEM solution.

However, the provided outputs are not limited to Splunk technology. Great emphasis was put on describing detailed steps of the ML processes, providing enough information to make it possible to reuse the concepts in other technologies or adapt them to specifics of different AD environments. Moreover, some generic steps, such as feature extraction from Windows events, may be beneficial for developing detections for other attack techniques.

This work differs from most of the existing research reviewed in section 3.2 in several aspects. Firstly, it does not propose a generic ML solution for detecting anomalies in AD but is rather focused on specific attacks instead. A ML algorithm is used as a part of a single detection rule, each designed to detect a particular attack technique. Secondly, the data used with ML models is filtered to specific event IDs and their status codes, which may also be considered a combination of signature-based and ML-based approaches.

5.4 Future Work

A natural continuation of this thesis is the practical deployment of the proposed detection rules in Splunk and their application for security monitoring of Active Directory. The validity of the proposed solution should be confirmed in different AD environments, and the detection capabilities evaluated by executing the actual attacks. As this thesis worked with a static portion of log data, the behavior and performance of the solutions should be monitored in live AD environments over time.

From the research perspective, the suitability of other ML algorithms for detecting the attacks could be reviewed. This thesis included two representative algorithms of each type, mostly due to the current limitations of Splunk MLTK, but there are many additional algorithms available in general. Support for a suitable ML algorithm can be implemented in MLTK through its ML-SPL API.

Supervised methods performed considerably well, despite the initial assumption. However, the inherent dataset imbalance originating from the nature of the attack detection task may still negatively impact the results. The performance of supervised methods could be further improved by addressing the class imbalance. Techniques commonly utilized for this purpose, such as oversampling or undersampling, were not applied in this thesis and could be explored in further research.

Conclusion

The goal of this thesis was to study the possibilities of applying machine learning techniques for security monitoring of Active Directory environments, identify suitable algorithms, and develop a set of detection rules for use in the Splunk platform.

Based on the initial analysis of Active Directory threats and the previous research, two specific attack techniques were selected: Password Spraying and Kerberoasting. Both these techniques are traditionally detected using signature-based rules with threshold conditions. However, adversaries may evade the threshold detection by reducing the number of attempts per time unit. In such a case, the attacks become challenging to detect, as the audited events do not significantly differ from legitimate activity. Hence, this thesis proposed utilizing machine learning techniques for detecting these attacks.

A review of machine learning approaches was performed, with the aim to identify algorithms suitable for use in security monitoring. Existing research applying machine learning to the detection of Active Directory attacks was surveyed, and the possibilities of the Splunk platform with regards to machine learning were analyzed.

Based on the findings, a set of suitable machine learning algorithms was selected. Given the nature of the attacks, an emphasis was put on anomaly detection techniques based on unsupervised and semi-supervised learning, but supervised methods were also included. Two representative algorithms were selected in each category.

Machine learning analysis was performed in compliance with CRISP-DM methodology. Log data used in this thesis originated from an Active Directory environment of a real organization. Malicious samples were added to the dataset and generated according to traces of attack tools execution. The data preparation phase focused on extracting features from the Windows event attributes relevant to the attack techniques considered.

Comprehensive experiments were conducted with the selected machine learning algorithms and their different configurations. The evaluation phase proved that there is not a single generic approach. The best results for Password Spraying detection were obtained by supervised Random Forest and unsupervised Isolation Forest algorithms. Semi-supervised One-class SVM was the best approach for Kerberoasting detection, which confirmed the results of our previous research. However, the supervised approach based on Random Forest appears to be a reasonable alternative.

The best approaches were implemented in the form of Splunk searches and compared to threshold detection rules from the previous research. In both attack scenarios, machine learning techniques significantly reduced the number of false alarms and detected several attacks missed by the threshold rules.

Two detection rules have been developed for each attack technique based on different machine learning types. Each approach has its advantages and disadvantages, which may be of different significance for a particular target Active Directory environment. Supervised methods require a labeled dataset with enough malicious samples available beforehand. Unsupervised methods need to process large numbers of events during detection, implying longer run times and increased demand for computational resources. These disadvantages are eliminated in semi-supervised methods, but those may be prone to produce more false alarms.

The developed solution allows for more efficient security monitoring of Active Directory environments. Since the Splunk platform is commonly adopted by organizations, deployment of the implemented detection rules in practice may be straightforward. However, the contributions of this thesis are not limited to Splunk technology. The detailed descriptions of the machine learning processes allow reproducing the solution in a different technology or reusing the concepts for developing detections of other attack techniques based on Windows Event Log.

Bibliography

1. DESMOND, Brian; RICHARDS, Joe; ALLEN, Robbie; LOWE-NORRIS, Alistair G. Active Directory: Designing, Deploying, and Running Active Directory. In: [online]. 5th ed. O'Reilly Media, 2013, chap. 1-2 [visited on 2021-05-01]. ISBN 978-1-4493-2002-7. Available from: https://learning.oreilly.com/library/view/active-directory-5th/9781449361211/.
2. *Microsoft Docs: Identity and Access documentation* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows-server/identity/identity-and-access.
3. *Microsoft Docs: Authentication vs. authorization* [online]. Microsoft Corporation, 2020 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/azure/active-directory/develop/authentication-vs-authorization.
4. *Microsoft Docs: Windows Authentication Technical Overview* [online]. Microsoft Corporation, 2016 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows-server/security/windows-authentication/windows-authentication-technical-overview.
5. *Microsoft Docs: Microsoft Negotiate* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-negotiate.
6. *Microsoft Docs: Microsoft NTLM* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-ntlm.
7. *Microsoft Docs: How to enable NTLM 2 authentication* [online]. Microsoft Corporation, 2020 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/troubleshoot/windows-client/windows-security/enable-ntlm-2-authentication.

8. *Microsoft Docs: [MS-APDS]: Authentication Protocol Domain Support* [online]. Microsoft Corporation, 2021 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-apds/dd444344-fd7e-430e-b313-7e95ab9c338e.
9. *Microsoft Docs: [MS-NLMP]: NT LAN Manager (NTLM) Authentication Protocol* [online]. Microsoft Corporation, 2021 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-nlmp/b38c36ed-2804-4868-a9ff-8dd3182128e4.
10. BROECKELMANN, Robert. *Kerberos and Windows Security: History* [online]. Medium, 2018 [visited on 2021-05-01]. Available from: https://medium.com/@robert.broeckelmann/kerberos-and-windows-security-history-252ccb510137.
11. NEUMAN, Clifford; YU, Tom; HARTMAN, Sam; RAEBURN, Ken. *The Kerberos network authentication service (V5)* [online]. 2005 [visited on 2021-05-01]. RFC, 4120. RFC Editor. Available from: https://www.rfc-editor.org/rfc/rfc4120.txt.
12. *Microsoft Docs: [MS-KILE]: Kerberos Protocol Extensions* [online]. Microsoft Corporation, 2021 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-kile/2a32282e-dd48-4ad9-a542-609804b02cc9.
13. *Microsoft Docs: Microsoft Kerberos* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-kerberos.
14. METCALF, Sean. *Kerberos, Active Directory's Secret Decoder Ring* [online]. Active Directory Security, 2014 [visited on 2021-05-01]. Available from: https://adsecurity.org/?p=227.
15. *Microsoft Docs: Service Principal Names* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/win32/ad/service-principal-names.
16. METCALF, Sean. *Active Directory Service Principal Names (SPNs) Descriptions* [online]. Active Directory Security, © 2011-2020 [visited on 2021-05-01]. Available from: https://adsecurity.org/?page_id=183.
17. *Microsoft Docs: Event Logging* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/desktop/eventlog/event-logging.
18. KENT, Karen; SOUPPAYA, Murugiah. *Guide to Computer Security Log Management* [online]. 2006 [visited on 2021-05-01]. Tech. rep. National Institute of Standards and Technology. Available from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf. Special Publication 800-92.
19. *Microsoft Docs: Event Logs* [online]. Microsoft Corporation, 2013 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc722404(v=ws.11).

20. *Microsoft Docs: Advanced security audit policies* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/advanced-security-auditing.

21. *Microsoft Docs: PowerShell: Script Tracing and Logging* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/powershell/scripting/windows-powershell/wmf/whats-new/script-logging?view=powershell-5.1.

22. *Microsoft Docs: Event Schema* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/win32/wes/eventschema-schema.

23. CALTAGIRONE, Sergio; PENDERGAST, Andrew; BETZ, Christopher. *The Diamond Model of Intrusion Analysis* [online]. 2013 [visited on 2021-05-01]. Tech. rep. Center For Cyber Intelligence Analysis and Threat Research Hanover Md. Available from: https://apps.dtic.mil/sti/pdfs/ADA586960.pdf.

24. HUTCHINS, Eric M.; CLOPPERT, Michael J.; AMIN, Rohan M. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. *Leading Issues in Information Warfare & Security Research* [online]. 2011 [visited on 2021-05-01]. Available from: https://lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

25. STROM, Blake E.; APPLEBAUM, Andy; MILLER, Doug P.; NICKELS, Kathryn C.; PENNINGTON, Adam G.; THOMAS, Cody B. *MITRE ATT&CK: Design and Philosophy* [online]. 2020 [visited on 2021-05-01]. Tech. rep. The MITRE Corporation. Available from: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf.

26. *Enterprise Matrix* [online]. The MITRE Corporation, 2020 [visited on 2021-05-01]. Available from: https://attack.mitre.org/versions/v8/matrices/enterprise/.

27. *Enterprise tactics* [online]. The MITRE Corporation, 2020 [visited on 2021-05-01]. Available from: https://attack.mitre.org/versions/v8/tactics/enterprise/.

28. *Enterprise Techniques* [online]. The MITRE Corporation, 2020 [visited on 2021-05-01]. Available from: https://attack.mitre.org/versions/v8/techniques/enterprise/.

29. KOTLABA, Lukáš. *Detection of Active Directory attacks*. Prague, 2019. Bachelor's Thesis. Czech Technical University in Prague, Faculty of Information Technology.

30. DUA, Sumeet; DU, Xian. *Data Mining and Machine Learning in Cybersecurity* [online]. Auerbach Publications, 2016 [visited on 2021-05-01]. ISBN 978-1-4398-3943-0. Available from: https://learning.oreilly.com/library/view/data-mining-and/9781439839430/.

31. *Techniques: Brute Force: Password Spraying* [online]. The MITRE Corporation, 2020 [visited on 2021-05-01]. Available from: https://attack.mitre.org/versions/v8/techniques/T1110/003/.

32. *Techniques: Valid Accounts: Domain Accounts* [online]. The MITRE Corporation, 2020 [visited on 2021-05-01]. Available from: https://attack.mitre.org/versions/v8/techniques/T1078/002/.

33. *Top 200 most common passwords of the year 2020* [online]. NordPass, 2021 [visited on 2021-05-01]. Available from: https://nordpass.com/most-common-passwords-list/.

34. *Microsoft Docs: Account Policies* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/account-policies.

35. METCALF, Sean. *Trimarc Research: Detecting Password Spraying with Security Event Auditing* [online]. Trimarc, 2018 [visited on 2021-05-01]. Available from: https://www.trimarcsecurity.com/single-post/2018/05/06/trimarc-research-detecting-password-spraying-with-security-event-auditing.

36. *Microsoft Docs: Active Directory Federation Services: AD FS Password Attack protection* [online]. Microsoft Corporation, 2018 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows-server/identity/ad-fs/technical-reference/ad-fs-password-protection.

37. BULLOCK, Beau et al. *DomainPasswordSpray* [comp. software]. GitHub, 2021 [visited on 2021-05-01]. Available from: https://github.com/dafthack/DomainPasswordSpray.

38. *Microsoft Docs: Audit Logon: Event 4625 F: An account failed to log on* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4625.

39. WILKIN, Jacob. *Spray* [comp. software]. GitHub, 2021 [visited on 2021-05-01]. Available from: https://github.com/Greenwolf/Spray.

40. FLATHERS, Ronnie; FLORES, Alex. *Kerbrute* [comp. software]. GitHub, 2020 [visited on 2021-05-01]. Available from: https://github.com/ropnop/kerbrute.

41. *Microsoft Docs: Audit Credential Validation: Event 4776 S, F: The computer attempted to validate the credentials for an account* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4776.

42. *Microsoft Docs: Microsoft Defender for Identity documentation* [online]. Microsoft Corporation, 2020 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/defender-for-identity/.

43. WEINERT, Alex. *Advancing Password Spray Attack Detection* [online]. Microsoft Tech Community, 2020 [visited on 2021-05-01]. Available from: https://techcommunity.microsoft.com/t5/azure-active-directory-identity/advancing-password-spray-attack-detection/ba-p/1276936.

44. MEDIN, Tim. *Attacking Microsoft Kerberos Kicking the Guard Dog of Hades* [video]. 2014 [visited on 2021-05-01]. Available from: http://www.irongeek.com/i.php?page=videos/derbycon4/t120-attacking-microsoft-kerberos-kicking-the-guard-dog-of-hades-tim-medin. In DerbyCon 4.0, Louisville, USA, 2014-09-26.

45. METCALF, Sean. *Trimarc Research: Detecting Kerberoasting Activity* [online]. Trimarc, 2017 [visited on 2021-05-01]. Available from: https://www.hub.trimarcsecurity.com/post/trimarc-research-detecting-kerberoasting-activity.

46. *Techniques: Steal or Forge Kerberos Tickets: Kerberoasting* [online]. The MITRE Corporation, 2020 [visited on 2021-05-01]. Available from: https://attack.mitre.org/versions/v8/techniques/T1558/003/.

47. KOTLABA, Lukáš; BUCHOVECKÁ, Simona; LÓRENCZ, Róbert. Active Directory Kerberoasting Attack: Monitoring and Detection Techniques. In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*. SciTePress, 2020, pp. 432–439. ISBN 978-989-758-399-5. Available from DOI: 10.5220/0008955004320439.

48. DELPY, Benjamin. *Mimikatz* [comp. software]. GitHub, 2020 [visited on 2021-05-01]. Available from: https://github.com/gentilkiwi/mimikatz.

49. JAGANATHAN, K.; ZHU, L.; BREZAK, J. *The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows* [online]. 2006 [visited on 2021-05-01]. RFC, 4757. RFC Editor. Available from: https://www.rfc-editor.org/rfc/rfc4757.txt.

50. SCHROEDER, William. *Kerberoasting Revisited* [online]. harmj0y, 2019 [visited on 2021-05-01]. Available from: http://www.harmj0y.net/blog/redteaming/kerberoasting-revisited/.

51. *Microsoft Docs: Encryption Type Selection in Kerberos Exchanges* [online]. Microsoft Corporation, 2010 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/archive/blogs/openspecification/encryption-type-selection-in-kerberos-exchanges.

52. *Microsoft Docs: Group Managed Service Accounts Overview* [online]. Microsoft Corporation, 2016 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows-server/security/group-managed-service-accounts/group-managed-service-accounts-overview.

53. SCHROEDER, William. *Rubeus* [comp. software]. GitHub, 2021 [visited on 2021-05-01]. Available from: https://github.com/GhostPack/Rubeus.

54. *Microsoft Docs: Audit Kerberos Service Ticket Operations: Event 4769 S, F: A Kerberos service ticket was requested* [online]. Microsoft Corporation, 2017 [visited on 2021-05-01]. Available from: https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4769.

55. BONACCORSO, Giuseppe. *Machine Learning Algorithms: Popular algorithms for data science and machine learning*. 2nd ed. Packt Publishing, 2018. ISBN 978-1-78934-799-9.

56. FLACH, Peter. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012. ISBN 978-1-107-09639-4.

57. AYODELE, Taiwo Oladipupo. Types of machine learning algorithms. *New Advances in Machine Learning*. 2010, vol. 3, pp. 19–48. Available from DOI: 10.5772/9385.

58. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL V. and Thirion, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER P. and Weiss, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* [online]. 2011, vol. 12, pp. 2825–2830 [visited on 2021-05-01]. Available from: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

59. SAMMUT, Claude; WEBB, Geoffrey I. *Encyclopedia of Machine Learning and Data Mining*. 2nd ed. Springer, 2017. ISBN 978-1-4899-7687-1. Available from DOI: 10.1007/978-1-4899-7687-1.

60. BREIMAN, Leo. Random Forests. *Machine Learning*. 2001, vol. 45, no. 1, pp. 5–32. ISSN 1573-0565. Available from DOI: 10.1023/A:1010933404324.

61. TRIFONOV, Roumen; GOTSEVA, Daniela; ANGELOV, Vasil. Binary classification algorithms. *International Journal of Development Research*. 2017, vol. 7, no. 11, pp. 16873–16879. ISSN 2230-9926.

62. CORTES, Corinna; VAPNIK, Vladimir. Support-vector networks. *Machine Learning*. 1995, vol. 20, no. 3, pp. 273–297. Available from DOI: 10.1007/BF00994018.

63. *Novelty and Outlier Detection* [online]. Scikit-learn Developers, © 2007-2020 [visited on 2021-05-01]. Available from: https://scikit-learn.org/stable/modules/outlier_detection.html.

64. SCHÖLKOPF, Bernhard; WILLIAMSON, Robert; SMOLA, Alex; SHAWE-TAYLOR, John; PLATT, John. Support Vector Method for Novelty Detection. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems* [online]. MIT Press, 1999, pp. 582–588 [visited on 2021-05-01]. Available from: https://proceedings.neurips.cc/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf.

65. BREUNIG, Markus; KRIEGEL, Hans-Peter; NG, Raymond; SANDER, Joerg. LOF: Identifying Density-Based Local Outliers. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. Association for Computing Machinery, 2000, pp. 93–104. Available from DOI: 10.1145/335191.335388.

66. LIU, Fei Tony; TING, Kai Ming; ZHOU, Zhi-Hua. Isolation Forest. In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422. ISSN 2374-8486. Available from DOI: 10.1109/ICDM.2008.17.

67. SOURI, Alireza; HOSSEINI, Rahil. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*. 2018, vol. 8, no. 1, pp. 1–22. Available from DOI: 10.1186/s13673-018-0125-x.

68. BUCZAK, Anna L.; GUVEN, Erhan. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. In: *IEEE Communications Surveys & Tutorials*. IEEE, 2015, vol. 18, pp. 1153–1176. Available from DOI: 10.1109/COMST.2015.2494502.

69. HE, Zecheng; ZHANG, Tianwei; LEE, Ruby B. Machine Learning Based DDoS Attack Detection from Source Side in Cloud. In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2017, pp. 114–120. ISBN 978-1-5090-6644-5. Available from DOI: 10.1109/CSCloud.2017.58.

70. HSIEH, Chih-Hung; LAI, Chia-Min; MAO, Ching-Hao; KAO, Tien-Cheu; LEE, Kuo-Chen. AD2: Anomaly detection on active directory log data for insider threat monitoring. In: *2015 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2015, pp. 287–292. ISBN 978-1-4799-8691-0. Available from DOI: 10.1109/CCST.2015.7389698.

71. GOLDSTEIN, Markus; ASANGER, Stefan; REIF, Matthias; HUTCHISON, Andrew. Enhancing Security Event Management Systems with Unsupervised Anomaly Detection. In: *Proceedings of the 2nd International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM*. SciTePress, 2013, pp. 530–538. ISBN 978-989-8565-41-9. Available from DOI: 10.5220/0004230105300538.

72. MATSUDA, Wataru; FUJIMOTO, Mariko; MITSUNAGA, Takuho. Detecting APT attacks against Active Directory using Machine Leaning. In: *2018 IEEE Conference on Application, Information and Network Security (AINS)*. IEEE, 2018, pp. 60–65. ISBN 978-1-5386-6925-9. Available from DOI: 10.1109/AINS.2018.8631486.

73. UPPSTRÖMER, Viktor; RÅBERG, Henning. *Detecting Lateral Movement in Microsoft Active Directory Log Files*. Karlskrona, 2019. Master's Thesis. Blekinge Institute of Technology, Faculty of Computing.

74. MEIJERINK, Mart. *Anomaly-based Detection of Lateral Movement in a Microsoft Windows Environment*. Enschede, 2019. Master's Thesis. University of Twente, Faculty of Electrical Engineering, Mathematics & Computer Science.

75. *Splunk Docs: Splunk Enterprise Overview* [online]. Splunk Inc., 2021 [visited on 2021-05-01]. Available from: https://docs.splunk.com/Documentation/Splunk/8.1.3/Overview/AboutSplunkEnterprise.

76. KAVANAGH, Kelly; BUSSA, Toby; SADOWSKI, Gorka. *Gartner Magic Quadrant for Security Information and Event Management* [online]. Gartner, 2020 [visited on 2021-05-01]. Available from: https://www.gartner.com/en/documents/3981040/magic-quadrant-for-security-information-and-event-manage.

77. *Splunk Docs: Splunk Enterprise Admin Manual: Types of Splunk Enterprise licenses* [online]. Splunk Inc., 2021 [visited on 2021-05-01]. Available from: https://docs.splunk.com/Documentation/Splunk/8.1.3/Admin/TypesofSplunklicenses.

78. *Splunk Docs: Splunk Enterprise Admin Manual: Apps and add-ons* [online]. Splunk Inc., 2020 [visited on 2021-05-01]. Available from: https://docs.splunk.com/Documentation/Splunk/8.1.3/Admin/Whatsanapp.

79. *Splunk Docs: Deploy and Use the Splunk Add-on for Windows* [online]. Splunk Inc., 2021 [visited on 2021-05-01]. Available from: https://docs.splunk.com/Documentation/WindowsAddOn/8.1.1/User/AbouttheSplunkAdd-onforWindows.

80. CARASSO, David. Exploring Splunk. In: [online]. New York: CITO Research, 2012, chap. 2-4 [visited on 2021-05-01]. ISBN 978-0-9825506-7-0. Available from: https://www.splunk.com/pdfs/exploring-splunk.pdf.

81. *Splunk Docs: Splunk Enterprise Search Manual: About the search language* [online]. Splunk Inc., 2017 [visited on 2021-05-01]. Available from: https://docs.splunk.com/Documentation/Splunk/8.1.3/Search/Aboutthesearchlanguage.

82. *Splunk Docs: Splunk Machine Learning Toolkit User Guide* [online]. Splunk Inc., 2021 [visited on 2021-05-01]. Available from: https://docs.splunk.com/Documentation/MLApp/5.2.1/User.

83. SPLUNK INC. *Python for Scientific Computing* [comp. software]. 2020 [visited on 2021-05-01]. Available from: https://splunkbase.splunk.com/app/2882/.

84. SPLUNK INC. *Splunk MLTK Algorithms on GitHub* [comp. software]. 2019 [visited on 2021-05-01]. Available from: https://splunkbase.splunk.com/app/4403/.

85. CHAPMAN, Pete; CLINTON, Julian; KERBER, Randy; KHABAZA, Thomas; REINARTZ, Thomas; SHEARER, Colin; WIRTH, Rudiger. *CRISP-DM 1.0: Step-by-step data mining guide* [online]. 2000 [visited on 2021-05-01]. Tech. rep. The CRISP-DM consortium. Available from: https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf.

86. JENSEN, Kenneth. *A diagram showing the relationship between the different phases of CRISP-DM and illustrates the recursive nature of a data mining project* [online]. 2012 [visited on 2021-05-01]. Available from: https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png. File: CRISP-DM Process Diagram.png.

87. *State of Data Science and Machine Learning 2020* [online]. Kaggle, 2020 [visited on 2021-05-01]. Available from: https://www.kaggle.com/kaggle-survey-2020.

88. SOKOLOVA, Marina; LAPALME, Guy. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*. 2009, vol. 45, no. 4, pp. 427–437. ISSN 0306-4573. Available from DOI: 10.1016/j.ipm.2009.03.002.

89. KOTLABA, Lukáš; BUCHOVECKÁ, Simona; LÓRENCZ, Róbert. Active Directory Kerberoasting Attack: Detection using Machine Learning Techniques. In: *Proceedings of the 7th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*. SciTePress, 2021, pp. 376–383. ISBN 978-989-758-491-6. Available from DOI: 10.5220/0010202803760383.

Appendix A

Acronyms

ACC accuracy.

AD Active Directory.

AD DS Active Directory Domain Services.

AD FS Active Directory Federation Services.

AD LDS Active Directory Lightweight Directory Services.

AES Advanced Encryption Standard.

API application programming interface.

AS Authentication Service.

CPU central processing unit.

CRISP-DM Cross-Industry Standard Process for Data Mining.

CSV Comma Separated Values.

DC domain controller.

DES Data Encryption Standard.

DLL dynamic link library.

DM data mining.

DoS Denial of Service.

FN false negative.

FP false positive.

FQDN Fully Qualified Domain Name.

GINA Graphical Identification and Authentication.

GPO Group Policy Object.

ID identifier.

IDS intrusion detection system.

k-NN k-Nearest Neighbor.

KDC Key Distribution Center.

LDAP Lightweight Directory Access Protocol.

LOF Local Outlier Factor.

LSA Local Security Authority.

ML machine learning.

MLTK Machine Learning Toolkit.

NTLM NT LAN Manager.

OS operating system.

OU organizational unit.

PAC Privilege Account Certificate.

PPV positive predictive value.

RBF radial basis function.

RFC Random Forest Classifier.

RPC Remote Procedure Call.

SAM Security Accounts Manager.

SGT Service-Granting Ticket.

SID security identifier.

SIEM Security Information and Event Management.

SMB Server Message Block.

SPL Search Processing Language.

SPN Service Principal Name.

SSO single sign-on.

SSP Security Support Provider.

SVC Support Vector Classifier.

SVM Support Vector Machine.

TGS Ticket-Granting Service.

TGT Ticket-Granting Ticket.

TN true negative.

TP true positive.

TPR true positive rate.

TTPs Tactics, Techniques, and Procedures.

UPN User Principal Name.

XML eXtensible Markup Language.

Appendix B

Contents of the Enclosed CD

- `README.md` ..... the file with CD contents description
- `ml_analysis` ..... the directory with the outputs of machine learning analysis
  - `notebooks` ..... the directory with Jupyter and Python source files
    - `helper_functions.py` ..... auxiliary Python functions
    - `kerberoasting.ipynb` ..... Jupyter notebook for Kerberoasting
    - `password_spraying.ipynb` ..... Jupyter notebook for Password Spraying
    - `requirements.txt` ..... the file with Python dependencies
  - `reports` ..... the directory with HTML reports
    - `kerberoasting.html` ..... HTML report for Kerberoasting
    - `password_spraying.html` ..... HTML report for Password Spraying
- `thesis` ..... the thesis text directory
  - `DP_Kotlaba_Lukas_2021.pdf` ..... the thesis text in PDF format
  - `DP_Kotlaba_Lukas_2021.zip` ..... ZIP archive with LaTeX source files of the thesis

Appendix C

Splunk Searches

C.1 Password Spraying

```
source=XmlWinEventLog:Security ((EventID=4625
    SubStatus=0xC000006A) OR (EventID=4771 Status=0x18))
| eval orig_time=_time
| eval src=if(cidrmatch("0.0.0.0/0", IpAddress), IpAddress,
    WorkstationName)
| rename TargetUserName AS user
| bin _time span=1d
| eventstats count AS cnt BY user, src, _time
| streamstats count AS event_no
| stats values(user) AS user,
        values(eval(cnt."-".user)) AS user_cnt,
        values(eval(orig_time."-".event_no)) AS orig_time,
        count AS cnt, dc(user) AS user_dc
        BY _time, src
| where user_dc > 1
| mvexpand orig_time
| rex field=orig_time "(?<time>\d+)-.+"
| sort 0 src, time
| delta time AS tdelta
| fields - _time, time, orig_time, user
| mvcombine tdelta
| eval deltas=mvindex(tdelta, 1, -1)
| streamstats count AS event_no
| eventstats mode(deltas) AS tdelta_mod BY event_no
| eval tdelta_mod_val=mvmap(deltas, if(like(deltas,
    tdelta_mod), 1, NULL))
| eval tdelta_mod_cnt=mvcount(tdelta_mod_val)
```

```
25 | fields - deltas, tdelta_mod_val, tdelta_mod
26 | eval user_cnt_name=split(user_cnt, " ")
27 | fields - tdelta, user_cnt
28 | mvexpand user_cnt_name
29 | rex field=user_cnt_name "(?<user_cnt>\d+)-.+"
30 | fields - user_cnt_name
31 | mvcombine user_cnt
32 | eventstats mode(user_cnt) AS user_mod,
33             avg(user_cnt) AS user_avg BY event_no
34 | eval user_mod_val=mvmap(user_cnt, if(like(user_cnt,
     user_mod), 1, NULL))
35 | eval user_mod_cnt=mvcount(user_mod_val)
36 | fields - user_mod, user_mod_val, user_cnt, event_no
37 | ...
```

Listing C.1: Password Spraying: Data preparation search for Isolation Forest

```
1 source=XmlWinEventLog:Security ((EventID=4625
    SubStatus=0xC000006A) OR (EventID=4771 Status=0x18))
2 | eval orig_time=_time
3 | eval src=if(cidrmatch("0.0.0.0/0", IpAddress), IpAddress,
    WorkstationName)
4 | rename TargetUserName AS user
5 | bin _time span=1d
6 | eventstats count AS cnt BY user, src, _time
7 | streamstats count AS event_no
8 | stats values(user) AS user,
9         values(eval(cnt."-".user)) AS user_cnt,
10        values(eval(orig_time."-".event_no)) AS orig_time,
11        count AS cnt, dc(user) AS user_dc,
12        count(eval(EventID="4625")) AS evt_4625_cnt,
13        count(eval(EventID="4771")) AS evt_4771_cnt
14        BY _time, src
15 | where user_dc > 1
16 | eval user_pers_dc=mvcount(mvfilter(match(user, <REGEX>)))
17 | fillnull user_pers_dc
18 | mvexpand orig_time
19 | rex field=orig_time "(?<time>\d+)-.+"
20 | sort 0 src, time
21 | delta time AS tdelta
22 | fields - _time, time, orig_time, user
23 | mvcombine tdelta
24 | eval deltas=mvindex(tdelta, 1, -1)
25 | streamstats count AS event_no
```

```
26 | eventstats sum(deltas) AS tdelta_sum,
27              mode(deltas) AS tdelta_mod,
28              avg(deltas) AS tdelta_avg,
29              max(deltas) AS tdelta_max BY event_no
30 | eval tdelta_mod_val=mvmap(deltas, if(like(deltas,
    tdelta_mod), 1, NULL))
31 | eval tdelta_mod_cnt=mvcount(tdelta_mod_val)
32 | fields - deltas, tdelta_mod_val, tdelta_mod
33 | eval user_cnt_name=split(user_cnt, " ")
34 | fields - tdelta, user_cnt
35 | mvexpand user_cnt_name
36 | rex field=user_cnt_name "(?<user_cnt>\d+)-.+"
37 | fields - user_cnt_name
38 | mvcombine user_cnt
39 | eventstats mode(user_cnt) AS user_mod BY event_no
40 | eval user_mod_val=mvmap(user_cnt, if(like(user_cnt,
    user_mod), 1, NULL))
41 | eval user_mod_cnt=mvcount(user_mod_val)
42 | fields - user_mod, user_mod_val, user_cnt, event_no
43 | ...
```

Listing C.2: Password Spraying: Data preparation search for RFC

C.2 Kerberoasting

```
source=XmlWinEventLog:Security EventID=4769
    TicketEncryptionType IN (0x1, 0x3, 0x17, 0x18)
| search IpAddress != "::1"
| regex ServiceName != "\$$"
| bin _time span=1d
| stats values(ServiceName) AS ServiceName,
        dc(ServiceName) AS svc_dc
        BY _time, IpAddress, TargetUserName
| where svc_dc > 2
| rex field=TargetUserName "((?<user>.+)@[\w\.]+)"
| eval user_is_np=if(match(user, <REGEX>), "1", "0")
| eval user_is_pers=if(match(user, <REGEX>), "1", "0")
| eval user_is_sys=if(match(user, <REGEX>), "1", "0")
| fields - user
| makemv ServiceName
| eval svc_sql_dc=mvcount(mvfilter(match(ServiceName,
    <REGEX>)))
| eval svc_app_dc=mvcount(mvfilter(match(ServiceName,
    <REGEX>)))
| fillnull svc_sql_dc svc_app_dc
| eval svc_other_dc=svc_dc - svc_sql_dc - svc_app_dc
| eval svc_other_dc=if(svc_other_dc < 0, 0, svc_other_dc)
| ...
```

Listing C.3: Kerberoasting: Data preparation search for One-class SVM and RFC