





SAPIENZA  
UNIVERSITÀ DI ROMA

## Automating attack path identification and classification in an Active Directory environment using BloodHound

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Informatica

Candidate

Alessandro Nocerino

ID number 2047604

Thesis Advisor

Daniela Gorla

A handwritten signature in black ink, likely belonging to Daniela Gorla, the Thesis Advisor.

Academic Year 2025/2026

Thesis not yet defended

---

**Automating attack path identification and classification in an Active Directory environment using BloodHound**

Bachelor's thesis. Sapienza – University of Rome

© 2025 Alessandro Nocerino. All rights reserved

This thesis has been typeset by L<sup>A</sup>T<sub>E</sub>X and the Sapthesis class.

Author's email: [anocerino2004@gmail.com](mailto:anocerino2004@gmail.com)

## Abstract

This work will discuss the finding of attack paths, classifying them based on their severity, in an Active Directory environment.

A **directory** is a hierarchical structure that stores information about objects on a network. A directory service, such as **Active Directory Domain Services** (AD DS), provides the methods for storing directory data and making this data available to network users and administrators. For example, AD DS stores information about various object types, like user accounts, computers, groups, Group Policy Objects and more. AD DS also provides a way for authorized users on the same network to access this information.

Objects can have relationships with other objects, meaning the possibility to read/write their properties with the right privileges. For example, a user might have the possibility to change another user account's password, which could be useful for an IT admin to reset forgotten passwords. The prospect of lateral movement with a privilege like this one is clear: by compromising the privileged user account, it is easy to compromise the other account by changing its password to something the attacker knows. These exploitable relationships between objects, which are the result of misconfigurations if they allow for privilege escalation, are what make an **Attack Path**: one can "hop" from one object to another, with each object potentially possessing new privileges.

Enumerating these attack paths requires checking each object's privileges and its relationships with other objects, as to determine how easy it is to compromise and what is the prospect of possible lateral movement post-compromise. This is a very long and cumbersome process in which it is very easy to miss critical attack paths if one tries to do it manually. **BloodHound** is a tool which, after collecting data about an AD environment and storing it in a database, represents it as a graph, with the objects being nodes and the relationships being edges. This makes enumeration much easier, as one can simply look at the graph to find attack paths.

In very large AD environments, graphs generated by BloodHound can become extremely large and complex, again making attack path enumeration a long and error prone process. This work will propose a way to automate this process using the database generated by BloodHound: attack paths will be ranked by their criticality, with criteria being how easy it is to gain an initial foothold from outside through the starting node of this path and how easy it is to traverse, in other words, the graph's edges will be assigned a weight based on their criticality. Finally, paths will be displayed from more severe ones to less.

In **Chapter 1** there will be an introduction on Active Directory and BloodHound, which the research relies on, then about the research itself. In **Chapter 2** an overview of Active Directory and its components is given. In **Chapter 3** we will look into attacking Active Directory, looking at some techniques and introducing BloodHound. In **Chapter 4** we will talk about automating attack path identification and classification based on criteria that will be defined, then a possible implementation. **Chapter 5** is a short conclusion.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Active Directory . . . . .	1
1.1.1	Overview . . . . .	1
1.1.2	Threat Landscape . . . . .	1
1.2	BloodHound and Attack Path Analysis . . . . .	2
1.2.1	Limitation . . . . .	3
1.3	Research Aim and Objectives . . . . .	3
1.3.1	Research Aim . . . . .	3
1.3.2	Research Objectives . . . . .	3
1.4	Scope and Limitations . . . . .	4
1.4.1	Scope . . . . .	4
1.4.2	Limitations . . . . .	4
<b>2</b>	<b>Active Directory</b>	<b>5</b>
2.1	History and Evolution of Active Directory . . . . .	5
2.2	Active Directory Architecture . . . . .	5
2.2.1	Forests, Trees, and Domains . . . . .	6
2.2.2	Objects in Active Directory . . . . .	6
2.2.3	Organizational Units (OUs) . . . . .	6
2.2.4	Group Policy . . . . .	6
2.2.5	Schema . . . . .	7
2.2.6	Global Catalog and Replication . . . . .	7
2.2.7	Trusts . . . . .	7
2.3	Authentication and Authorization in Active Directory . . . . .	8
2.3.1	Kerberos Authentication . . . . .	8
2.3.2	Kerberos Authentication Workflow . . . . .	8
2.3.3	NTLM Authentication . . . . .	10
2.3.4	Authorization Mechanisms . . . . .	10
2.3.5	Tiered Administration Model . . . . .	11
<b>3</b>	<b>Attacking Active Directory</b>	<b>12</b>
3.1	Initial Foothold into Active Directory . . . . .	12
3.1.1	Phishing and Credential Theft . . . . .	12
3.1.2	Password Attacks . . . . .	12
3.1.3	AS-REP Roasting . . . . .	12
3.2	Lateral Movement . . . . .	13
3.2.1	Pass-the-Hash . . . . .	13
3.2.2	Kerberoasting . . . . .	13
3.2.3	Weak Access Control Lists (ACLs) . . . . .	14
3.2.4	Delegation . . . . .	14

---

3.3	PowerView . . . . .	15
3.4	BloodHound . . . . .	15
3.4.1	Nodes . . . . .	16
3.4.2	Edges . . . . .	16
3.4.3	Querying with Cypher . . . . .	17
<b>4</b>	<b>Attack path classification</b>	<b>18</b>
4.1	Initial Foothold Difficulty . . . . .	18
4.2	Traversing the path . . . . .	19
4.2.1	Tier 1 . . . . .	20
4.2.2	Tier 2 . . . . .	21
4.2.3	Tier 3 . . . . .	21
4.2.4	Tier 4 . . . . .	22
4.2.5	Edges requiring additional conditions . . . . .	23
4.2.6	DCSync edge . . . . .	24
4.3	Implementation . . . . .	24
4.3.1	Test Environment / Dataset . . . . .	24
4.3.2	Data Collection . . . . .	25
4.3.3	Path Enumeration . . . . .	25
4.3.4	Results and Example Output . . . . .	30
<b>5</b>	<b>Conclusion</b>	<b>34</b>
	<b>Bibliography</b>	<b>35</b>

# Chapter 1

## Introduction

### 1.1 Active Directory

#### 1.1.1 Overview

In modern enterprise environments, the management of user identities, authentication, and authorization is critical to ensuring both operational efficiency and protection against cyber threats. Active Directory (AD), introduced by Microsoft in Windows 2000, has become the cornerstone of identity and access management in Windows-based networks [7]. By providing centralized authentication, authorization, and directory services, AD enables organizations to manage thousands of users, devices, and applications under a unified framework [1].

Active Directory operates as a hierarchical, centralized directory service that stores information about objects on a network, such as users, groups, computers, printers, and security policies. Through its domain-based architecture, it allows administrators to organize resources, enforce security rules, and delegate administrative responsibilities. Features such as Group Policy Objects (GPOs) enable consistent configuration management across large-scale environments, while replication mechanisms ensure data integrity and availability across multiple domain controllers [1] [3].

Over the years, AD has evolved beyond traditional identity management into a critical component of enterprise security and IT infrastructure. It facilitates single sign-on and access control, and plays a central role in compliance auditing and incident response.

#### 1.1.2 Threat Landscape

AD's widespread use and certain architectural vulnerabilities make it a high-value target in cyberattacks. In cases of security breaches, adversaries often focus on compromising AD to escalate privileges, enabling them to spread malware, exfiltrate data, and launch ransomware attacks. Identifying the breach source and extent of damage can be challenging, especially with attackers able to maintain persistence within the network [7]. IBM's 2021 Cost of a Data Breach Report revealed that attackers can remain undetected in compromised networks for up to 277 days after obtaining domain administrator credential [4].

AD has been the backbone of enterprise identity and access management for over two decades, and its ubiquity has made it a prime target for adversaries. Once

compromised, AD provides attackers with the “keys to the kingdom,” enabling lateral movement, privilege escalation, and persistence across an organization’s network [7]. The importance of securing AD is underscored by both academic research and industry reports highlighting the scale of attacks against directory services.

Recent empirical work has shown how modern ransomware can directly impact AD domain services. McDonald et al. [9] demonstrated through controlled experiments with variants such as WannaCry and TeslaCrypt that, although domain services may continue running after infection, they become effectively unusable due to file encryption and cascading failures. Schroder and Park [11] further argue that tiered AD architectures are critical in thwarting lateral movement, highlighting how architectural weaknesses translate directly into elevated risk.

Industry data paints a concerning picture. Approximately 50% of organizations reported experiencing an AD attack within the last 1–2 years, with over 40% of these proving successful, and penetration tests against AD environments succeeding in 82% of cases [5]. In ransomware incidents, AD is targeted in nine out of ten cases, with 87% of these attacks causing significant operational disruption, yet only 27% of organizations maintain dedicated, recoverable AD backups [6, 8].

These findings underscore the urgency of strengthening AD security. The evidence highlights a dual challenge: on the one hand, attackers continue to exploit fundamental weaknesses in AD design and configuration; on the other, organizations often lack the visibility and resilience required to detect, mitigate, and recover from such compromises. This convergence of threats makes Active Directory both a high-value target and a critical focal point for advancing enterprise security practices.

Successful Active Directory attacks consist of three primary steps: discovery, privilege escalation through theft of valid account credentials, and gaining access to other computers in the network/domain. Once attackers gain a foothold in the target network, they immediately shift their focus to gaining elevated access to additional systems that will help them accomplish their final goal, such as encrypting and exfiltrating organizational data [7].

## 1.2 BloodHound and Attack Path Analysis

While traditional enumeration techniques in AD rely on manual queries and scripts to identify objects, relationships and permissions, the complexity of large enterprise environments makes such approaches inefficient and error-prone. To address this, BloodHound was created by Rohan Vazarkar, Will Schroeder, and Andy Robbins [36]. It is an automated tool that leverages graph theory to model AD environments. It ingests data about AD components, and represents them as nodes and edges within a graph database. By doing so, it highlights potential attack paths: sequences of permissions and privileges that an adversary could exploit to escalate from a low-privilege account to a high-value target such as a Domain Administrator.

BloodHound has become a widely adopted tool in both offensive and defensive security contexts. For red teams and penetration testers, it provides a powerful means of mapping privilege escalation opportunities within a matter of minutes. For defenders, it offers insight into misconfigurations and privilege assignments that may otherwise remain invisible in complex environments [7]. Its ability to identify non-obvious privilege chains makes the advantage it brings to enumeration clear.

### 1.2.1 Limitation

However, as AD environments grow in size, the graphs generated by BloodHound can contain tens of thousands of nodes and relationships. Manually navigating and interpreting these dense graphs quickly becomes infeasible for administrators and security teams. Critical attack paths may be buried under large volumes of less relevant data, creating the risk that organizations fail to address their most serious vulnerabilities. This complexity introduces a need for automation: methods that can not only extract attack paths from the BloodHound database but also prioritize them according to their likelihood, feasibility, or impact on enterprise security.

While the tool excels at data collection and visualization, its utility in large-scale environments is limited by the cognitive load placed on the analyst. Automating the identification and ranking of attack paths is therefore a crucial step toward making BloodHound more actionable for defenders and more practical for use in enterprise-scale Active Directory security operations.

## 1.3 Research Aim and Objectives

### 1.3.1 Research Aim

The aim of this dissertation is to automate the identification and prioritization of attack paths in Active Directory environments by leveraging the database generated by BloodHound. In large-scale networks, the graphs produced by BloodHound often contain tens of thousands of nodes and relationships, making manual exploration impractical for administrators and defenders. This research seeks to reduce this complexity by developing methods that automatically extract and rank the most relevant attack paths, thereby improving the usability of BloodHound in enterprise-scale environments.

### 1.3.2 Research Objectives

To achieve this aim, the following objectives have been defined:

1. **Analyze the BloodHound database schema:** Examine the structure of BloodHound's Neo4j database to understand how AD entities and relationships are stored and represented.
2. **Design automated path-finding techniques:** Develop algorithms to query the BloodHound database for privilege escalation paths, with a focus on scalability and efficiency in large environments.
3. **Implement path prioritization:** Establish criteria (e.g., path length, exploitation feasibility) to rank and highlight the most significant attack paths.
4. **Evaluate the proposed approach:** Test the automated system using simulated Active Directory datasets to assess improvements in accuracy, scalability, and usability compared to manual graph exploration.
5. **Discuss implications for defenders:** Analyze how automation can support defensive security operations and recommend practical applications of the research for enterprise AD security.

## 1.4 Scope and Limitations

### 1.4.1 Scope

This dissertation focuses on the analysis of Active Directory (AD) environments through the database generated by BloodHound. The scope is limited to the following areas:

- **Active Directory environments:** The research primarily considers on-premises AD infrastructures. While Azure Active Directory and hybrid identity systems are acknowledged, they are not the central focus of this work.
- **BloodHound data model:** The study is concerned with the graph database generated by BloodHound, particularly the Neo4j schema that represents AD objects, relationships, and permissions.
- **Evaluation:** Testing is performed on simulated and/or controlled AD datasets. The evaluation aims to demonstrate feasibility and potential rather than provide exhaustive benchmarking across all possible enterprise configurations.

### 1.4.2 Limitations

The research is subject to the following limitations:

- **Dataset availability:** Access to large, real-world enterprise AD datasets is restricted for ethical and confidentiality reasons. As a result, evaluation relies on lab-based or publicly available data, which may not capture the full diversity of production environments.
- **Tool dependency:** The research depends on BloodHound's existing data collection mechanisms and Neo4j database format. Improvements to data collection or changes in BloodHound's architecture may impact the applicability of the proposed methods.

## Chapter 2

# Active Directory

This chapter will describe in more detail the components and working of Active directory

### 2.1 History and Evolution of Active Directory

Active Directory was first introduced with Windows 2000 Server as a replacement for the Windows NT domain model. Earlier NT-based domains relied on a single Primary Domain Controller (PDC) with optional Backup Domain Controllers (BDCs), which limited scalability and fault tolerance. AD addressed these limitations by adopting open standards such as the Lightweight Directory Access Protocol (LDAP) for queries and Kerberos for authentication, while introducing multi-master replication across domain controllers to improve resilience and administrative flexibility [13].

Over subsequent releases of Windows Server, Microsoft enhanced AD with new features. Windows Server 2003 added refinements to Group Policy and introduced the Global Catalog for faster forest-wide searches. Windows Server 2008 introduced fine-grained password policies and Read-Only Domain Controllers (RODCs), designed to improve security in distributed environments. Later versions, including Windows Server 2012 and 2016, expanded auditing capabilities and improved integration with cloud services [13].

The rise of cloud computing marked the next stage in AD's evolution, with Microsoft introducing Azure Active Directory as a cloud-based identity and access management solution. Today, many enterprises operate in hybrid environments that integrate traditional on-premises AD with Azure AD, enabling single sign-on (SSO), multi-factor authentication (MFA), and tighter integration with Microsoft's cloud ecosystem [13].

### 2.2 Active Directory Architecture

AD is built on a hierarchical architecture designed to provide scalability, flexibility and centralized management for enterprise networks. Its structure allows administrators to organize resources logically, enforce security policies, and delegate administrative responsibilities. The architecture is composed of several key building blocks, including forests, trees, domains, organizational units, objects, and the schema.

### 2.2.1 Forests, Trees, and Domains

At the highest level of the hierarchy is the **forest**, which represents the security boundary of an Active Directory environment. A forest contains one or more **trees**, each of which is a collection of one or more **domains**. A domain is the fundamental unit of AD, representing a logical grouping of users, computers, and resources. Domains provide the administrative boundary for policies and authentication. Domains within the same tree share a contiguous namespace, while different trees in the same forest share a common schema and global catalog [13, 3].

### 2.2.2 Objects in Active Directory

Objects are the fundamental data units stored in AD. Each object represents a distinct entity within the directory and is defined by the schema. The most common object classes include:

- **User objects:** Represent individual user accounts. They contain attributes such as usernames, passwords (stored in hashed form), group memberships, and personal details (e.g., email address, department).
- **Group objects:** Represent collections of users, computers, or other groups. Groups simplify administration by allowing permissions to be assigned collectively.
- **Computer objects:** Represent devices (workstations, servers) joined to the domain. They allow administrators to manage machine-specific attributes, apply policies, and authenticate computers to AD.
- **Service accounts:** Special accounts (e.g., Managed Service Accounts, Group Managed Service Accounts) used to run applications and services with defined privileges.

Objects are organized into containers, such as organizational units (OUs), to support delegation and policy application [13].

### 2.2.3 Organizational Units (OUs)

Within a domain, resources can be further organized using **Organizational Units (OUs)**. OUs provide a flexible way to group users, computers, and groups according to organizational structures (e.g., departments, geographic locations). They also enable delegation of administrative control, allowing specific administrators to manage only the resources within a given OU [3].

### 2.2.4 Group Policy

**Group Policy Objects (GPOs)** are an integral part of AD architecture, allowing administrators to define configuration settings and security policies that apply to users and computers within a domain or OU. GPOs can enforce password policies, software installations, desktop restrictions, and security configurations. Their hierarchical inheritance model allows policies to be applied at multiple levels, with rules for precedence and conflict resolution [13].

### 2.2.5 Schema

The AD schema is a critical component of Microsoft's Active Directory. It defines the structure and rules for all objects and attributes that can exist within an AD forest. Essentially, it acts as a blueprint for how data is organized and managed in the directory.

The schema contains formal definitions for two key elements:

- **Object Classes:** These define the types of objects that can be created in AD, such as users, groups, computers, and organizational units.
- **Attributes:** These specify the properties or characteristics of each object, such as a user's name, email address, or group membership.

Each object class and attribute in the schema is uniquely identified and follows a specific syntax, ensuring consistency and compatibility across the directory.

The schema is *extensible*, meaning that administrators or developers can customize it by adding new object classes or attributes to meet specific organizational needs. However, modifying the schema should be done cautiously, as changes are replicated across the entire AD forest and can have significant impacts.

The schema is stored in the **Schema Partition** of the AD database and is replicated to all domain controllers in the forest. This ensures that all domain controllers share a consistent understanding of the directory's structure and data [3].

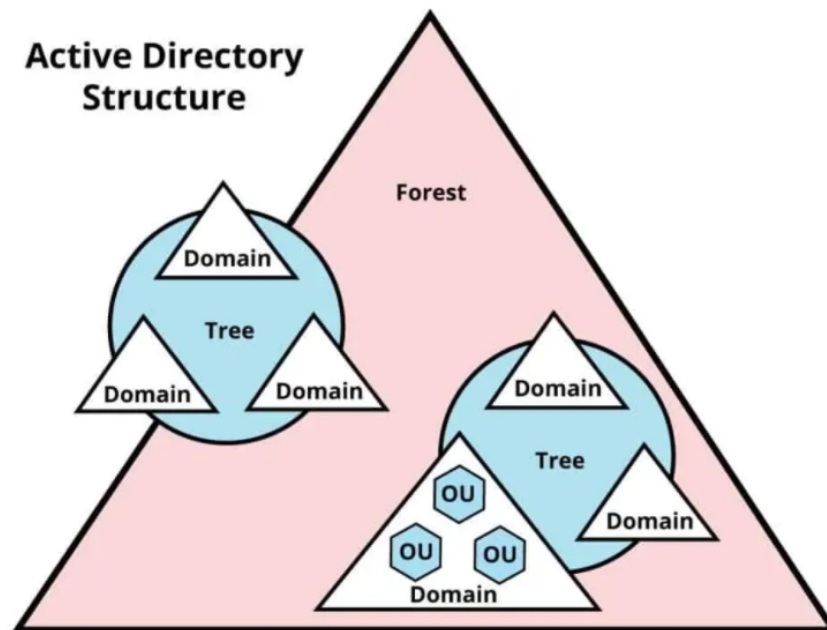
### 2.2.6 Global Catalog and Replication

The **global catalog** is a distributed data repository that contains a searchable, partial representation of every object in the forest. It enables efficient searches across multiple domains, ensuring that users and administrators can locate resources without needing to query each domain individually. Replication between domain controllers ensures that changes made to objects are synchronized across the environment, using a multi-master model to enhance resilience and fault tolerance [3].

### 2.2.7 Trusts

To support complex enterprise environments, AD implements **trust relationships** that allow users in one domain or forest to access resources in another. Trusts can be one-way or two-way, and transitive or non-transitive, providing flexibility in designing secure multi-domain or multi-forest environments [3].

**Scope Note.** Although Active Directory includes many architectural elements such as the schema, replication mechanisms, and trust relationships between domains and forests, the focus of this dissertation is narrower. The work presented here is primarily concerned with the objects and relationships that form *attack paths* within a domain. In practice, these paths are composed mainly of user, group, and computer objects, along with the access control relationships that link them. As such, while the broader architecture provides useful context, subsequent chapters will emphasize those object classes most relevant to privilege escalation and lateral movement analysis.



**Figure 2.1.** Active Directory hierarchical architecture: forests, trees, domains, and OUs.

## 2.3 Authentication and Authorization in Active Directory

Authentication and authorization are central functions of Active Directory, ensuring that only legitimate users and devices gain access to resources, and that their access is limited according to defined security policies. AD implements these functions primarily through the Kerberos protocol, with support for legacy mechanisms such as NTLM (NT LAN Manager), and enforces access decisions using role- and group-based models.

### 2.3.1 Kerberos Authentication

Kerberos [37] is a network authentication protocol developed at MIT's Project Athena, designed to provide secure identity verification in open and distributed computing environments. It is a trusted third-party service based on the Needham-Schroeder model [39], enhanced with timestamps to mitigate replay attacks. Kerberos maintains a central database of principals (users and services) and their private keys, from which it issues tickets and session keys to support authentication. The authentication process relies on two core credentials: tickets, which securely identify a client to a server, and authenticators, which prove that the entity presenting a ticket is its rightful owner. A ticket is encrypted with the server's key and contains information such as client identity, network address, lifetime, and a session key, while the authenticator, encrypted with that session key, provides freshness through timestamps [14].

### 2.3.2 Kerberos Authentication Workflow

The service responsible for authenticating and authorizing principals is called the Key Distribution Center (KDC). The KDC comprises two subservices: the

authentication server (AS) responsible for authentication and the ticket-granting service (TGS) responsible for issuing tickets. The KDC exists and runs in any writable domain controller (DC).

1. The client hashes the user's password. This hash serves as the secret key for securing communication between the client and the KDC.

The client then sends an encrypted timestamp (using the secret key) to the AS. The AS verifies the client's knowledge of the password by decrypting the timestamp with the stored hash from the AD database.

2. If verification succeeds, the AS replies with:
  - (a) An encryption key (for subsequent communication with the KDC), encrypted with the user's password hash.
  - (b) A *Ticket-Granting Ticket (TGT)*, which contains information about the user and is encrypted with the `krbtgt` account key (known only to the TGS).
3. The client uses the TGT to request access to a specific service from the TGS.
4. The TGS validates the TGT and the request, then replies with:
  - (a) A *Service Ticket*, encrypted with the service's secret key, containing user group info, a session key, and a timestamp.
  - (b) A copy of the session key, encrypted with the key obtained in Step 3.
5. Finally, the client sends a request to the target server along with the Service Ticket. The server decrypts the Service Ticket using its shared key with the KDC. If successful, this confirms the request is authorized by the KDC [15].

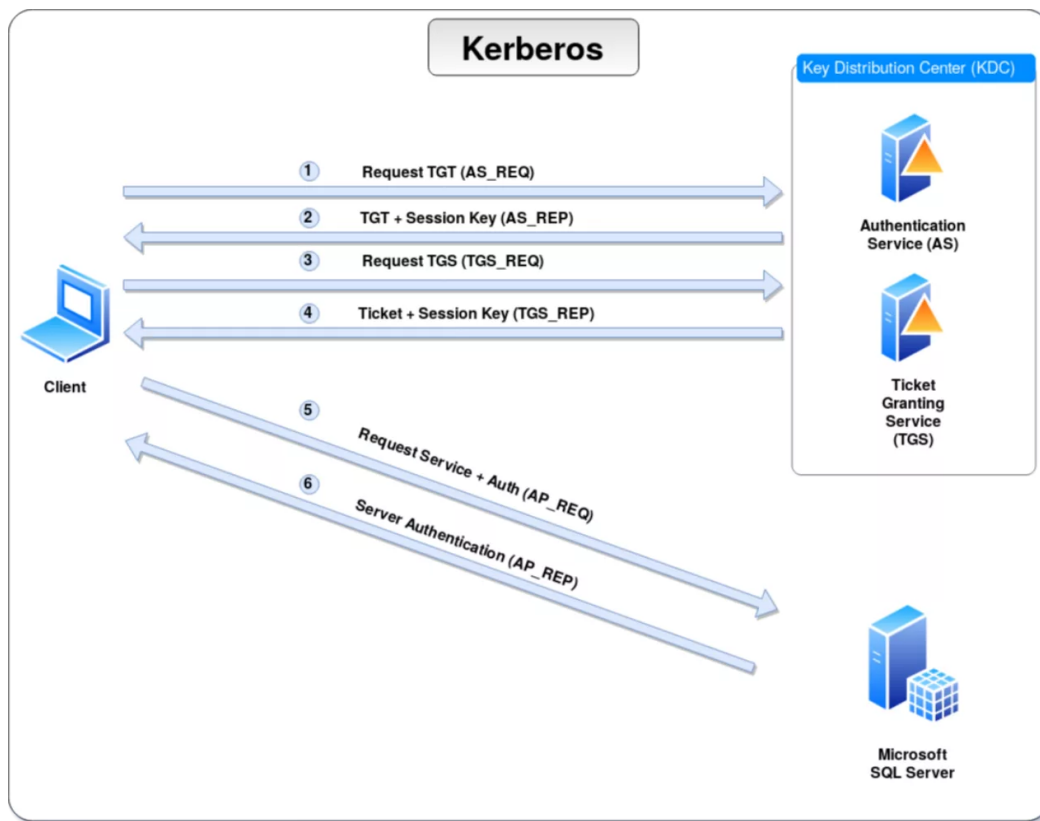


Figure 2.2. Visual representation of the Kerberos authentication workflow.

### 2.3.3 NTLM Authentication

NTLM [38] is a challenge-response authentication protocol that predates Kerberos and remains available for backward compatibility with older systems and applications. Unlike Kerberos, NTLM does not provide mutual authentication and is more susceptible to attacks such as pass-the-hash and relay attacks [40]. Although still supported, NTLM is widely regarded as insecure and its use is generally discouraged in modern AD deployments.

### 2.3.4 Authorization Mechanisms

Authorization in AD is achieved through a combination of access control lists (ACLs) and group memberships. Each object in AD is secured with a discretionary access control list (DACL), which defines the permissions granted to users or groups. Administrators can assign rights directly to individual accounts, but best practice is to manage permissions using **security groups**. Security groups simplify access control by allowing privileges to be managed collectively.

Role-based access control (RBAC) in AD is effectively implemented through group nesting and delegation. Administrators can create hierarchical group structures that reflect organizational roles, thereby reducing complexity and ensuring consistent application of permissions. This model also facilitates delegation of administrative tasks by assigning limited rights to junior administrators or specific departments without granting them full domain-level privileges.

### 2.3.5 Tiered Administration Model

One of the most significant defensive strategies introduced by Microsoft to secure AD is the **Tiered Administration Model**. This model aims to reduce the risk of privilege escalation and lateral movement by isolating administrative privileges into distinct security tiers [11].

The model is based on three logical tiers:

- **Tier 0:** Domain controllers, Active Directory forest-level services, and other assets that directly control the security of the enterprise. Accounts with Tier 0 privileges are considered the most sensitive and must be tightly protected.
- **Tier 1:** Enterprise servers and applications. Administrators in this tier are responsible for managing infrastructure such as file servers, application servers, and database servers.
- **Tier 2:** User workstations and devices. Accounts in this tier handle the management of endpoints such as laptops and desktops used by regular employees.

The key principle of the tiering model is that *administrative accounts should never be used to log into systems of a lower tier*. For example, a Tier 0 administrator should not log into a Tier 1 server, and a Tier 1 administrator should not use their credentials on a Tier 2 workstation. This prevents attackers who compromise a lower-tier system from harvesting high-privilege credentials and using them to escalate their access.

By enforcing strict separation of administrative privileges, the tiering model creates barriers that limit the propagation of attacks across an AD environment. In practice, however, many organizations fail to implement tiering consistently, and administrators often use privileged accounts across multiple tiers for convenience. Such deviations undermine the security benefits of the model and leave enterprises vulnerable to attack paths that span across domains and systems. Later chapters will demonstrate how tools such as BloodHound reveal these risky practices by mapping relationships and highlighting escalation opportunities despite recommended tiering policies.

## Chapter 3

# Attacking Active Directory

### 3.1 Initial Foothold into Active Directory

Attacks against AD typically begin with an adversary obtaining a low-privileged account or compromising a workstation that is joined to the domain. This *initial foothold* provides the attacker with the credentials or access required to begin enumerating the environment and moving laterally. Common techniques for gaining this first level of access include social engineering, credential harvesting, password attacks, and exploitation of exposed services.

#### 3.1.1 Phishing and Credential Theft

Phishing remains one of the most common entry points into enterprise networks. Attackers often deliver malicious attachments or links that capture user credentials when victims attempt to log in to a spoofed site. Once harvested, these credentials can be used to authenticate into AD systems if multi-factor authentication is not enforced [15].

#### 3.1.2 Password Attacks

Many Active Directory environments implement protections against brute force attacks, such as account lockout policies, which trigger after multiple failed login attempts on a single account. While effective against traditional brute force, these protections can be bypassed through a technique known as **password spraying**.

Password spraying is a variant of brute force in which an attacker attempts to authenticate using a single commonly used password across a large number of accounts, rather than trying many different passwords against one account. This approach significantly reduces the risk of triggering account lockouts, since only one attempt is made per account in each round [16].

It is clear that the usage of weak passwords greatly increases this technique's success rate. In particular, if the default password for an organization's newly created account is known, one can spray this password across many users until one is found who hasn't yet changed it, thus gaining an initial foothold in the domain.

#### 3.1.3 AS-REP Roasting

Kerberos authentication introduces its own set of weaknesses when improperly configured. In particular, if a user account is configured to not require pre-

authentication, attackers can request an Authentication Service response (AS-REP) from a domain controller without knowing the user's password. The response is encrypted with the user's key and can be subjected to offline brute-forcing until the password is recovered. This technique, known as AS-REP roasting, is a common way to obtain valid credentials in the early stages of an attack [7].

Under normal circumstances, with pre-authentication activated, the user initiates the Kerberos authentication procedure by dispatching an AS-REQ (Step 1 of the Kerberos authentication workflow described in 2.3.2) message to the DC. This message is encrypted with a timestamp, which is further encrypted with the hash of the user's password [7].

## 3.2 Lateral Movement

Once a foothold is established, attackers expand their access by moving laterally across the environment. Lateral movement involves abusing misconfigured permissions, over-privileged accounts, or cached credentials to pivot from one system or user context to another. Even a compromised low-level account can enable an attacker to escalate privileges if security boundaries such as Microsoft's tiered administration model are not enforced [11]. Common techniques include pass-the-hash, pass-the-ticket, Kerberoasting, and exploiting weak access control lists (ACLs).

### 3.2.1 Pass-the-Hash

The pass-the-hash (PtH) attack has been a well-known technique in Windows environments for many years. It relies on the fact that authentication in systems using the NTLM protocol can be performed with an NT hash of a password, without requiring the plaintext password itself. When a user logs into a Windows system, an NTLM hash of the password is generated and stored, but unlike modern password hashing schemes, NTLM does not employ salting to strengthen these hashes. This limitation makes them vulnerable to reuse once extracted [7].

A PtH attack typically involves two stages. First, the attacker extracts NT hashes from a compromised machine. The most common method is credential dumping from the Local Security Authority Subsystem Service (LSASS) process, which stores authentication material in memory to support single sign-on functionality. Tools such as *Mimikatz* are commonly used to retrieve these hashes. In the second stage, the attacker uses the extracted hashes to create valid authentication tokens and impersonate the compromised user, gaining access to resources and hosts across the domain as though the password had been entered [15].

Although Microsoft has introduced mitigations in successive versions of Windows to reduce PtH exposure, new techniques for extracting and reusing hashes continue to evolve, keeping this attack relevant in modern AD compromises [15, 17].

### 3.2.2 Kerberoasting

Kerberoasting is an attack technique that exploits the way Kerberos service accounts are configured in Active Directory (AD). In many environments, services are run under normal user accounts rather than dedicated managed service accounts. When a service runs under such a user account, the service ticket issued by AD for accessing that service is encrypted with the NTLM hash of the account's password.

This creates an opportunity for an attacker to extract the ticket and attempt to recover the plaintext password through offline brute-forcing.

The attack proceeds in three main stages. First, the attacker enumerates accounts in AD with a defined **Service Principal Name (SPN)**, which indicates that the account is used to run a service. Next, the attacker requests a service ticket for the target account from the Key Distribution Center (KDC) using the SPN value. Finally, the attacker exports the ticket to disk and subjects it to offline password-cracking attacks. Since the ticket is encrypted with the account's NTLM hash, a weak or guessable password can be recovered, granting the attacker control of the service account[15].

Kerberoasting is especially dangerous when service accounts have elevated privileges or when passwords are weak or reused. Because the cracking step is performed offline, there is no risk of triggering account lockouts or alerts within the AD environment.

### 3.2.3 Weak Access Control Lists (ACLs)

Access privileges for resources in Active Directory Domain Services are usually granted through the use of an access control entry (ACE). An ACE defines an access or audit permission on an object for a specific user or group. An access-control list (ACL) is the ordered collection of access control entries defined for an object [3].

Each AD object has a **Security Descriptor**, which contains a discretionary access-control list (DACL). A DACL contains a list of ACEs. Each ACE grants or denies a set of access rights to a user or group. The access rights correspond to the operations, such as reading and writing properties, that can be performed on the object.

In theory, ACLs enable fine-grained access control across the directory; in practice, they are frequently misconfigured, granting users or groups far more rights than necessary. For example, if a user has the right to reset the password of another account, they can impersonate that account without knowing the original password. Similarly, if a user can modify the membership of a privileged group, they can add themselves to escalate privileges.

### 3.2.4 Delegation

Delegation is an Active Directory feature that permits the impersonation of an account by users or computers. [18]

In Kerberos, by default, a service can only act as itself. It cannot pretend to be the user when calling another service. But in many enterprise apps, you need a service to “hop” on behalf of the user. For example, a user interacts with a web server, which in turn might have to query an SQL database. This is the “double hop” problem, and it is where delegation comes in.

**S4U2self and S4U2proxy** S4U2self stands for Service for User to Self, which allows a service to obtain a Service Ticket, on behalf of a user (typically called principal) to itself. In other words, it allows a service to request a special forwardable service ticket to itself on behalf of a specific user. This extension can be used to also produce a Service Ticket to oneself on behalf of another domain user. The impersonation of users can then be performed through the S4U2Proxy.

### Unconstrained delegation

Unconstrained delegation is the permission for a computer to impersonate any user to any service. Once it is enabled, any time a user connects to this computer, their TGT is stored in the computer memory for later use [19]. The attacker compromises a computer and tricks any privileged user such as the domain admin to log in on the compromised computer. The attacker then extracts the TGT of the privileged user and uses it to access any service on the domain. The previous scenario is just a simple attack, and other sophisticated attacks may occur by abusing this feature in different ways [20, 15].

### Constrained delegation

Constrained delegation enables administrators to configure which services an Active Directory user or computer account can delegate to and which authentication protocols can be used.

In theory, constrained delegation limits the damage that could result if an AD account is compromised. But constrained delegation can be abused: An adversary who compromises the plaintext password or password hash of an account that is configured with constrained delegation to a service can then impersonate any user in the environment to access that service. For example, if constrained delegation is configured to a Microsoft SQL SPN, an attacker could get privileged access to that database [21].

### Resource-based constrained delegation

In Resource-Based Constrained Delegation (RBCD), the target resource specifies which accounts (computers or services) are allowed to delegate to it, by setting the msDS-AllowedToActOnBehalfOfOtherIdentity attribute [22]. If an attacker can write to this attribute of a target computer account (e.g., through misconfigured permissions or by controlling another computer account), they can add their own compromised account as a trusted delegator. This allows them to impersonate any user to that target service.

## 3.3 PowerView

PowerView is a PowerShell tool originally developed as part of the PowerSploit framework for AD reconnaissance. It allows the enumeration of domains, users, groups, computers, sessions, trusts, and access control lists.

PowerView provides functions like Set-DomainObject, Add-DomainGroupMember, Get-DomainGroupMember, and Set-DomainUserPassword for manipulating AD objects. It's commonly used in attack scenarios for privilege escalation, group membership changes, and password resets.

## 3.4 BloodHound

BloodHound, developed by SpecterOps in 2016, is a tool designed to automate the enumeration of AD environments and reveal attack paths. It leverages graph theory to model AD as a network of nodes and edges, where nodes represent objects

such as users, groups, and computers, and edges represent relationships such as group memberships, session data, or access control rights.

Data is collected using an ingestor called **SharpHound**, which gathers information from LDAP queries, session enumeration, and group membership data. This information is then stored in a **Neo4j graph database**, allowing analysts to explore the environment using the Cypher query language. By representing AD in this way, BloodHound highlights paths through which an attacker could escalate privileges from a compromised low-level account to a high-value target such as a Domain Administrator.

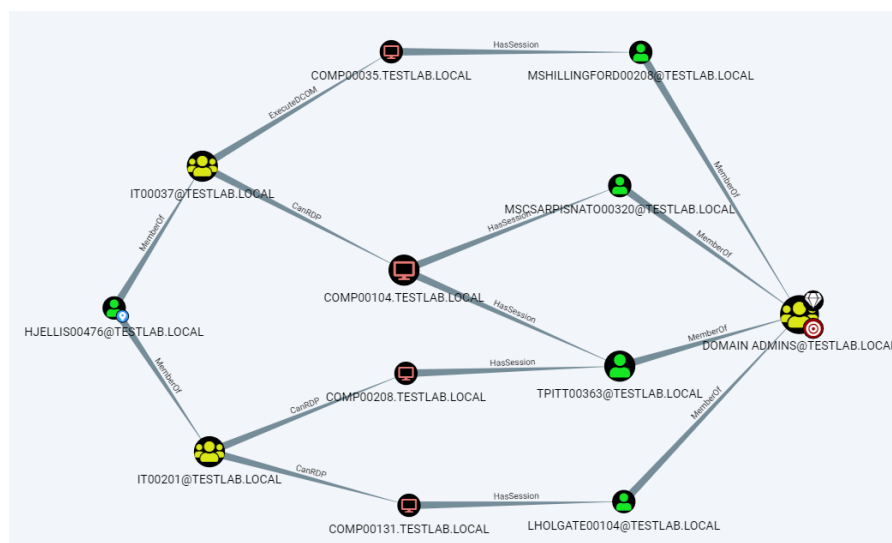
BloodHound is used by both red teams and defenders. For attackers, it provides a rapid method of discovering privilege escalation paths. For defenders, it offers a way to audit and remediate hidden misconfigurations that may otherwise remain undetected in large and complex environments [7].

### 3.4.1 Nodes

In the BloodHound graph model, **nodes** correspond directly to AD objects. Each user, group, computer, domain, or organizational unit in AD is represented as a node in the graph. Importantly, the properties of these nodes are inherited from the original AD objects: for example, a user node will store attributes such as its **security identifier (SID)**, **last logon time**, and **group memberships**, while a computer node may contain information about local administrators or operating system details. By retaining the underlying AD attributes, BloodHound ensures that its graph representation remains faithful to the actual environment.

### 3.4.2 Edges

**Edges** represent the relationships and permissions between AD objects, and thus define how an attacker might move from one node to another. These edges capture both explicit and implicit relationships, such as group memberships, administrative rights, session information, and delegation settings. Examples include **MemberOf**, which connects a user to a group, or **AdminTo**, which connects a user or group to a computer where they hold local administrator privileges. By combining nodes and edges, BloodHound reveals attack paths that emerge from chains of seemingly benign relationships, highlighting how low-level accounts can escalate to high-value targets.



**Figure 3.1.** Example BloodHound legacy graph showing paths from a user object to the Domain Admins group. Each node and edge is labeled.

### 3.4.3 Querying with Cypher

BloodHound stores its data in a Neo4j graph database, which is queried using the **Cypher** query language. Cypher provides a simple but powerful syntax for expressing graph queries, making it possible to search for paths, relationships, and specific object properties. For example, analysts can query for the shortest path between a low-privilege user and a Domain Administrator, or enumerate all computers where a given user has local administrator rights.

A simple example query to find all paths from a specific user to any Domain Admin using only MemberOf or AdminTo edges is shown below:

```
MATCH p=shortestPath(
  (u:User {name:"jdoe@domain.local"})
  -[:MemberOf|AdminTo*1..]->
  (n:Group {name:"DOMAIN ADMINS@domain.local"})
)
RETURN p
```

Despite its strengths, BloodHound has limitations. In large-scale AD forests with tens of thousands of users and millions of relationships, the graphs produced can become extremely complex. Analysts are often confronted with “graph sprawl,” making manual exploration impractical. Sifting through endless paths proves to be an extremely long and, importantly, error prone process. This gap motivates the research presented in Chapter 4, which seeks to automate attack path identification and prioritization to make BloodHound data more actionable for defenders.

## Chapter 4

# Attack path classification

This chapter presents a methodology for **automating the identification and classification of attack paths** in AD. The aim is to move beyond raw graph visualization and provide a scoring-based framework that ranks paths by severity and exploitability. The approach considers two main criteria: (1) the ease with which an attacker can compromise an initial foothold, and (2) the difficulty or cost of traversing the relationships that compose the path. By assigning weighted scores to edges and incorporating contextual factors such as account age or Kerberos pre-authentication, the method prioritizes the most critical paths for analysis and remediation.

### 4.1 Initial Foothold Difficulty

The first criterion in evaluating attack paths is the difficulty of gaining an initial foothold in the domain. In practice, attackers rarely begin with direct access to highly privileged accounts; instead, they compromise low-privilege users or workstations and escalate from there. Assessing the susceptibility of these starting points is therefore essential to understanding the overall risk of a path.

Several user attributes were selected as indicators of foothold difficulty:

- **AS-REP roastability:** If a user account does not require Kerberos pre-authentication, it is vulnerable to AS-REP roasting, allowing attackers to request and crack encrypted responses offline.
- **Password age:** Accounts with very old passwords are more likely to be weak or compromised, while recent password changes suggest stronger protection.
- **Last logon:** Accounts that have logged on recently will more likely pass under the radar when receiving a malicious login, when compared to accounts which haven't received a login in a long time and suddenly do.

These attributes are combined into a scoring model that assigns points based on the relative ease of compromise. The cumulative score is then mapped to qualitative severity levels ranging from *Critical* to *Very Low*.

**Severity Scoring Process.** As we said, the initial foothold severity is determined by combining three account features: AS-REP roastability, password weakness or age, and recent logon activity.

For example, an account that is roastable, weak, and recently active will have a high criticality, since it offers an attacker both technical vulnerability and operational cover. Conversely, accounts that are not roastable, have strong or recently changed passwords, and show little or no recent logon activity have low criticality. Table 4.1 summarizes this process.

Condition	Severity Score
AS-REP roastable, weak password, recent logon	Critical
AS-REP roastable, weak password, no recent logon	Very High
Not roastable, weak password, recent logon	High
AS-REP roastable, recent password, no recent logon	Medium
Not roastable, weak password, no recent logon	Low
All other cases (strongest accounts)	Very Low

**Table 4.1.** Foothold severity scoring logic based on account attributes.

This scoring framework provides a theoretical basis for ranking user accounts as potential entry points into AD. The implementation details, including the specific queries used to extract these attributes from the BloodHound database, are described in Section 4.2.

## 4.2 Traversing the path

Once an attacker has obtained an initial foothold in the domain, the next challenge is to traverse the environment toward more privileged accounts, critical assets or, in general, any target that is necessary in achieving the attacker's goal. This process is determined more often than not by the sequences of relationships and permissions that link objects together. Each step along an attack path represents a traversal of an edge in the BloodHound graph, such as membership in a group, local administrator rights on a computer, or the ability to reset another user's password.

The ease or difficulty of traversing a given path depends on the types of edges involved and the context in which they appear. Some relationships, such as direct membership in an administrative group, provide an attacker with immediate and highly valuable escalation opportunities. Others, such as weak ACLs or delegation misconfigurations, require more effort or specialized knowledge to exploit. By abstracting edges into categories of relative difficulty, the model captures the intuition that some relationships are inherently more powerful than others, and that attackers are more likely to choose paths that consist of the most easily exploitable edges.

For their evaluation, we will divide edges in **2 categories**: ones which don't require additional conditions to be true in order to be exploited, with a fixed severity if encountered, and ones that do require it, so their severity will vary based on the aforementioned conditions. Subsequently, **4 severity tiers** will be defined based on the edge in question's ease of exploitation, traces left behind and privileges granted on the target node. Edges of the first category can directly be ranked based on their characteristics, while edges of the second will require additional context relative to the domain in order to be ranked. Finally, only edges considered "traversable" will be considered; if an attack path encounters an untraversable edge, the path will be dropped.

In the remainder of this dissertation, the term **principal** will be used as a generic label for Active Directory objects that can participate in attack paths, including

users, groups, computers, and service accounts. This shorthand follows common usage in the security community, where a principal denotes any identity or object capable of holding permissions or being the subject of access control.

### 4.2.1 Tier 1

The following are edges which grant traversal from the source principal to the target with relative ease and can possibly generate little to no traces. Edges which are similar enough are grouped together.

#### **GenericAll, GenericWrite**

The **GenericAll** permission grants full control over a target object and therefore enables multiple avenues of abuse. Over a user object, an attacker may reset the account's password, add shadow credentials by writing to the `msDS-KeyCredentialLink` attribute [24], or modify the `servicePrincipalNames` attribute to enable targeted Kerberoasting. Over an organizational unit (OU), the attacker may insert new ACEs that inherit down to all contained objects. When applied to a computer object, **GenericAll** allows reading the LAPS password (see the following **ReadLAPSPassword** edge), adding shadow credentials, or performing a resource-based constrained delegation attack. [23]

**GenericWrite** has a very similar abuse pattern.

#### **ReadLAPSPassword**

LAPS (Local Administrator Password Solution) is a tool to manage windows computer local administrator passwords which among other things ensures that they are rotated regularly and are unique for each computer [26]. The **ReadLAPSPassword** edge allows the source principal to read the LAPS password of the target computer, effectively traversing the edge. Reading LDAP properties is extremely low risk [23].

#### **HasSIDHistory**

This edge indicates that, when a Kerberos ticket is created for the source principal, it will include the SID for the target principal, and therefore grant the source principal the same privileges and permissions as the target principal [23]. This is a very easy edge to traverse, as Kerberos tickets created by the source principal already have the same rights as the ones created by the target.

#### **WriteDACL, Owns, WriteOwner**

With the ability to modify the DACL on the target object, you can grant yourself almost any privilege against the object you wish [23]. Ownership of the object also grants the privilege to modify its DACL. Once the rights have been granted, the abuse pattern is very similar to the one described for the **GenericAll** edge.

#### **AddMember, AddSelf, MemberOf**

The **AddMember** edge indicates the principal has the ability to add arbitrary principals to the target security group. **AddSelf** indicates that the source principal can add itself, granting the same privileges as the target group, whereas **MemberOf**

means the source principal is already part of the target group. For the purposes of path traversal, the first two edges are equivalent.

### **AdminTo**

This edge indicates that the source principal is a local administrator on the target computer. From there, often lateral movement consists in impersonating other users logged into the machine, and, as admins, being able to disable host-based protections that might be in place to prevent this [23].

#### **4.2.2 Tier 2**

These are edges which require more complex exploitation and/or leave traces of activity.

### **SyncLAPSPassword**

The `SyncLAPSPassword` edge allows the source principal to read the LAPS password through a directory synchronization, which might generate a 4662 event "An operation was performed on an object" [28] on the domain controller if appropriate SACL [27] are in place.

### **ForceChangePassword**

This edge indicates that the source principal can arbitrarily change the target principal's password [23]. The best method to abuse this edge is to use PowerView's `Set-DomainUserPassword` function. By doing this, a 4724 event "An attempt was made to reset an accounts password" [25] will be generated on the Domain controller that handled this request, thus leaving some traces.

### **AddKeyCredentialLink**

This edge indicates that the source principal can write the `msds-KeyCredentialLink` property of a target user or computer. In the context of PKINIT authentication [29], which uses public key cryptography, this property contains the target's public key used by the domain controller to decrypt authentication data, which, if successful, grants a session ticket. This means that writing the `msDS-KeyCredentialLink` property of a user means you can obtain a TGT for that user [24].

#### **4.2.3 Tier 3**

These are edges which require more complex exploitation, possibly with multiple steps.

### **SQLAdmin**

This edge indicates that the source principal is the account configured to run the SQL server instance running on the target computer. Apart from accessing the database itself, various techniques can be employed to run operating system commands through the SQL Server. An example using the `Invoke-SQLOSCcmd` cmdlet of the PowerUpSQL toolkit to execute the `Whoami` command:

```
Invoke-SQLOSCmd -Verbose -Command "Whoami" -Threads 10  
-Instance sqlserver\instance
```

The privileges granted when executing commands depend on the privileges the Admin service account has, but often these are granted local administrator privileges [23].

### ExecuteDCOM

The **ExecuteDCOM** relationship indicates the ability to instantiate a COM object on a remote host and invoke its methods via DCOM/RPC. Under certain conditions, for example when the remote COM class runs with elevated privileges, exposes unsafe methods, or processes attacker-controlled input; this can lead to remote code execution on the target machine. Practical examples include abusing remote MMC, Excel, and other COM servers that accept remote activation; an attacker who can trigger such instantiations may execute code in the context of the COM server (potentially with high privileges) [30, 31, 32].

Network-level mitigations (e.g., Windows Firewall blocking DCOM/RPC) can prevent the practical exploitation of relationships represented in the BloodHound graph; however, such mitigations do not remove the underlying permissions or registrations and therefore do not necessarily cause the corresponding graph edge to disappear.

### AllowedToDelegate

This edge indicates that the source principal is configured for constrained delegation on the target computer. For a detailed explanation on constrained delegation, see subsection 3.3.4. In theory, the source principal can only impersonate other principals for the specific services specified on the target, but an issue exists where the service name (**sname**) inside the delegated Kerberos ticket isn't protected, so an attacker can change it to any service. So even if the **msds-AllowedToDelegateTo** property on the target principal lists **HTTP/host.domain.com**, the ticket can be altered to target **LDAP/host.domain.com**, **HOST/host.domain.com**, etc., enabling full compromise of the server regardless of the originally allowed service [23]. Thus, after compromising the source principal, an attacker must impersonate a principal with admin rights on the target computer. Finally, the impersonated user cannot be in the "Protected Users" group.

#### 4.2.4 Tier 4

These are edges which do not guarantee privilege escalation, therefore traversing them does not ensure that the following principal in the path will be reachable.

### CanRDP

This edge indicates that the source principal can start a remote desktop session on the target computer with the same privileges as the source. On a workstation with an active user, impersonating the same user takes over their session and kicks them off, while impersonating a different user will prompt you to force the current user off before logging in [23].

### CanPSRemote

PowerShell remoting (New-PSSession / Invoke-Command) opens an interactive session on a remote host using supplied credentials, enabling remote command execution and, possibly, follow-on escalation; it should be used cautiously because it generates logon events and is subject to detection via script-block logging and AMSI [33].

#### 4.2.5 Edges requiring additional conditions

As mentioned above, the following edges' severity may vary based on additional factors.

#### AllowedToAct, AddAllowedToAct, WriteAccountRestrictions

The `AllowedToAct` edge indicates that the source principal is configured for resource-based constrained delegation on the target principal. For more information on RBCD, see subsection 3.3.4. This edge's severity can vary based on the services running on the target principal; specifically, it inspects the target's Service Principal Names (SPNs). In detail:

1. **HOST/ service:** The HOST/ SPN represents the machine account and host-level services; impersonating to HOST/ typically yields machine-level authentication and enables actions that lead to SYSTEM-level compromise, so it is put in tier 2.
2. **WSMAN/ service:** WSMAN allows for PowerShell remoting on the target, which, if run impersonating an admin, enables total compromise; so it is put in tier 2.
3. **RPCSS/ service:** RPCSS brokers RPC/DCOM endpoints and enables remote service control and COM activation; if impersonation grants access to privileged RPC or COM interfaces, this often allows service creation or privileged code execution as SYSTEM; so it is put in tier 3, similarly to `ExecutedDCOM`.
4. **CIFS/ service:** CIFS (SMB) exposes file shares and administrative shares on the target; impersonation that grants CIFS access typically allows reading/writing system files, dropping or replacing service binaries, and staging payloads that lead to service restart or execution as SYSTEM; so it is put in tier 3.

The `AddAllowedToAct` edge indicates that the source principal can add another principal for RBCD on the target principal.

Similarly, the `WriteAccountRestrictions` edge indicates the principal has the ability to modify several properties on the target principal, most notably it can also add another principal for RBCD on the target.

#### AllExtendedRights

Extended rights are special rights granted on objects which allow reading of privileged attributes, as well as performing special actions [23]. This edge's effect varies based on the target principal.

1. If the target principal is a **User**: The edge grants the ability to change the password of the target user, making it similar to the `ForceChangePassword` edge; thus, it is placed in tier 2.
2. If the target principal is a **Computer**: The source principal can read the LAPS password of the target computer, making it similar to the `ReadLAPSPassword` edge and placing it in tier 1.

#### 4.2.6 DCSync edge

Although this work focuses on paths that terminate at privileged groups (e.g., `Domain Admins`), the analysis explicitly considers `DCSync` edges. `DCSync` represents an action by which an account with specific directory rights can replicate (and thereby extract) credential material for arbitrary domain principals. Because `DCSync` effectively yields domain-wide secrets, including those of high-value groups, it functions as a practical terminal on many attack paths.

When this edge is encountered, path analysis is terminated. The edge itself is of the highest severity.

### 4.3 Implementation

This section describes a possible implementation of the attack-path identification pipeline introduced in the previous section. The implementation ingests BloodHound's Neo4j dataset collected from a controlled Active Directory lab (generated via `BadBlood` [34]), enumerates candidate paths to privileged groups, and applies the foothold and traversal scoring models to rank these paths. We outline the test environment, data-collection steps, database setup, path-enumeration strategy, scoring engine, and the visualization and reporting mechanisms used to present results to analysts.

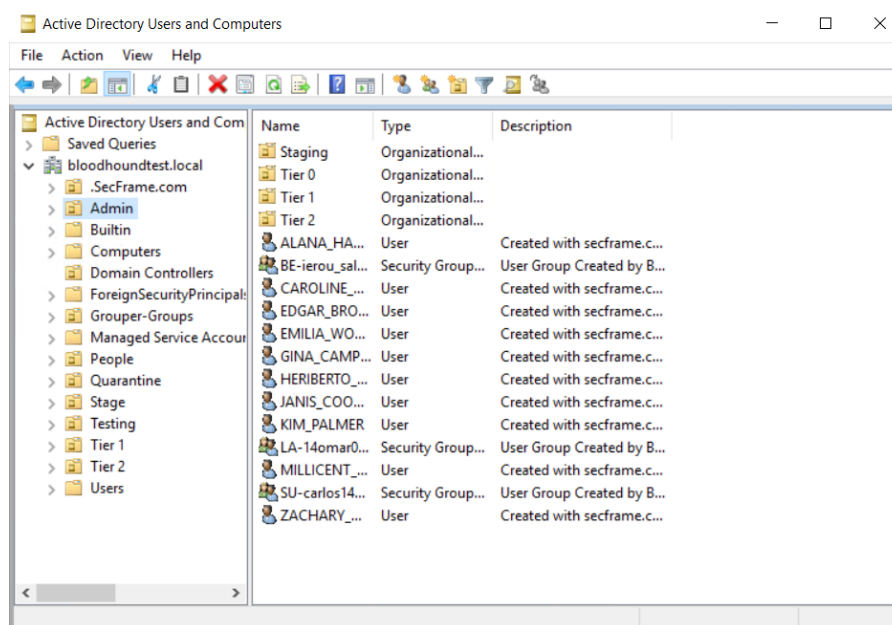
#### 4.3.1 Test Environment / Dataset

Experiments were performed on a controlled, synthetic Active Directory environment created with the `BadBlood` [34] script. The generated domain aims to be realistic: it contains multiple organizational units, nested and builtin groups, tiered administrative accounts, domain controllers, service accounts, and a mix of workstations and servers so as to simulate a typical enterprise deployment.

Figure 4.1 shows the AD object counts (users, computers, groups) obtained via PowerShell commands, and Figure 4.2 is a screenshot of the Active Directory Users and Computers (ADUC) console from the lab environment to illustrate the look-and-feel of the dataset.

```
PS C:\Users\Administrator> (Get-ADUser -Filter *).Count
2493
PS C:\Users\Administrator> (Get-ADGroup -Filter *).Count
548
PS C:\Users\Administrator> (Get-ADComputer -Filter *).Count
100
```

**Figure 4.1.** AD object counts (users, groups, computers) obtained with Get-AD PowerShell commands.



**Figure 4.2.** Representative screenshot of Active Directory Users and Computers (ADUC) in the lab domain.

### 4.3.2 Data Collection

Data for the analysis was collected from the BadBlood lab domain using SharpHound (section 3.5). SharpHound was executed from a domain-joined host running with a low-privilege account; this mirrors realistic attacker behaviour and demonstrates that extensive enumeration is possible without privileged credentials. The collection run produced a set of JSON files containing nodes and relationship data exported by the various SharpHound collection modules.

The JSON output was then imported into the BloodHound Legacy application, which populated a Neo4j graph database for subsequent querying and analysis. BloodHound Legacy was chosen for ingestion because it offers a wide set of built-in queries which facilitate exploration and isolation of paths.

**Rationale for low-privilege collection.** Running SharpHound from a low-privilege account reflects a common attack surface: many enumeration sources in AD are exposed to normal domain users, which allows effective graph construction without elevated rights. Demonstrating successful collection from a non-privileged host strengthens the practical relevance of the subsequent path analysis.

**Format and ingestion.** SharpHound’s JSON files were ingested into BloodHound Legacy using its import facility, which converts the JSON artifacts into nodes and edges in Neo4j. The resulting graph preserves AD attributes (SIDs, SPNs, group memberships, ACLs, session information, etc.).

### 4.3.3 Path Enumeration

Path enumeration is the core stage of the process: given a set of candidate starting principals (i.e., users judged as potential footholds), the implementation

enumerates all relevant paths from each principal to one or more designated high-value targets (for this work, privileged groups such as `Domain Admins`) and then submits each path to the scoring engine described in 4.2.

**Architecture overview.** The enumeration was implemented in Python using the official `neo4j` driver. The program connects to the BloodHound/Neo4j instance, obtains a list of candidate users in bounded ID ranges (to allow chunked parallel processing), and for each user retrieves the set of graph paths that reach the target group within a configurable minimum and maximum path length. Path retrieval and scoring are executed in parallel via a thread pool to maximize throughput while avoiding excessive single-thread blocking on the database connection.

**Selection of candidate users.** Rather than enumerating every node indiscriminately, the pipeline first selects users that are enabled, not marked as high-value (as these are likely much more protected from external attacks), and that have at least one path to the target group within the configured path length bounds.

Example cypher query:

```
MATCH (g:Group)
WHERE id(g) = $gid
MATCH(u:User)
WHERE $min_id <= id(u) AND id(u) < $max_id
AND u.enabled = true AND NOT u.highvalue
AND EXISTS {{
    MATCH (u)-[*{min_l}..{max_l}]->(g)
}}
```

where `gid` indicates the group ID of the high-value group to be reached and `min_l` and `max_l` indicate the minimum and maximum length of the path. `min_id` and `max_id` are used to separate the users in batches for parallel processing. Subsequently, User `u` is used for initial foothold enumeration, as described in 4.1.

For this purpose, the BloodHound user node `pwdlastset`, `lastlogontimestamp` and `dontreqpreauth` properties can be used for, respectively, when the password was last set, when was the last time the user logged in, and if they require Kerberos pre-authentication, which, if disabled, allows for ASREP roasting.

**Path retrieval strategy.** Paths were retrieved using Cypher patterns and APOC helpers available in the Neo4j server. In practice the implementation used a variable-length path match to capture all traversals composed of allowed relationship types, with APOC used to exclude paths that contain cycles, since real AD graphs contain nested groups and other structures that naturally produce cycles. This preserves meaningful, acyclic attack sequences while discarding cyclic paths that would otherwise greatly inflate counts while adding no value, because a path with a cycle implies that an attacker would simply be adding hops to an attack sequence. Conceptually the retrieval follows this pattern:

- For each user batch, find all paths  $p : (u : User) - [*\ell_{\min}..\ell_{\max}] -> (g : Group)$  where  $g$  is the target group;

Example cypher query:

```

MATCH p=(u:User {{name:$username}})
-[*{min_l+1}..{max_l+1}]->
(g:Group)
  WHERE id(g) = $gid

```

The double brackets around `username` are because this is a formatted string so it needs to be interpreted as a single bracket. This is because operations have to be performed on `min_l` and `max_l`. 1 has to be added because the user indicates the path length (as in number of nodes in the path) but here the relationships are counted, which are 1 less than the number of nodes.

- For each path, eliminate it if it contains a cycle.

```

AND size(nodes(p)) = size(apoc.coll.toSet(nodes(p)))

```

The above compares the number of nodes to the number of nodes inside the set of nodes, which will be equal if the path contains no duplicate nodes.

- Return the path as an ordered list of nodes and relationships for downstream scoring.

```

RETURN p;

```

**Filtering and length bounds.** Variable-length traversals in large AD graphs grow exponentially with depth; without limits, even modest domains yield an intractable number of paths. To keep enumeration realistic and tractable, this work enforces configurable *minimum* and *maximum* path lengths (measured in relationships/hops). In practice:

- **Minimum length**  $\ell_{\min}$  filters out degenerate cases (e.g., direct membership in `Domain Admins`) which might inflate counts with paths which strictly speaking should be counted, by definition, but in practice might be obvious and offer little insight into actual misconfiguration chains which realistically could have been overlooked. In any case, unless otherwise specified there is no minimum length.
- **Maximum length**  $\ell_{\max}$  caps search depth to avoid combinatorial blow-up; empirically,  $\ell_{\max} \in [\ell_{\min} + 6, \ell_{\min} + 8]$  captures the vast majority of realistic AD attack chains (users rarely execute  $> 8$  distinct pivots in practice). Practical tests in the lab showed that raising  $\ell_{\max}$  after a certain point increased candidate paths exponentially with minimal impact on the top-ranked results; therefore, the default configuration uses  $\ell_{\max} = \ell_{\min} + 6$  unless otherwise specified.

These bounds reflect both *operational realism* (long chains are brittle, noisy, and unlikely to be executed end-to-end) and *computational efficiency* (path counts and runtime rise sharply with depth). Limits are applied during path enumeration so paths outside  $[\ell_{\min}, \ell_{\max}]$  are never materialized.

**Path Scoring** Each enumerated path is assigned a severity score ranging from 0 to 100 that reflects both the ease of compromise of its starting principal and the difficulty of traversing its relationships. In other words, a path's severity is the sum on the difficulty of gaining an initial foothold on its starting node, and the total number and severity of the edges.

Each path starts with 100 points, and has them deducted based on its characteristics. Considering a higher score means the path is more severe, a greater point deduction implies less severe conditions. The first deduction reflects the *initial foothold difficulty*: paths that originate from accounts that are easier to compromise, as described in 4.1, incur lesser reductions, since a weak starting principal makes the path more attractive to an attacker. The score is then adjusted for the path length: every hop imposes an additional point deduction so that, all else equal, shorter paths are preferred as they are both easier to execute and more robust in realistic attack scenarios. Finally, for each edge in the path, the score is reduced according to the specific relationship types encountered along the path: each edge is penalised in proportion to its relative exploitability as defined by the edge classification model described in 4.2. The lower the tier the edge belongs to (meaning, the less severe the edge), the more points will be subtracted. The resulting single score therefore balances ease of entry, traversal effort, and the power of the privileges gained, yielding a ranked list of paths that prioritises the most realistic and dangerous escalation routes.

Edges that, following the scoring model described in 4.2 can be directly inserted in a severity tier, can be assigned a severity score immediately which, as has been stated above, will be subtracted from the cumulative score of the path currently being analyzed each time it is encountered.

In practice:

```
for i, rel in enumerate(path.relationships):
    rel_type = rel.type
    start_name = rel.start_node['name']
    end_name = rel.end_node['name']

    if rel_type in edge_points:
        to_subtract = edge_points[rel.type] # severity points
                                           # in the dictionary
        score -= to_subtract
```

where `edge_points` is a dictionary assigning these edges to their tier/severity points, following the *Path Retrieval Strategy* paragraph logic, and `path.relationships` is the list of relationship objects in the path currently being analyzed. Each relationship object contains its source and target node information.

Conversely, edges which require additional logic will be processed accordingly:

```
elif rel_type == "AllowedToAct" or rel_type == "AddAllowedToAct"
or rel_type == "WriteAccountRestrictions":
    to_subtract = allowed_to_act(end_name)
    if to_subtract is None:
        break
    score -= to_subtract
```

```

elif rel_type == "AllExtendedRights":
    to_subtract = all_extended_rights(rel.end_node)
    if to_subtract == -1:
        break
    score -= to_subtract

```

AllowedToAct, AddAllowedToAct and WriteAccountRestrictions are grouped together following the logic in subsection 4.2.5.

Possible implementation:

```

#return score based on what SPNs are set on the target node and what
#you can do by impersonating an admin to them

def allowed_to_act(target_node):
    score = -1
    spns = target_node["servicePrincipalName"]
    if not spns:
        return score

    for spn in target_node["servicePrincipalName"]:
        if spn.startswith("HOST/"):
            # assign HOST severity
        elif spn.startswith("WSMAN/"):
            # assign WSMAN severity
        elif spn.startswith("CIFS/"):
            # assign CIFS severity if a higher severity hasn't
            # already been assigned
        elif spn.startswith("RPCSS/"):
            # assign RPCSS severity if a higher severity hasn't
            # already been assigned

    return score

```

**DCSync.** This edge is a terminal edge, as described in 4.2.6. On encountering this edge, paths are not expanded beyond it.

**De-duplication and path signatures.** Paths that are structurally identical but discovered from different starting enumeration orders are de-duplicated. The implementation computes a compact *path signature* comprised of the ordered sequence of node identifiers and relationship types; signatures are stored in a set to filter duplicates before presentation.

**Parallelism and batching.** User enumeration and path retrieval are parallelised by chunking user id ranges and processing chunks via a thread pool. For this purpose, Python's `ThreadPoolExecutor` [35] was used. `ThreadPoolExecutor` is well-suited for parallelizing I/O-bound operations because it manages a pool of worker threads that can execute tasks concurrently without the overhead of creating and destroying threads for each job. Since I/O tasks (like retrieving data from a database) spend much of their time waiting for external resources, the Global Interpreter Lock in

Python does not significantly limit performance, allowing multiple threads to make progress while others are blocked.

In the first phase, tasks are submitted to the executor which finds initial footholds and their severity for various user ID ranges, allowing multiple user chunks to be fetched concurrently. The results are gathered as soon as each thread finishes, so faster tasks contribute earlier without waiting for slower ones. Once the user list is built, the second phase submits one task per user batch to the user processing function, which as we mentioned previously will find paths to the target group for a user and compute their severity, again running these computations in parallel. By iterating with `as_completed` [35], results are collected in the order they finish, which improves efficiency and balances load across workers.

Tests in the lab showed that, with  $\ell_{\min} = 3$  and  $\ell_{\max} = 7$ , program execution time was reduced from **~20 seconds** to **~8 seconds** using 6 workers.

#### 4.3.4 Results and Example Output

To evaluate the automated classification system, the program described in 4.3 was executed on the BadBlood-generated Active Directory environment, as described in 4.3.1. Enumeration produced a Neo4j graph containing all user, group, and computer objects, as well as their relationships as discovered by SharpHound. The automated pipeline then identified enabled, non-high-value users with valid paths to privileged groups within the configured path length range. For each user, the system computed an initial foothold severity based on the heuristics defined in 4.1 and then retrieved and scored all valid escalation paths leading to the target group, assigning to each path a severity score as described in the **Path Scoring** paragraphs.

The analysis produced several hundred distinct paths prior to deduplication, which was reduced to a concise list of unique escalation chains after applying path-signature filtering to eliminate structurally identical paths. The resulting output was automatically formatted into an HTML report, containing ranked paths and per-path severity scores. Figure 4.3 shows an excerpt from this report, including three example paths visualised by the system. In this case,  $\ell_{\min}$  and  $\ell_{\max}$  were set to 3 and 6, respectively. The high value target to be reached is the **Domain Admins** group.

**Path 3. Severity: 79**

User CLIFFORD\_MARQUEZ@BLOODHOUNDTEST.LOCAL initial severity: **Low**

**Path 4. Severity: 77**

User STEWART\_FOWLER@BLOODHOUNDTEST.LOCAL initial severity: **Low**

**Path 5. Severity: 71**

User LORNA\_LYNN@BLOODHOUNDTEST.LOCAL initial severity: **Low**



**Figure 4.3.** Excerpt from the generated report showing three paths discovered in the BadBlood domain (4.3.1). Each path is labelled with its severity score and initial user severity.

**Example paths.** Path 3 (the first path of the above three) originates from the domain user CLIFFORD\_MARQUEZ@BLOODHOUNDTEST.LOCAL, who was assigned a **low** foothold severity according to the logic in 4.1.

From this starting point, the script discovered an escalation chain consisting of only two hops. The path itself is short and consists exclusively of low-cost, high impact edges, whereas the user has a low initial foothold severity, earning the path a severity score of 79/100.

The second and third path demonstrate how path analysis ends once the DCSync edge is encountered, regardless of the target group, for the reasons stated in 4.2.6.

Taking the second path as example, we can verify this path actually exists by looking it up on BloodHound. To do this, we can find paths from STEWART\_FOWLER@BLOODHOUNDTEST.LOCAL to BLOODHOUNDTEST.LOCAL (the name of the domain) , which in fact yields the above path:



**Figure 4.4.** The path found by the program visualized in BloodHound.

Attack paths with higher criticality were found by setting the target node to `USERS@BLOODHOUNDTEST.LOCAL`:

**Path 1. Severity: 95**

User WARREN\_TAYLOR@BLOODHOUNDTEST.LOCAL initial severity: **Very high**

**Path 2. Severity: 95**

User WARREN\_TAYLOR@BLOODHOUNDTEST.LOCAL initial severity: **Very high**



**Figure 4.5.** Excerpt from the generated report showing the top two most severe paths discovered in the BadBlood domain with this target group.

Path 1 originates from the user `WARREN_TAYLOR@BLOODHOUNDTEST.LOCAL`, who is assigned a *very high* initial foothold severity, and the edges of which it is comprised of are of high severity, earning it a total severity score of 95/100. Path 2 follows a similar pattern, originating from the same user but traversing an alternative route.

These critical attack path could have easily gone unnoticed using manual exploration of paths with BloodHound.

## Chapter 5

# Conclusion

The aim of this dissertation was to design and implement a method for automatically identifying and prioritising attack paths in Active Directory environments. To achieve this, heuristics were developed to evaluate both the difficulty of compromising an initial foothold and the relative cost of traversing different edge types in the BloodHound graph. By combining these two factors into a unified severity score, the system highlights paths that are both easy to initiate and highly damaging if left unaddressed.

In closing, the work demonstrates that automated path scoring and classification can enhance the practical utility of BloodHound in defensive contexts. While developed in a synthetic lab domain, the approach shows how heuristic-driven analysis can help defenders reduce complexity and target their efforts where they matter most.

# Bibliography

- [1] P. Svensson, “*Mastering Active Directory.*“, Birmingham, UK: Packt Publishing, 2021.
- [2] D. Mistry, “*Learn Active Directory Management in a Month of Lunches.*“, Shelter Island, NY: Manning Publications, 2019.
- [3] Microsoft, Active Directory official documentation, <https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview>
- [4] “*Cost of a Data Breach Report 2022.*” [Online]. Available: <https://www.ibm.com/reports/data-breach>
- [5] Semperis, “*Organizations Increasing Investment in Active Directory Security.*” [Online]. Available: <https://www.semperis.com/blog/organizations-increasing-investment-active-directory-security/>. [Accessed: Aug. 22, 2025].
- [6] Optiv, “*New Ransomware Statistics Reveal Gaps in Active Directory Resilience.*” [Online]. Available: <https://www.optiv.com/insights/discover/blog/new-ransomware-statistics-reveal-gaps-active-directory-resilience>
- [7] Picus Security, “*The Complete Active Directory Security Book: Exploitation, Detection, and Mitigation Strategies.*“, 2023. Available: <https://www.picussecurity.com/resource/handbook/the-complete-active-directory-security>
- [8] Petri IT Knowledgebase, “*Active Directory Ransomware Report 2024.*” [Online]. Available: <https://petri.com/active-directory-ransomware-report-2024/>.
- [9] G. McDonald, P. Papadopoulos, N. Pitropakis, J. Ahmad, and W. J. Buchanan, “*Ransomware: Analysing the Impact on Windows Active Directory Domain Services.*” Available: <https://arxiv.org/abs/2202.03276>.
- [10] N. Shah, G. Ho, M. Schweighauser, M. H. Afifi, A. Cidon, and D. Wagner, “*A Large-Scale Analysis of Attacker Activity in Compromised Enterprise Accounts.*” Available: <https://arxiv.org/abs/2007.14030>.
- [11] T. Schroder and S. K. Park, “*Securing Sideways: Thwarting Lateral Movement by Implementing Active Directory Tiering.*” Available: <https://arxiv.org/abs/2508.11812>. [Accessed: Aug. 22, 2025].

- [12] L. A. Meyer, S. Romero, G. Bertoli, T. Burt, A. Weinert, and J. L. Ferres, “*How Effective is Multifactor Authentication at Deterring Cyberattacks?*” Available: <https://arxiv.org/abs/2305.00945>.
- [13] B. Desmond et al., *Active Directory, Fourth Edition*. Sebastopol, CA: O’Reilly Media, 2008.
- [14] J. G. Steiner, C. Neuman, and J. I. Schiller, “*Kerberos: An Authentication Service for Open Network Systems.*”
- [15] B. I. Mokhtar, A. D. Jurcut, M. S. ElSayed, and M. A. Azer, “*Active Directory Attacks—Steps, Types, and Signatures.*”, *Electronics*, 2022, 11, 2629. Available: <https://doi.org/10.3390/electronics11162629>
- [16] OWASP, “*Password Spraying.*” OWASP Cheat Sheet Series, 2023. [Online]. Available: [https://owasp.org/www-community/attacks/Password\\_Spraying\\_Attack](https://owasp.org/www-community/attacks/Password_Spraying_Attack).
- [17] Rights, R.F. *Use Offense to Inform Defense. Find Flaws before the Bad Guys Do*, SANS Institute: Rockville, MD, USA, 2015.
- [18] C. Higgs, “*Authorisation and Delegation in the Machination Configuration System.*”, *LISA*, 2008, 8, 191–199.
- [19] J. DeClercq and G. Grillenmeier, “*Microsoft Windows Security Fundamentals: For Windows 2003 SP1 and R2.*”, Elsevier, Amsterdam, The Netherlands, 2011. ISBN 9780080491882.
- [20] M. Amador, K. Bagwell, and A. Frankel, “*A note on interval delegation.*”, *Econ. Theory Bull.*, 2018, 6, 239. Available: <http://dx.doi.org/10.1007/s40505-017-0133-4>
- [21] J. Dibley, “*Constrained Delegation Abuse: Attacking Constrained Delegation to Achieve Elevated Access.*”, Available: <https://blog.netwrix.com/2023/04/21/attacking-constrained-delegation-to-elevate-access/>.
- [22] J. R. Dora and L. Hluchy, “*Attacks on Active Directory - Resource-based Constrained Delegation and New Patches.*”, *Cybernetics & Informatics (K&I)*, 2025. Available: <https://ieeexplore.ieee.org/document/10916465>
- [23] SpecterOps BloodHound edges documentation, <https://bloodhound.specterops.io/resources/edges/overview>
- [24] ELad Shamir, “*Shadow Credentials: Abusing Key Trust Account Mapping for Account Takeover.*”, <https://posts.specterops.io/shadow-credentials-abusing-key-trust-account-mapping-for-takeover-8ee1a53566ab>
- [25] Ultimate IT Security, “*Windows Security Log Event ID 4724*” <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4724>
- [26] Sean Metcalf, “*Microsoft LAPS Security Active Directory LAPS Configuration Recon*”, <https://adsecurity.org/?p=3164>
- [27] Microsoft documentation, “*Access control lists*”, <https://learn.microsoft.com/en-us/windows/win32/secauthz/access-control-lists>

- [28] Microsoft documentation, *4662(S, F): An operation was performed on an object*, <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4662>
- [29] Microsoft documentation, *[MS-PKCA]: Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol*, [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-pkca/d0cf1763-3541-4008-a75f-a577fa5e8c5b](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/d0cf1763-3541-4008-a75f-a577fa5e8c5b)
- [30] Enigma0x3, *Lateral Movement using the MMC20.Application COM Object*, <https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/>
- [31] Enigma0x3, *Lateral Movement via DCOM: Round 2*, <https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/>
- [32] Enigma0x3, *Lateral Movement using Excel.Application and DCOM*, <https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/>
- [33] Microsoft documentation, *PowerShell security features*, <https://learn.microsoft.com/en-us/powershell/scripting/security/security-features?view=powershell-7.5>
- [34] David Prowe, *BadBlood*, <https://github.com/davidprowe/BadBlood>
- [35] Python documentation, *concurrent.futures — Launching parallel tasks*, <https://docs.python.org/3/library/concurrent.futures.html>
- [36] SpecterOps, *BloodHound*, <https://github.com/SpecterOps/BloodHound?tab=readme-ov-file>
- [37] Microsoft documentation, *Microsoft Kerberos*, <https://learn.microsoft.com/en-us/windows/win32/secauthn/microsoft-kerberos>
- [38] Microsoft documentation, *Microsoft NTLM*, <https://learn.microsoft.com/en-us/windows/win32/secauthn/microsoft-ntlm>
- [39] Catherine A. Meadows, *Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches*, <https://apps.dtic.mil/sti/pdfs/ADA465136.pdf>
- [40] Elad Shamir, *The Renaissance of NTLM Relay Attacks: Everything You Need to Know*, [https://specterops.io/wp-content/uploads/sites/3/2025/04/SPO\\_NTLM\\_WhitePaper\\_Updated.pdf](https://specterops.io/wp-content/uploads/sites/3/2025/04/SPO_NTLM_WhitePaper_Updated.pdf)