International Journal of Information Security
https://doi.org/10.1007/s10207-023-00751-6

REGULAR CONTRIBUTION

Start thinking in graphs: using graphs to address critical attack paths in a Microsoft cloud tenant

Marius Elmiger[1] · Mouad Lemoudden[1] · Nikolaos Pitropakis[1] · William J. Buchanan[1]

Abstract
The challenge of securing IT environments has reached a new complexity level as a growing number of organisations adopt cloud solutions. This trend increases the possibility of overseen attack paths in an organisation's IT infrastructure. This paper proposes a methodology for assessing the security of a Microsoft cloud tenant based on the relationships between different cloud entities through the use of graphs. This paper argues for using graph theory as an effective method to understand and uncover complex entity attack paths. To achieve this, we implemented a graph analytics platform using data from a Microsoft cloud test tenant. Methods based on graph theory proved to measurably reduce possible attack paths. Our research can support defenders who want to better understand the interrelationships of Microsoft cloud entities as well as identify and remediate possible attack paths.

Keywords Graph theory · Attack path · Microsoft cloud · Azure AD · Cloud security · Neo4j · BloodHound

1 Introduction

A growing number of organisations are moving to cloud infrastructures [49]. Microsoft, Amazon, and Google are the most prominent players in the public cloud market. This trend falls into the new emerging model of "Everything-as-a-Service" (XaaS) where "virtualised physical resources, virtualised infrastructure, as well as virtualised middleware platforms and business applications are being provided and consumed as services in the Cloud" [28]. Cloud service providers (CSPs) are usually differentiating between three services models, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) [2, 30, 48]. By adopting a cloud service model, the customer can profit from shared IT responsibilities, which allows them to outsource repetitive tasks like hardware maintenance, OS or application updates. However, the customer is always in the driver's seat with respect to configuring, monitoring, and protecting the cloud tenant independently from the chosen service model.

To provide the appropriate access level to the cloud resources, identities play a crucial role in a cloud environment. "In the cloud, identity is everything" or "Identity is the future" or "Identity is the new perimeter" are just a few quotes from different sources (∼13'900 results on Google) to underpin the importance of identity management in the cloud. Therefore, organisations must be aware of the Authentication, Authorisation and Accounting (AAA) principle to establish an identity-centric perimeter that protects entities with appropriate controls.

In this paper, the focus lies on the Microsoft cloud. The identity provider (IdP) in the Microsoft cloud is Azure Active Directory (Azure AD), responsible for AAA. A compromise of an organisation's Azure AD can lead to full access to all cloud services in the tenant. This is comparable to a compromise of an Active Directory (AD) forest in an on-premises environment. AD and Azure AD are similar but are built on different technologies. For example, AD uses legacy authentication protocols such as NTLM and Kerberos, while Azure AD uses modern authentication protocols such as OAuth, SAML, and OpenID Connect. Both IdPs share similar challenges to defenders regarding the difficulty to manage, analyse, or audit sequences of permissions or memberships. As a result, adversaries often utilise such sequences

✉ Mouad Lemoudden
m.lemoudden@napier.ac.uk

Marius Elmiger
marius@elmigers.ch

[1] School of Computing, Edinburgh Napier University, Edinburgh EH10 5DT, UK

Published online: 19 September 2023

to compromise IdPs such as Azure AD or Active Directory [9, 14, 48]. Microsoft offers various reports regarding the hygiene of privileged Azure AD roles or permissions as a countermeasure [36, 40, 41]. While these reports are helpful, they represent a list of improvements and are not providing a holistic view of sequences of privileges in a tenant. With the help of graph theory, this paper aims to overcome the paradigm of "think in lists" by identifying and visualising privileged identity attack paths in an Azure AD tenant.

1.1 Contributions

This paper offers a practical approach for integrating, representing, and analysing Microsoft cloud tenant attack paths with the help of graph theory. The following contributions were made:

- A methodology for how a graph analytics platform can be built to analyse a Microsoft cloud tenant. The methodology can also be used for other solutions.
- A built solution based on Neo4j and BloodHound to perform graph data analysis of ingested data from a Microsoft cloud tenant.
- Practical examples of how graph analytic methods can be used to identify privileged entities in a graph.
- A practical attack path demonstration on the basis of the analysed attack graph.

1.2 Organisation of the paper

The paper is organised as follows: Section 2 will present background on cloud entities, attack graphs, and discuss related work, as well as describing the proposal to find, implement, and analyse new attack paths in a Microsoft cloud tenant. In Sect. 3, the technical details of how the implementation and research were conducted are described. Section 4 discusses the analysis of the attack graph and the resulting attack path. Finally, Sect. 5 concludes the paper by summarising the work undertaken, presenting major findings and future research directions.

2 Material and methods

2.1 Background

In this section, we will present theoretical background on cloud entities and attack graphs, discuss related work, and describe the methods of our approach to find, implement, and analyse new attack paths in a Microsoft cloud tenant.

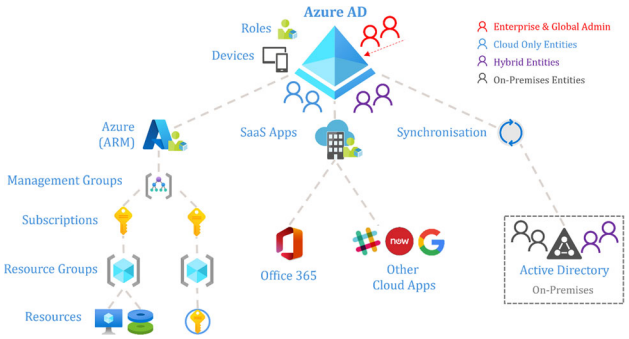

Fig. 1 Azure AD architecture

2.1.1 Microsoft cloud entities

Overall, every CSP provides similar entities in their cloud solution, but the characteristics and management can differ. The identity provider for the Microsoft Cloud is Azure AD (AAD), as depicted in Fig. 1. A Microsoft tenant hosts one AAD instance. AAD is responsible for the tenant's entities by providing the functionality of authentication, authorisation, and accounting [31]. A compromise of an organisation's AAD can lead to full access to all cloud services in the customer's tenant. This is comparable to a compromise of an AD Forest in an on-premises environment. In the following list, we describe the essential entities in the scope of a Microsoft cloud tenant:

- *Roles* grant access to various services like virtual machines, storage accounts, applications, and more.
- *Groups* are used to grouping AAD objects, simplify access control and enable dynamic membership.
- *Personal identities* are generally used by a person to log in to cloud services. Two major types of personal identities exist: cloud-only users and hybrid-identities.
- *Service principals* represent the identity of a AAD cloud application or are also used in IaaS and PaaS to provision new services, manage services, or in general, for automation tasks.
- *Cloud applications* are software applications or services that leverage the capabilities IaaS and PaaS offerings, and they are typically associated with a service principal.
- *Endpoints* are desktops, laptops, or other devices communicating over a network with the tenant.
- *Azure management groups, subscriptions, and resource groups* are used to establish a hierarchy for policy assignment and for billing purposes.
- *Azure resources* are always assigned to resource groups and have different configuration plane and security configuration possibilities.

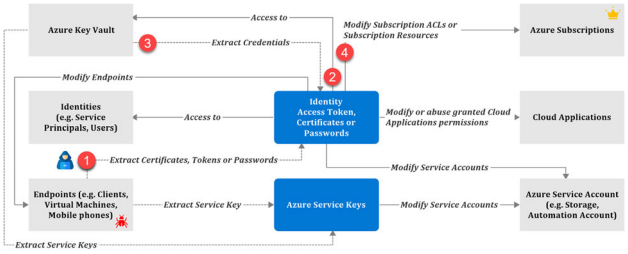

Fig. 2 Example of a simplified pivoting graph in a Microsoft cloud tenant

2.1.2 Graphs

After having identifying the most valuable Microsoft cloud entities, how can an organisation keep track of possible pivoting points among them? One possibility would be to create a list of critical entities and relationships. However, the list would grow over time in size and complexity and would barely allow an understanding of direct and indirect entity relationships. Another approach is to use a graph. Graphs have, besides visualisation, other advantages over lists. First, direct and indirect relationships between entities can be evaluated. Second, possible attack paths are calculable and can be weighted with graph algorithms. Third, visualisation or graph algorithms can assist in recognising patterns or possible attack paths [26]. The graph in Fig. 2 depicts a hypothetical pivoting path.

Suppose an adversary stole an access token with Azure key vault access rights by compromising an endpoint (1-2), the adversary would then extract secrets from the Azure key vault (3). Next, in the key vault, we assume that the adversary found a service principal password. Finally, by enumerating the service principal's access rights, the adversary discovers that the service principal has Azure subscriptions Owner rights (4). With that right, the adversary has widespread access to all resources located under this subscription. From here, the adversary can continue with the pivoting loop till the attacker's goal has been reached.

2.1.3 Attack graphs

The literature describes an attack graph as a general technique to model security configurations in an environment with machines connected over a network protected by firewalls, which may be vulnerable to attacks [1, 21, 23, 46, 55, 58–60]. The reviewed attack graph models from the authors incorporate software vulnerabilities, insecure permissions, insecure firewall rules and other security issues. MulVal is an open-source tool that is used in research to visualise and calculate possible attack paths and risk metrics [47, 59].

Sommestad & Sandström [57] did an empirical test of the accuracy of the attack graph results. Two independent Red Teams examined the test environment, which hosted over a thousand virtual interconnected machines. The test results yielded poor attack prediction results. The authors rationalised the results due to the combination of inaccurate vulnerability scans and improper interpretation of the privileges that vulnerabilities grant. They also argue that vulnerability scanners are limited in their accuracy and can only detect approximately half of the vulnerabilities in a network [19, 20]. Researchers often use attack graphs in combination with hosts, firewall rules, and vulnerabilities. However, our research focuses on attack graphs related to identities and their privileges and not vulnerabilities.

Identity Attack Graphs: Dunagan et al. [9] describe how, with the help of graph theory, they identified highly privileged accounts and how, with their method, they could minimise the attack surface. They utilised machine learning in the graph analytics process to identify privileged AD login sessions on compromised hosts from which the adversary moved laterally to other hosts. They describe the found attack paths as "identity snowball attacks", nowadays known as lateral movement.

Based on prior work, Ho et al. [18] describe techniques to detect lateral movement centred on commonly available enterprise logs. They built a graph based on machine login activities and identified suspicious login sequences corresponding to lateral movement. An inference algorithm was used to determine the movement to which each login belongs. The paths found by the interference algorithm were then used together with a set of detection rules and a new anomaly scoring algorithm to identify the login paths most likely to reflect lateral movement.

Bouillot & Gras [6] published a paper and a tool that, with the help of Graph theory, allowed to analyse sequences of AD permissions and memberships in order to identify complex attack paths. Based on that study, Robbins et al. released the tool BloodHound [53]. The tool is maintained until today, incorporates the ideas from Bouillot & Gras, and was extended with additional complex AD relationships to identify sophisticated attack paths.

Identity Attack Paths: Identity attack paths are the chains of abusable privileges and user behaviours that create direct and indirect connections between entities. The graph in Fig. 3 presents an attack path to obtain tenant administrative rights over an indirect path by abusing the privileges of User 3. A directed edge in the graph means that the from-node can control the to-node. In this paper, we want to reveal such attack paths with the help of graph visualisation and graph analytics.

2.2 Related work

A large number of organisations are using Active Directory for the management and administration of their on-premises

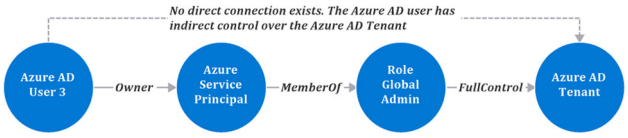

Fig. 3 Attack path example

IT environment. Due to the criticality of Active Directory, security analysis and hardening measures are seen as mandatory. Various methods exist to audit an Active Directory environment, such as vulnerability scanning tools or in the form of security recommendations. However, the complexity and possible blind spots in Active Directory were and are still a major concern. To address this challenge, Bouillot & Gras [6] published a method to assess Active Directory with the help of graph theory. Inspired by this idea, Robbins et al. [53] released BloodHound, a dedicated tool in addition to Bouillot & Gras's solution, which also uses graph theory at its core. Both tools can uncover complex sequences of entity relationships that an adversary could abuse. By discovering such unintended attack paths, an Active Directory security posture can be measurably improved [51]. The same approach can be applied to cloud environments. At the beginning of 2022, two open-source tools exist, namely BloodHound and Stormspotter [3]. Both tools use graph theory to find attack paths in a Microsoft cloud tenant.

2.2.1 Microsoft cloud security

The security responsibilities when using a cloud solution like the Microsoft cloud is shared between the customer and the cloud provider. For example, for identity and directory infrastructure, Microsoft provides AAD as the IdP. Still, the life cycle and access management of identities is the customer's responsibility. From the responsibility model [39], it is apparent that organisations remain with the primary responsibilities such as governance over business-critical processes, data, security monitoring, and many other IT tasks. Unfortunately, this fact can be quickly overlooked by an organisation and thus can lead to a chaotic adoption of cloud solutions [8]. Therefore, it is crucial to avoid repeating known bad practices and invest in a resilient and secure foundation before leveraging the vast, lucrative possibilities a cloud platform can offer [10, 31, 48].

The Microsoft Cloud services are designed with layers of cloud security and support a zero-trust security model. This means that the security of identities, data and applications is not based on the assumption that the network is secure, but rather on the principle that no one should be trusted by default [56]. To maintain a high level of security, regular reviews and adaptations to the hardening measures are essential. To facilitate auditing of Microsoft cloud tenant, various built-in methods are available, including vulnerability scanning

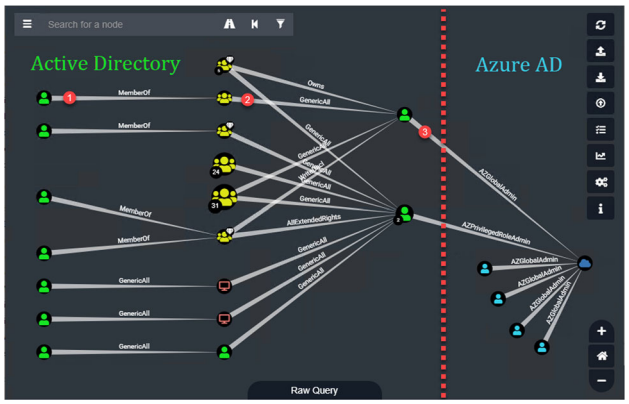

Fig. 4 Attack paths example with BloodHound

tools and security advice [36, 40, 41]. However, just like with Active Directory, the discovering of access relationships between different cloud entities also raises similar concerns regarding complexity and potential blind spots.

2.2.2 BloodHound

The BloodHound tool was created by three red team researchers [53] with the aim of finding attack paths in Active Directory environments. The idea is based on prior work such as [6] and [9].

BloodHound itself is not collecting data. Instead, it is a web interface that visualises AD attack paths in the form of graphs from the ingested data to a Neo4j database. For the data collection, Robbins et al. wrote a separate tool called SharpHound. SharpHound queries data via LDAP from AD controllers and from Windows systems via RPC and SMB. The collected data are stored in JSON files, which can then be imported to a Neo4j database.

For ARM and Azure AD, Robbins et al. [53] created a new collector script named AzureHound. At the time of writing, the script leverages Microsoft PowerShell modules to gather data from a Microsoft cloud tenant. Similar to SharpHound, the data are written to a JSON file. If imported together with AD data, the graph can reveal attack paths in the tenant.

Figure 4 illustrates an attack graph in BloodHound with different paths from AD entities to an organisation's tenant. The red line, the numbered icons, and the green and blue text in the figure are manually added annotations.

To generate the graph above, the shortest path algorithm from any node *n* to node *m*: *tenant* with the length of *3* is used in Listing 1.

```
1 MATCH p=shortestPath((n)-[*1..3]->(m:
    AZTenant {azname:"TEST TENANT"}))
2 WHERE NOT n=m AND (n:AZUser OR n:User)
    RETURN p
```

Listing 1 Cypher query to show the shortest path to an Azure AD tenant

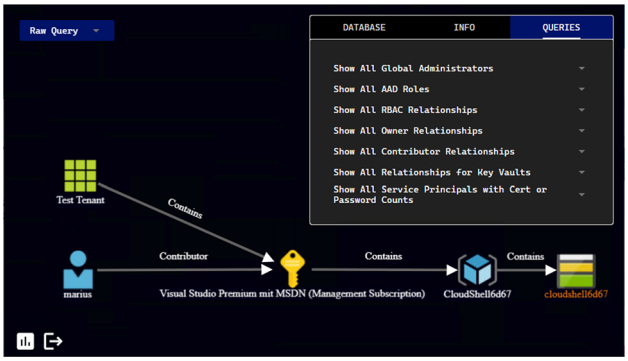

Fig. 5 Attack path example with Stormspotter

The green nodes in the graph are users in AD, the yellow nodes are groups, the blue person nodes are cloud-only users, the cloud symbol is the tenant, and the red icons are computers. The edges between the nodes are *MemberOf*, *AllExtendedRights*, *GenericAll*, *Owns*, *AZGlobalAdmin*, and *AZPrivilegedRoleAdmin*.

A hypothetical attack path could be that an adversary compromises the AD user at step (1). The adversary is then a member of an AD group with *GenericAll* rights over another AD user account. With that right, the adversary can change the password of the AD user in step (2). Finally, in step (3), the adversary authenticates as the newly compromised user and has complete control over the tenant as Global Administrator.

The applied example demonstrates the strength of graph theory in combination with the BloodHound tool. BloodHound 4.0.3 has many edges implemented for AD. In contrast to AD, numerous edges still needed to be implemented to allow a more comprehensive analysis of a Microsoft Cloud.

2.2.3 Stormspotter

Stormspotter visualises attack graphs like BloodHound but focuses entirely on Azure AD and Azure ARM. The tool was released in 2020 by the Microsoft Azure Red Team [3]. Like BloodHound, Stormspotter has two components, namely a frontend and a Python collector script called Stormcollector. This script can collect data over APIs from Azure AD and Azure ARM.

Figure 5 depicts the Stormspotter web frontend and shows an attack graph from a cloud only user to a storage account

The cypher query in Listing 32 was used to show all edges from any entity *n* to the storage account *m* with the name *cloudshell6d67* from the graph database.

```
MATCH (n)-[*1..]->(m:StorageAccount)
    WHERE m.name = 'cloudshell6d67'
    RETURN *
```

Listing 2 Cypher Query to show all edges to storage account cloudshell6d67

The output shows a user, *marius*, which has the *Contributor* rights over the subscription *Visual Studio Premium with MSDN*. The subscription contains a resource group called *CloudShell6d67* with a storage account called *cloudshell6d67*. If an adversary can compromise the user account *marius*, the contributor role grants extensive rights to the adversary to control all resources assigned to the mentioned subscription, which includes the storage account.

Stormspotter 1.0.0b4.4 offers edges focusing on Azure AD and Azure ARM. But still, numerous edges need to be included so that a complete attack graph cannot be drawn. Also, graph algorithms such as shortest path still need to be supported in the web frontend of Stormspotter. To summarise, Stormspotter is an excellent tool for getting an overview of an Azure ARM environment. However, more edges and better support of cypher queries should be added to tap into the full potential of graph analysis possibilities.

2.2.4 Summary

BloodHound 4.0.3 implements a high quantity of nodes and edges to analyse AD. However, for the Microsoft cloud tenant, important edges and nodes such as API permissions, cloud application roles and Azure AD Privileged Identity Management (PIM) roles need to be included. The frontend of BloodHound is stable and fulfils our requirements to analyse attack graphs. On the other hand, Stormspotter 1.0.0b4.4 would offer slightly more edges and nodes regarding ARM but has many gaps in its current version, such as stability, missing Azure AD edges, and supportability of cypher queries in the Stormspotter GUI. Table 1 summarises the comparison of both tools. In conclusion, BloodHound is more suitable for our purposes and will be used for our implementation.

2.3 Methodology

The previous sections provided the theoretical background regarding how graph theory can be used to identify and address critical attack paths in the Microsoft cloud. The following sections describe how the theoretical knowledge was applied to find, implement, and analyse new attack paths in a Microsoft cloud tenant.

Figure 6 represents the stages that were defined to conduct our research. Stage 1 covers the high-level planning, stage 2 implements new nodes and edges and stage 3 analyses and evaluates the newly added nodes and edges. Multiple iterations are possible between stage 2 and stage 3. In the following subsections, the stages are explained in detail.

Table 1 Attack graph tool comparison

Criteria	BloodHound v4.0.3	Stormspotter v1.0.0b4.4
AD	A high number of edges implemented	No Active Directory edges implemented
AAD	Basic nodes and edges are implemented	Basic nodes and edges are implemented
ARM	A few entities are implemented. Comprehensive Graph analysis is limited	Most entities are available, but numerous relationships are missing
CloudApp	Not implemented	Not implemented
Dashboard	Based on Linkurious [29] and compiled with Electron [15]. The web frontend is stable	Based on Vue [25] and the Quasar Framework [50]. The web frontend is not always stable
GraphDB	Neo4j	Neo4j

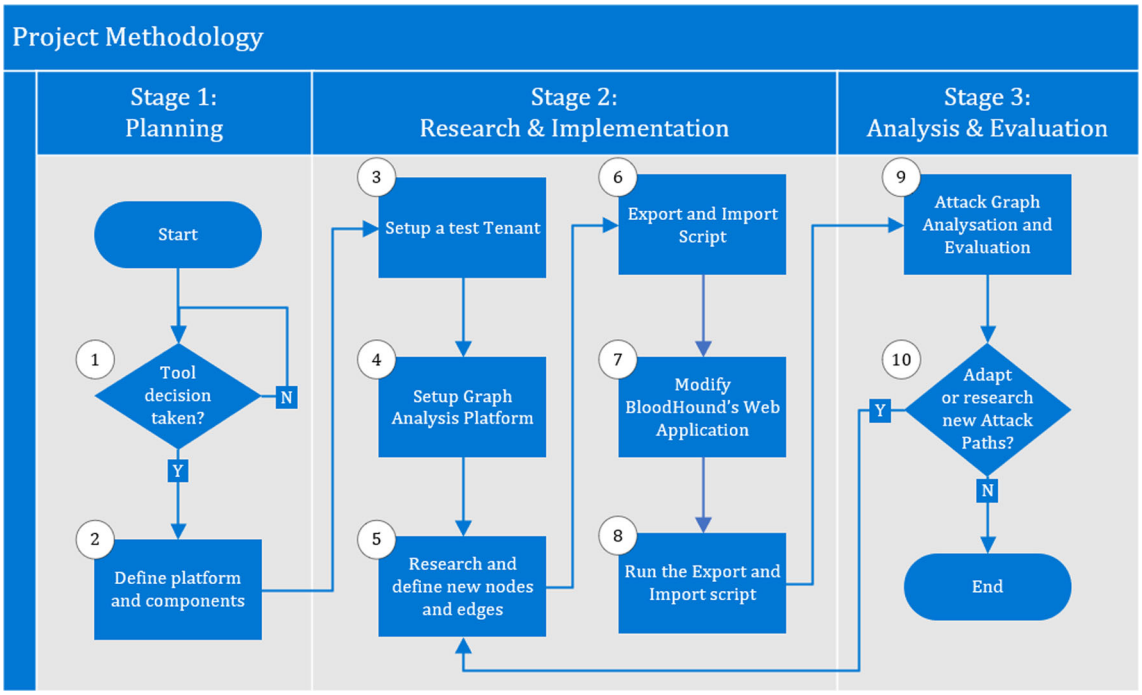

Fig. 6 Project methodology

2.3.1 Stage 1: planning

This stage covers the fundamental decisions on which the implementation and analysis stage depends on.

Step 1. The first step in the workflow was to decide which tool to use to visualise and analyse our attack graphs. Based on the comparisons of both tools, we decided to use BloodHound 4.0.3. BloodHound offers a stable visualisation GUI, using Neo4j as a graph platform, and already has Active Directory and a limited set of the Microsoft cloud tenant nodes and edges implemented. Stormspotter v1.0.0b4.4 could have been an option, but it is, in its current version, not stable enough, has only a few more Microsoft cloud tenant edges implemented compared with BloodHound, and has no nodes or edges for Active Directory.

Step 2. Figure 7 illustrates the components which were required for the research platform. The *first* component of the figure is the Microsoft cloud tenant, from which we extract data and research possible attack paths. Microsoft offers developer subscriptions that are suitable for projects like ours [37]. The *second* component is a script with which we will extract data from the tenant and import the data to the Neo4j database.

The *third* component is Neo4j. Neo4j has two versions—the community edition, which is open source and licensed under GPLv3 and the enterprise edition, which requires a commercial license and offers additional features such as

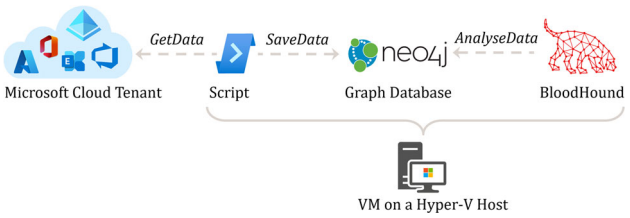

Fig. 7 Research platform

Fig. 8 Translating noun-verb-noun to a graph model

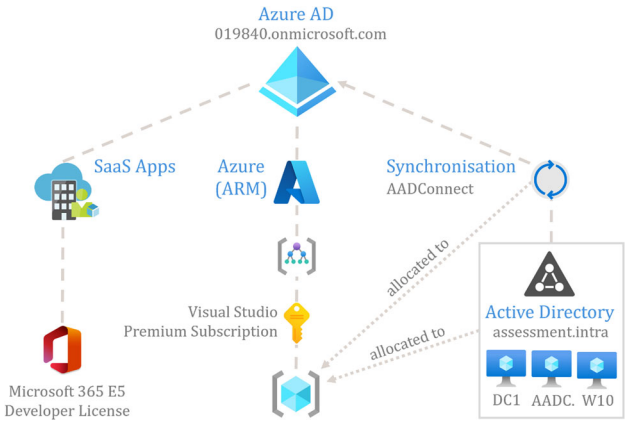

Fig. 9 Overview of the test tenant setup

horizontal scaling, fine-grained access control, high availability, and clustering. We decided to use the Neo4j Desktop Version for our project, which comes with a free developer license for the enterprise edition. The *fourth* component is BloodHound, which can be downloaded from GitHub [53] and can be redistributed and modified under the terms of the GPLv3 or above. The *fifth* and last component is a Windows 10 virtual machine running on a Hyper-V on which we will run the script, the graph database, and BloodHound. A trial license will be used for Hyper-V and the Windows 10 OS. Windows offers all the options to extract data from a Microsoft cloud tenant, such as existing tools and PowerShell modules. Linux could have also been an option but was not considered as more effort would have been required to obtain some data without the already existing community implementations and official Microsoft PowerShell modules.

2.3.2 Stage 2: research and implementation

In Stage 2, the test tenant will be implemented to allow the actual research to map attack paths to entities and integrate the logic into the graph database.

Steps 3–4. During steps three and four, we will set up a Microsoft cloud tenant with a Developer E5 and an Azure ARM subscription. More details on how the setup was done are described in Sect. 5.

Step 5. In this step, we investigated the existing entities in a Microsoft tenant and how they are related to each other. Based on the outcome, we create graph data models by defining nodes and edges by following the rule of thumb from [16, 27]:

1. *If you want to start your traversal on some piece of data, make that data a node.*
   Finding: By studying how a Microsoft cloud tenant works or by creating examples in the test tenant, we can find entities that can be managed by a particular role. The role and the found entities are nodes.
2. *Node-Edge-Node should read like a sentence or phrase from your queries.*
   Finding: An example could be: The role Global Administrator has the ADMIN_TO right over ARM as shown in Fig. 8.
3. *Nouns and concepts should be node labels. Verbs should be edge labels.*
   Finding: For our example, Global Administrator and Azure ARM are the nouns, and ADMIN_TO is the verb.
4. *When in development, let the direction of your edges reflect how you would think about the data in your domain.*
   Finding: By applying the rule of thumbs from the previous rules with the pattern Node-Edge-Node, which is equal to subject-verb-object, we usually can identify the direction. Thus, the edge direction comes from the subject and goes to the object, as illustrated with an arrow in Fig. 8.

Steps 6–8. Based on the required nodes and edges defined in *Step 5*, the export and import script will be adapted and executed. Depending on the modification, the BloodHound Web Application requires adaptation to graphically represent new nodes correctly. Further details of these three steps can be found in Sect. 5.

2.3.3 Stage 3: analysis and evaluation

Stage 3 analyses and evaluates the implementation of the attack paths prepared in Stage 2. If further iterations are required due to adaptations or a new attack path implementation, we will start over again at step 5 in Stage 2. This iteration can be ongoing because of the continuous change in the Microsoft cloud and the discovery of new attack paths. Details regarding the analysis and evaluation are explained in Sect. 4.

Table 2 Graph analysis platform component prerequisites

Applications	Neo4j Desktop 4.4	BloodHound 4.0.3	Scripts
Oracle JDK 11	Required	Not required	Not required
NodeJS 16.13.1	Not required	Optional	Not required
Python 3.10.1	Not required	Not required	Required
Azure CLI PowerShell Module	Not required	Not required	Required
Azure AD PowerShell Module	Not required	Not required	Required

3 Implementation and results

In this section, the technical details of how the implementation and research were conducted are described. The sequence is in alignment with the presented methodology workflow described in the previous section. All scripts used during this implementation can be found on GitHub [12].

3.1 Setup Microsoft cloud test tenant

A Microsoft cloud test tenant was created by following the Microsoft 365 developer Visual Studio guideline [37]. The test tenant comes with a Microsoft 365 E5 Developer license, which includes most of the products Microsoft is offering regarding SaaS applications. For Azure ARM, we are using a Visual Studio Premium subscription, which has a 150 USD limit per month.

To simulate a small company, we configured the tenant according to the Microsoft Azure Active Directory deployment guide [32]. In addition, we also created three virtual machines on Azure ARM to simulate an on-premises Active Directory environment. The identities are synchronised with the Microsoft tool AAD Connect. Figure 9 provides an overview of the test tenant setup.

3.2 Setup graph analysis platform

To set up the specified graph analysis platform, a virtual machine using Windows 10 20H2 as OS was created on an already existing hypervisor. On the virtual machine, we installed the Neo4j desktop application 4.4 according to Neo4j desktop installation guideline [45]. For BloodHound 4.0.3, we followed the BloodHound installation manual for Windows [53]. Table 2 provides an overview of the installed applications, including the prerequisites for the graph analysis platform. NodeJS was required for our project because we must adapt and re-compile the web frontend with new nodes and edges.

3.3 Research

The goal of the research was to gain the required technical knowledge to utilise the implemented graph analytics platform and extend the attack graph provided by BloodHound with additional Microsoft cloud entity edges. The research was conducted along the following lines:

1. Analyse the graph database schema of BloodHound 4.0.3.
2. Analyse the import and export script of BloodHound 4.0.3.
3. Analyse the web interface of BloodHound 4.0.3.
4. Analyse which use cases are already covered by BloodHound 4.0.3 to represent Microsoft cloud entities in a graph.
5. Define new use cases to extend the attack graph.
6. Create a graph database schema that can represent the new use cases.

3.3.1 BloodHound 4.0.3 schema analysis

The BloodHound database schema was analysed with the Hackolade tool and the Neo4j browser. The developers of BloodHound implemented 17 use cases. Examples of the use cases covered by this version are noted in Table 3. Figure 10 depicts the schema.

3.3.2 Modelling the new graph database schema

Based on the knowledge gathered during the literature review and the analysis conducted in the previous section, a graph database schema was created, as depicted in Fig. 11. Compared with BloodHound 4.0.3, the new schema introduces 7 new nodes and redefines the edges to include more granular relationships between cloud entities. The number of edges in BloodHound 4.0.3 is 7, while for our schema, the number has increased to 17 edges. In total, 52 use cases were defined and noted during our knowledge gather. The identified use cases give us a more complete view of the different interactions between the different entities in a Microsoft cloud tenant, specifically Identity-centric and Azure DevOps-related ones. Other nodes that relate to storage services and network components, to name a few, could be added and those would entail new edges. It is expected that more use cases could be identified in the future as new cloud entities and relationships are added and implemented by the cloud provider. Table 4 offers a few examples of the new use cases. One important

Table 3 Example of use cases covered by BloodHound 4.0.3

Use case	Subject	Edge	Object
Azure AD Tenant contains Azure entities	AZTenant	AZContains	All AZ Objects
Azure AD Account can own an Azure AD Group	AZUser	AZOwns	AZGroup
Azure ARM Resource Group can contain ARM resources	AZResourceGroup	AZContains	AZVM,AZKeyVault
Azure AD Account can own a device	AZUser	AZOwns	AZDevice
Azure AD Account is a member of an Azure AD Group	AZUser	MemberOf	AZGroup

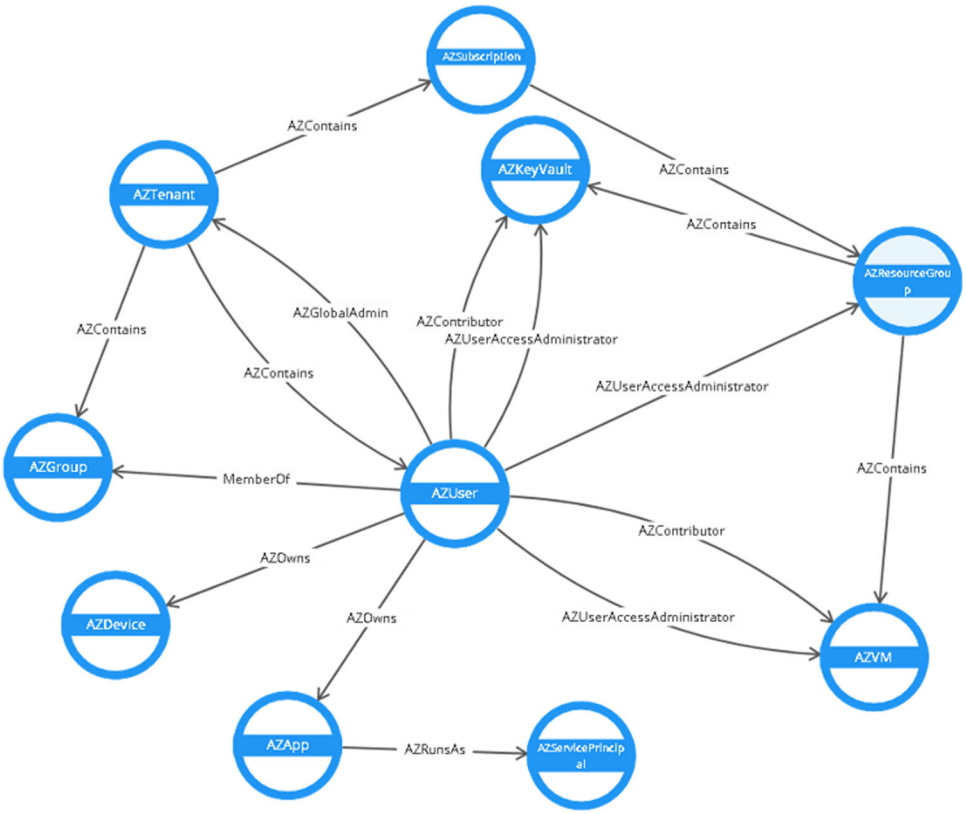

Fig. 10 BloodHound 4.0.3 graph database schema

design difference compared to the BoodHound 4.0.3 schema is that in our schema, roles are represented as nodes and not as edges. The rationale for this decision is to show role-specific attack paths too. For example, at the time of the data collection, an Azure AD user could not be assigned to a role but still be eligible to request one.

3.4 Export and import script

3.4.1 Review export and import script

To export data from the Microsoft cloud, BloodHound comes with a PowerShell script called *AzureHound.ps1*. The script utilises Microsoft Cloud PowerShell modules and stores the acquired data in JSON files.

The import of the data to the Neo4j Database is done over the BloodHound Web Application by using an import option.

3.4.2 Export script creation

We decided to extend the *AzureHound.ps1* script with the required functionality based on our use case definitions. The rationale for this decision was mainly to leverage as many of the default components as possible to save time. During the development, a substantial amount of the existing code was replaced or extended. Eventually, we decided to rename the export script to *AzHound.ps1* to avoid confusion. The process used to export data with *AzHound.ps1* is depicted in Fig. 12.

Figure 12 also shows the Azure AD PowerShell modules that were used to export the required data with 'AzHound.ps1'. To understand the functionality of each module, we consulted the Azure Active Directory PowerShell for Graph module and Azure PowerShell reference documentation from Microsoft [33]. During the development phase, we discovered that the Azure AD modules are

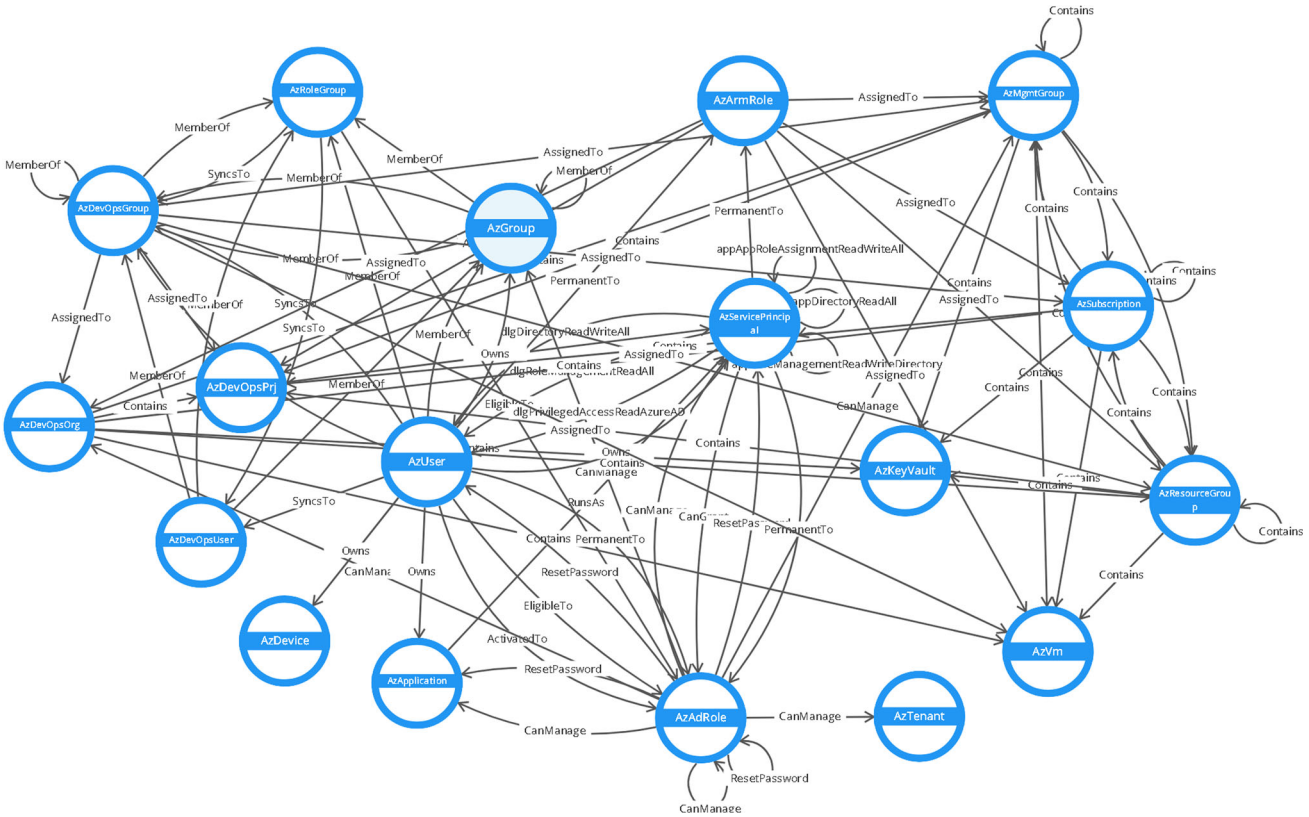

Fig. 11 Final graph analysis platform schema

Table 4 Examples of added use case coverage of the graph analytics platform

Use case	Subject	Edge	Object
Azure AD Account is eligible for an Azure AD role	AzUser	EligibleTo	AzAdRole
Azure AD Account owns an Azure Service Principal	AzUser	Owns	AzServicePrincipal
Azure AD Role Group has a permanent AAD role assignment	AzRoleGroup	PermanentTo	AzAdRole
Azure AD Group can be a member of an Azure Role Group	AzGroup	MemberOf	AzRoleGroup
Azure AD Group can be represented as an Azure DevOps Group	AzGroup	SyncsTo	AzDevOpsGroup
Azure AD Service Principal can manage Azure AD Roles	AzServicePrincipal	CanGrant	AzAdRole
Certain Azure AD Roles can reset passwords of Azure AD Users	AzAdRole	ResetPassword	AzAdUser
ARM Management Groups can contain ARM Subscriptions	AzMgmtGroup	Contains	AzSubscription
AD Service Principals to deploy ARM resources	AzDevOpsPrj	RunsAs	AzServicePrincipal
Azure DevOps Projects or Organisations	AzDevOpsGroup	AssignedTo	AzDevOpsOrg

not offering all attributes that we needed to implement our use cases. Thus, information for privileged identity management and DevOps had to be retrieved over the Microsoft Graph REST API [38] and the Azure DevOps Services REST API [35]. Most of the information could be queried with an unprivileged Azure AD user account. However, to retrieve privileged identity management data, additional permissions had to be granted, such as *PrivilegedAccess.Read* to the user. In a similar way, in order to query data from ARM, at the very least, read rights were required. Summarised, the user who executed the script requires low-level access rights to query all the data. The script then retrieves data from the Microsoft cloud APIs and saves the acquired data to structured JSON files.

The code snippet example, shown in Listing 3, explains the export process of Azure AD users more in detail. On the lines 1–5, the connection is established with Azure AD. If there is no existing token available, the user who executes the script receives a login prompt. On line 7, Azure AD user information is dumped with specific object properties to min-

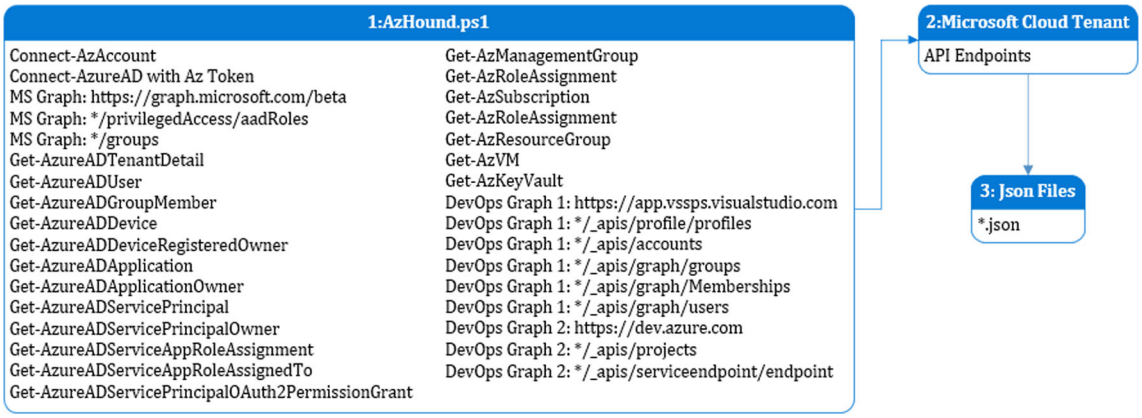

Fig. 12 AzHound.ps1 export process overview

imise the data size. On lines 9–17, every user in the array *$AADUsers* is parsed, and if required, data are manipulated and again added to an array. Lines 18–22 interpret the array and writes a structured JSON file to a defined directory. The script AzHound.ps1 can be found on GitHub [12].

```
1  $LoginStatus = Get-AzContext # Check to see if we are logged in
2  ...
3  Connect-AzAccount # If not we connect with the Az module to the ARM endpoint
4  ...
5  Connect-AzureAD -TenantId $LoginStatus.Tenant.Id -AccountId $LoginStatus.Account.Id | Out-Null
6  ...
7  $AADUsers = Get-AzureADUser -All 1 | Select-Object ObjectType,UserPrincipalName,
           OnPremisesSecurityIdentifier,ObjectID,TenantId,email,AccountEnabled,ImmutableId,JobTitle,
           Mobile,ProxyAddresses,UserType # Get all Azure AD User object
8  ...
9  $AADUsers | ForEach-Object { # Loop through User objects
10 ...
11 $CurrentUser = [PSCustomObject]@{ # Add User attributes to an object
12 objectid = $User.ObjectID
13 userPrincipalName = $User.UserPrincipalName
14 email = $User.Mail
15 ...
16 $null = $Coll.Add($CurrentUser) # Add the object to an array
17 ...
18 New-Output -Coll $Coll -Type ''users'' -Directory $OutputDirectory # Write the JSON File with gathered
           information
19 ...
20 function New-Output($Coll, $Type, $Directory) { # Writes a JSON file
21 ...
22 $FileName = $Directory + [IO.Path]::DirectorySeparatorChar + ''az'' + $($Type) + ''.json''
```

Listing 3 AzHound.ps1 code snippet

3.4.3 Import script creation

We decided not to use or modify the existing BloodHound importer process. Instead, we decided to use the APOC library and *cypher-shell.jar* [44] as an import solution. The APOC library comes with the Neo4j database and consists of about 450 functions to help with many different tasks in areas like data integration, graph algorithms, or data conversion. For our import script, we used two functions from the library *apoc.load* to parse the JSON files and *apoc.merge* to create edges. *cypher-shell.jar* is the Neo4j command-line interface to execute cypher queries against a Neo4j database. Figure 13 depicts the import script process.

The code snippet Listing 4 showcases an example of how information of an Azure AD user is parsed to create a new AzUser node in the Neo4j database. The *azusers.json* file, which was created during the export process, is presented.

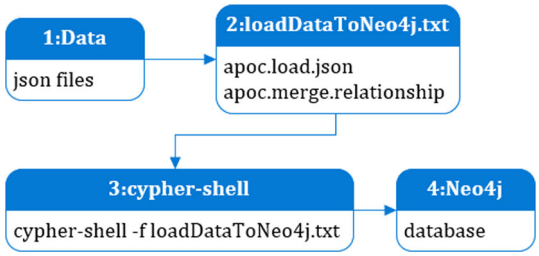

Fig. 13 Import script process overview

Lines 3–7 are metadata information and are used for the BloodHound import process. Lines 8–13 contain the user information required to create a new node. In this example, the user is *Leto*.

```
1  // azusers.json
2  "meta": {
3      "count": 17,
4      "type": "azusers",
5  },
6  "data": [
7  {
8      "displayname": "Leto",
9      "userPrincipalName": "leto@019840.onmicrosoft.com",
10     "objectid": "569d91c1-8f8c-4ac5-aaf8-9e753f7e45df",
11     ...
```

Listing 4 azusers.json file snippet

The second file *loadDataToNeo4j.cypher*, shown in Listing 5, calls on line 2, the APOC *load.json* function, to parse the data from the *azusers.json* file. Lines 5–9 create the AzUser node based on the parsed data from the JSON file.

```
1  // cypher-shell -f loadDataToNeo4j.cypher
2  CALL apoc.load.json("file://azusers.json") // Parse the JSON file with the Azure AD User information
3  YIELD value // Create a bound veriable from the APOC procedure call
4  UNWIND value.data AS fromJson // Transform data list back into individual rows
5  MERGE (n:AzUser{objectid:fromJson.objectid}) // Create AzUser node with the objectid from the JSON
6  ON CREATE SET n.displayname = fromJson.displayname, // Set properties
7           n.userPrincipalName = fromJson.userPrincipalName,
8           n.email = fromJson.email,
9           ...
```

Listing 5 loadDataToNeo4j.cypher code snippet for node creation

The code snippet in Listing 6 shows another example that creates edges between Azure AD users and Azure AD roles. The *azrolesAndAssignments.json*, which was created during the export process, is presented. Lines 3–7 are once again

the metadata information. Lines 8–14 contain the role id, the member of the role, and the assignment status.

```
1  // azrolesAndAssignments.json
2  "meta": {
3  "count": 18,
4  "type": "azrolesAndAssignments",
5  },
6  "data": [
7  {
8      "roleDisplayName": "Password Administrator",
9      "roleojectid": "966707d0-3269-4727-9be2-8c3a10f19b9d",
10     "member": "569d91c1-8f8c-4ac5-aaf8-9e753f7e45df",
11     "assignmentState": "EligibleTo",
12       ...
```

Listing 6 azrolesAndAssignments.json file snippet

Listing 7 calls, on line 2, the APOC *load.json* function to load the data from the *azrolesAndAssignments.json* file. On line 5, the database is queried for the *objectid*, which is the user *Leto* we saw before in the *azusers.json* file. On line 6, we search for the role with the *objectid*, which represents the Password Administrator. The roles were separately imported before, just like the Azure users. After the match, line 7 creates, with the *apoc.merge.relationships* function, an edge between the two nodes with the property *EligibleTo*.

```
1  // loadDataToNeo4j.txt
2  CALL apoc.load.json("file://azrolesAndAssignments.json") // Parse the JSON file
3  YIELD value // Create a bound veriable from the APOC call
4  UNWIND value.data AS fromJson // Transform data list back into individual rows
5  match (a {objectid:fromJson.member}) // Find member objectid pattern in the database
6  match (b {objectid:fromJson.roleojectid}) // Find roleobjectid pattern in the database
7  CALL apoc.merge.relationship(a,fromJson.assignmentState,{},{},b) yield rel as rely // Create an edge
           from member a to role b
8  return count(*) as relationships; // Return the amount of created edges
```

Listing 7 loadDataToNeo4j.cypher code snippet for edge creation

Further import statements can be found in the file *loadDataToNeo4j.cypher* under the BloodHoundAz GitHub repository [12].

3.5 Modification of the BloodHound web GUI

The BloodHound web application is based on Linkurious and compiled with Electron. It consists of multiple JavaScript files, which encompasses the functionality of the GUI, such as the navigation, loading of data, additional node, and edge information. We only did a minimal change to at least support the visualisation of the newly introduced node types. Therefore, we did not introduce any significant contribution to BloodHound's web GUI. With that decision, we do not have additional information regarding the nodes or edges available directly in the GUI. However, this will not impact the analysis as we still can query information directly from the graph database or use the Neo4j Browser if we require detailed node or edge information. The complete adjustment of the web application is something to be considered for future work. The change to the Bloodhound index javascript file is reflected in Fig. 14. The supported nodes in the current version are described in Table 5.

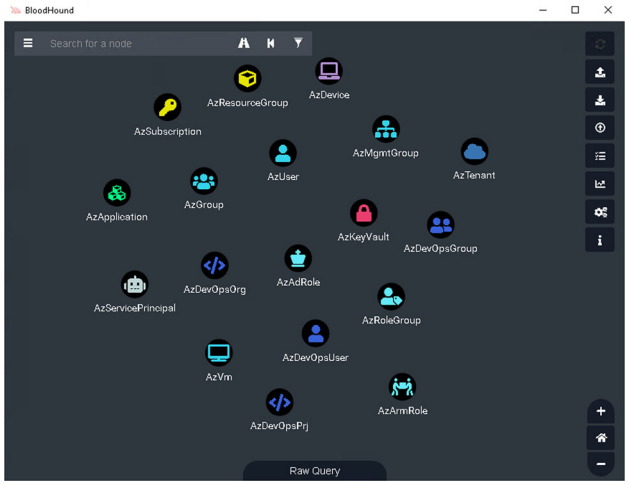

Fig. 14 Modified BloodHound GUI showing the 18 supported node types

Table 5 Description of the supported node types

Nodes	Description
AzTenant	Azure AD Tenant
AzAdRole	Azure Active Directory roles
AzApplication	Applications in an AAD Tenant
AzArmRole	Roles in Azure ARM
AzDevice	Devices in Azure AD
AzDevOpsGroup	Azure DevOps groups
AzDevOpsOrg	Azure DevOps organisations
AzDevOpsPrj	Azure DevOps projects
AzDevOpsUser	Azure DevOps users
AzGroup	Azure AD Groups
AzKeyVault	Key Vaults in Azure ARM
AzMgmtGroup	Management Groups in Azure ARM
AzResourceGroup	Resource Groups in Azure ARM
AzRoleGroup	Azure Role Groups in Azure AD
AzServicePrincipal	Azure Service Principal in AAD
AzSubscription	Subscriptions in Azure ARM
AzUser	Azure Users in Azure AD
AzVm	Virtual Machines in Azure ARM

3.6 Run the export and import script

Two manual steps are required to query the data from the Microsoft cloud and load the data to the Neo4j database. The first step is to execute in a PowerShell console the 'AzHound.ps1' script. The Azure AD user should have the following rights to guarantee the export of the required data:

- Assigned ARM Reader role.
- Approved *PrivilegedAccess.Read.AzureAD* delegation.

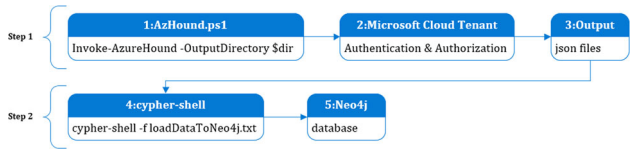

Fig. 15 Import and export process overview

– Approved *PrivilegedAccess.Read.AzureADGroup* delegation.
– Approved *PrivilegedAccess.Read.AzureResources* delegation.

The second step is executed in a command-line window. The cypher-shell will prompt for the username and password of the Neo4j database. Afterwards, the data is imported and ready for the analysis stage. Figure 15 depicts the described process.

4 Discussion

This section describes methods to analyse an attack graph and a resulting attack path. The outcome of the analysis is based on the defined use cases mentioned in the previous section and the entities created in the Azure AD test tenant.

4.1 Azure AD test tenant analysis

The attack graph analysis and the implementation steps depend on available test data. For that reason, we created in the Azure AD tenant multiple test entities. Following, we present the test tenant entities on which the attack graph analysis is based.

The cypher query was used to retrieve all available nodes from the Neo4j graph database. Figure 16 depicts the result of the query with a bar chart. Noticeable is the high number of *AzServicePrincipals*. Around 390 of the 405 service principals were created by Microsoft and should exist in every Azure AD tenant. They are required to guarantee the operability of M365 applications such as Graph Explorer, Exchange Online, SharePoint Online and Teams. But also, these default service principals can be abused. Therefore, these service principals were also added to the Graph [42].

To verify the use cases outlined in the previous section, we granted various permissions to the created entities in the Microsoft cloud. In total, 2108 edges were created from ingested data. A cypher query was used to retrieve all edges from the Neo4j graph database. Figure 17 depicts the result in the form of a bar chart. The two highest edge numbers are *CanManage* and *ResetPassword*. The high number is due to the fact that certain Azure AD roles have the right to manage or reset passwords of all entities in an Azure AD tenant.

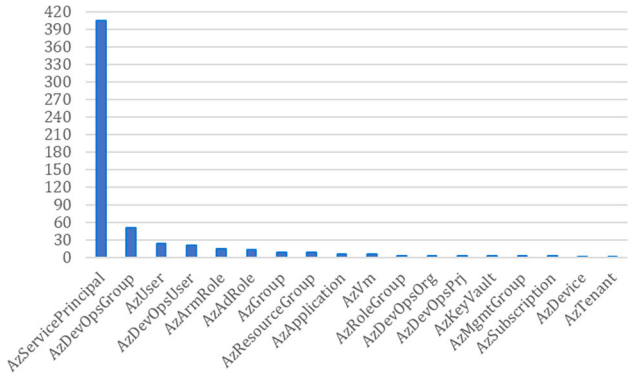

Fig. 16 Nodes overview

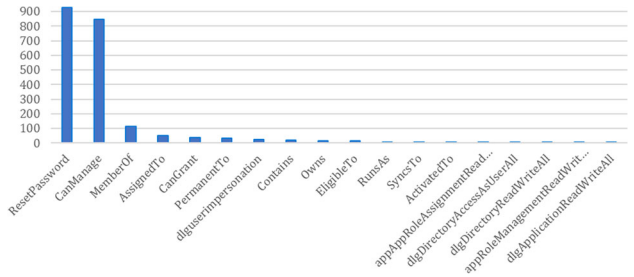

Fig. 17 Edges overview

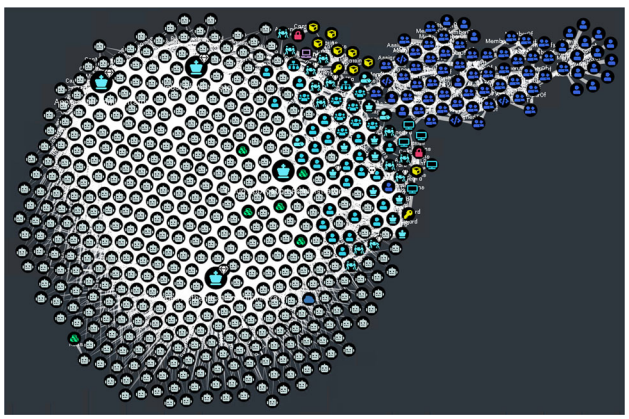

Fig. 18 Confusing and abstract attack graph

4.2 Attack graph analysis

When chosen poorly, a graph can become too abstract and too confusing to find the correct information. Figure 18 shows such a graph which was generated with a cypher query that used the all shortest path graph algorithm to calculate all possible paths to any node in the graph.

A better method to analyse an attack graph is to find influential nodes by using centrality algorithms. Neo4j provides multiple available centrality algorithms. For the attack graph presented in this work, two centrality algorithms were used to identify powerful nodes or, in the context of a Microsoft cloud, privileged entities.

Fig. 19 Closeness centrality result

4.2.1 Closeness centrality algorithm

The closeness centrality algorithm measures the nodes' average distance to all other nodes and generates a list of nodes that are able to propagate information efficiently through a graph [43]. Nodes with a high closeness score have the shortest distances to all other nodes, which makes them privileged entities in the context of a Microsoft cloud. A cypher query was used to run the algorithm against the graph.

The result of the query is depicted in Fig. 19. For visualisation reasons, not every node was included in the chart. The closeness score shows that roles such as *Global Administrator* or *Application administrators* have the shortest distance to all, followed by certain users such as *marius* and particular service principals such as *019840-Thinking-in-Graphs* and *myApplication*.

4.2.2 Degree centrality algorithm

The degree centrality algorithm can help to determine popular nodes in a graph. The algorithm measures the number of incoming and outgoing edges from a node. If nodes have a high number of edges, in particular outgoing edges, the entity should be flagged as highly privileged in the context of the Microsoft cloud. A cypher query was used to run the algorithm against the graph.

The result of the query is depicted in Fig. 20. For visualisation reasons, not every node was included in the treemap chart. Similarly to the results presented by the closeness centrality chart, degree centrality also identified the roles as highly privileged in the context of the Microsoft cloud. The roles are followed by service principals such as *myApplication* and Azure AD users such as *marius*.

In summary, both algorithms present similar results, which underline that the aforementioned discovered entities, in particular the role *Global Administrator*, are the privileged entities in the Microsoft cloud. The result proves on one side that the created graph provides accurate data because the *Global Administrator* is, in fact, the most powerful entity in the Microsoft cloud [34]. On the other side, they show that powerful entities that are not existing by default in an Azure AD tenant can be prioritised for further analysis.

4.2.3 Shortest path algorithm

The centrality algorithm results can help to identify powerful nodes or, in the context of the Microsoft cloud, privileged entities. Path algorithms can help to find all nodes that have direct or indirect paths to such powerful nodes. A cypher query was used to find the shortest path from any node to the node with the name *Global Administrator* overall available edges. The result of the cypher query is presented in the form of an attack graph in Fig. 21.

The visualisation shows that different types of entities have direct or indirect paths to the role *Global Administrator*. The first edges to the *Global Administrator* role are *EligibleTo*, *PermanentTo*, *CanManage*, and *CanGrant*. These are the closest chokepoints (Warning symbol) to becoming a *Global Administrator*. Going further to the left, other edges appear, which form the indirect paths to the node *Global Administrator*. For defenders, the chokepoints are essential. This is because edges that are close to the *Global Administrator* node are controlling a vast number of entities. But which edges should be removed first to reduce the total amount of Azure AD users that can become *Global Administrators*? One method to answer this question is to measure with the cypher query in Listing 8 the current percentage of Azure AD users who have a direct or indirect path to *Global Administrator*.

```
1 MATCH (u:AzUser)
2 MATCH (r:AzAdRole{name:"Global Administrator"})
3 WITH g, COUNT(u) as userCount
4 MATCH p = shortestPath((u:AzUser)-[*1..]->(r))
5 RETURN 100.0 * COUNT(DISTINCT u) / userCount as percent
6 // Result: 62%
```

Listing 8 Cypher query to measure how many users have a path to the Global Administrator role

For our attack graph in Fig. 21, 62% of the total 24 Azure AD Users have a path to the Global Administrator role. To minimise this controlling number, graph theory can be used to calculate the effectiveness of edge removals before touching the actual IT environment. For example, by removing the *CanGrant* edge from the Service Principal (1) shown in Fig. 21 and by re-running the cypher query in Listing 8, the new net result would be 54%. This means that by removing the *CanGrant* edge, an additional 8% of Azure AD Users would not be able to become Global Administrators in the Azure AD tenant.

4.3 Attack path example

In the following, a hypothetical attack path presenting how a user can become a Global Administrator by exfiltrating credentials from Azure DevOps and abusing Azure AD application role permissions is described. The presented attack

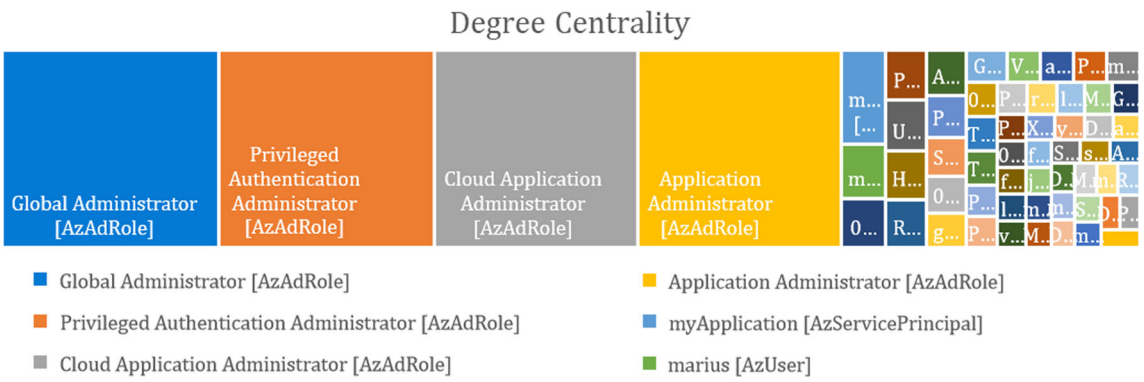

Fig. 20 Degree centrality result

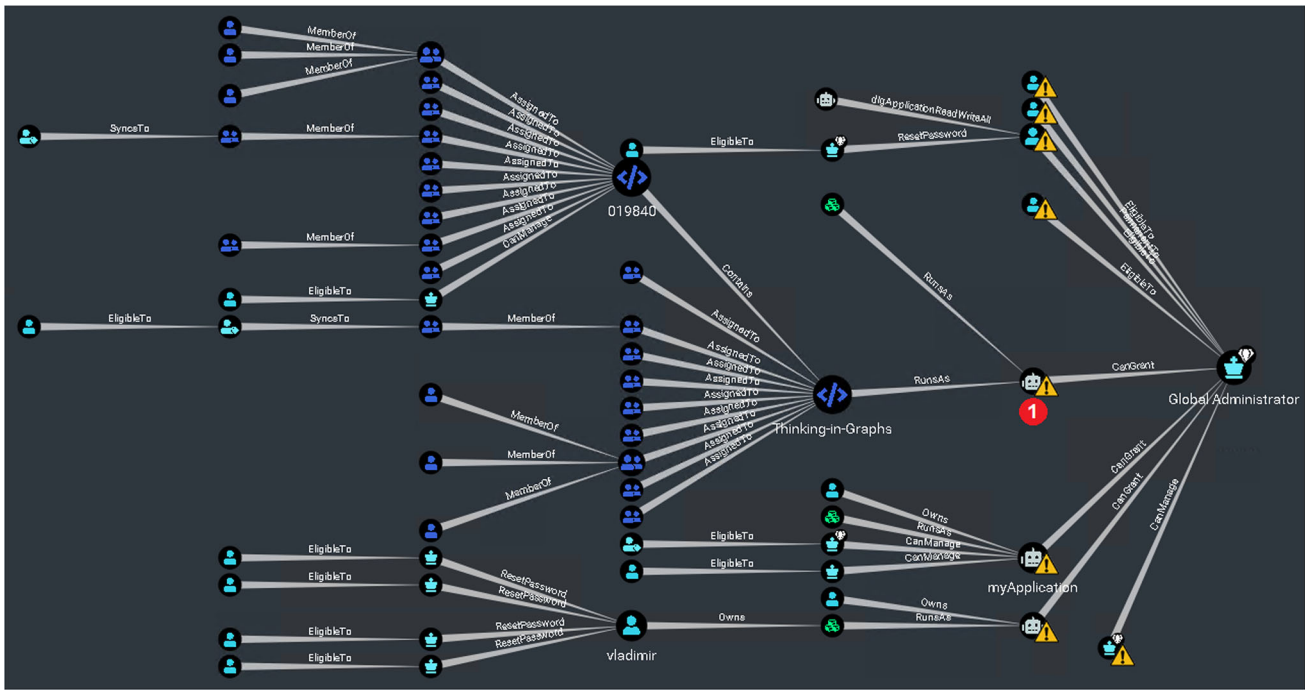

Fig. 21 Shortest path result from any node to the Azure AD tenant

path was inspired by [52], who covers the app role abuse scenario more in detail. The chosen attack path was extracted from the shortest path attack graph. A cypher query was used to visualise the attack path in Fig. 22. It is important to note that this example was extracted manually from the attack graph shown in Fig. 21 to showcase the impact of an attack path scenario.

The attack path depicts, in the upper-left corner, a user called *Rabban*. The assumption is that the user was compromised by an adversary. The adversary has the goal to become Global Administrator to gain full control over the Azure AD tenant. The following six steps describe the attack path from the viewpoint of the adversary.

1. The adversary activates the privileged access group called *DevOpsRole-Build Admin* in the Privileged Identity Management GUI for the user *Rabban*.
2. The Group *DevOpsRole-Build Administrators* with its members are synchronised to Azure DevOps. Azure DevOps is a SaaS platform from Microsoft that provides an end-to-end DevOps toolchain for developing and provisioning. The *DevOpsRole-Build Administrators* group is a direct member of three built-in DevOps groups. According to the description of the Microsoft documentation, the memberships should allow the user *Rabban* to create and modify pipelines. In Azure DevOps, pipelines are generally used for deployments. The deployments use service principals or other types of key material to authen-

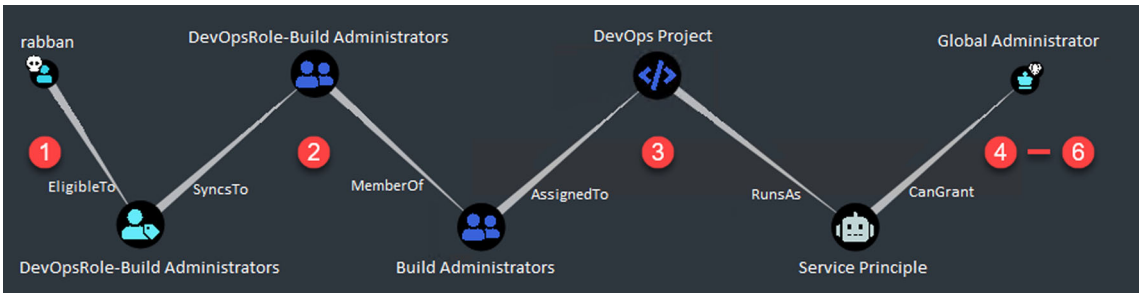

Fig. 22 Attack path example

ticate to a target system. This makes the Azure DevOps platform particularly interesting for adversaries.

3. By opening the pipeline configuration, the adversary notices that a service principal is used to deploy resources to the targeted Azure AD tenant. This is also shown by the edge *RunsAs* in the attack path in Fig. 22. The adversary decides to dump the password of the service principal by modifying the pipeline to print the password to the terminal. By default, Azure DevOps prevents the output of credentials in plain text. Converting the credentials to hex circumvents these preventive measures, and the credentials can be retrieved as shown in Fig. 23 and Fig. 24.
4. The next three steps use a PowerShell script [11] that was created to automate the second part of the attack. The gained service principal key is converted to ASCII, and the adversary connects with the service principal to Azure AD.
5. The service principal has now been assigned the app role *AppRoleAssignment.ReadWrite.All*. This role allows the service principal to request a new app role called *RoleManagement.ReadWrite.Directory*. Per documentation of Microsoft, the newly granted app role allows the service principal to manage Azure AD role memberships.
6. In the last step, depicted in Fig. 25, a new token, which includes the new app role, is requested for the service principal. The adversary decides to add the user *Rabban* to the *Global Administrator* role and achieves the goal to become a Global Administrator.

To prevent the described multistage attack path, different security controls such as MFA, role approval requests, or auditing could have been implemented to make it more difficult for an attacker. But in our opinion, prevention starts already before the implementation of such security layers. It begins with understanding how an IT system such as the Microsoft cloud works. One method is screening entities and their permissions and by asking the question, how can this permission or entity impact my Azure AD tenant? Finding the answer can be trivial or difficult, but it will help categorise and map entities according to their priorities. With that approach, the journey to "knowing your assets" has started. Graph theory is an effective method that can help on this journey which makes complex relationships visible and remediation measurable. Blind spots such as the presented attack path can be analysed, and decisions can be made to remove unwanted

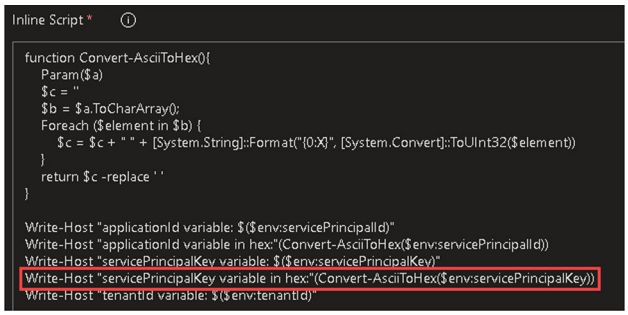

Fig. 23 Azure DevOps attack path

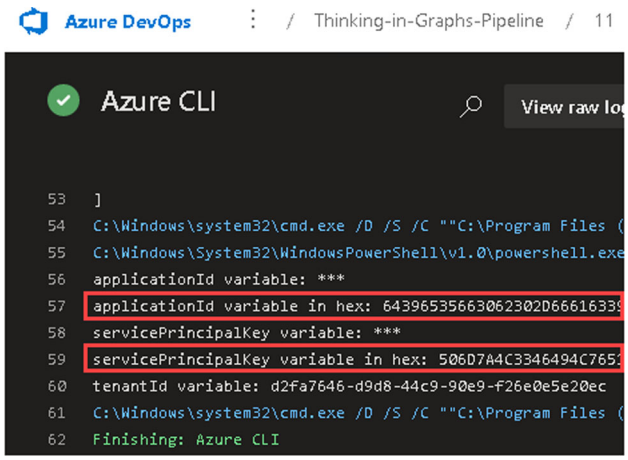

Fig. 24 Dump service principal key from Azure DevOps pipeline

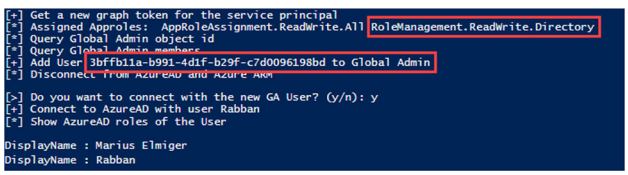

Fig. 25 Privilege escalation to Global Administrator

paths to *Global Administrator* or implement tangible preventive measures.

5 Conclusion

Our work highlighted that cloud technology is primarily identity-centric, unlike on-premises environments, where everything is placed in an internal network, and the security configurations are set around the perimeter. Based on this insight, cloud credential pivoting, a post-compromise technique by which the adversary tries to gather new credentials from cloud tenant resources, was presented. Similar to on-premises, resources in the cloud can have different access permissions that form direct or indirect relationships with one another. Thus, the most familiar Microsoft cloud tenant entities were presented, followed by a pivoting graph example, illustrating the relationship between the entities which could be abused by an adversary. We concluded that when it comes to identifying critical attack paths in a Microsoft cloud tenant, graphs provide unique and valuable insights into highly connected data.

To validate this statement through a practical implementation, a methodology with three main stages was presented. During the first two stages, a graph analysis platform using tools such as the Neo4J graph database and BloodHound was planned and built. This also included the design of a graph database schema and the definition of new node and edge types. To populate the graph database with data, an export and import script was created to ingest test data from a Microsoft cloud test tenant. In the last stage, the graph analytics methods were presented to identify privileged entities, and a proposal was described to measurably reduce attack paths to such entities.

The goal of this paper is to evaluate the benefits of graph-based data representation for understanding and uncovering complex entity attack paths in the Microsoft cloud. Through the work, we presented the advantages of using graphs. The presented technical approach can also be applied to other IT environments such as Google or Amazon Cloud.

Our work recorded the following key findings:

- Attack path identification and analysis are limited by two aspects, the collected data and the implemented edges between nodes.
- Various methods exist to query data from the Microsoft cloud, such as with the official Azure PowerShell cmdlets or directly from the Microsoft Graph REST API.
- The data that can be queried from the Microsoft cloud is not always following the same data schema. Hence, often research is required to interpret the queried data correctly.
- The Microsoft cloud, compared with an on-premises Active Directory, has multiple services which can make access decisions, such as Azure AD API, ARM API, MS Graph API, various cloud applications and more. Visualising these access decisions in the form of edges in a Graph can uncover the complexity of a Microsoft cloud environment.

5.1 Future work

The edges and nodes that were implemented during this research were just a start and can be extended with many more use cases. New nodes could be added in regard to Azure ARM, such as storage accounts, Azure functions, Azure logic apps, network components and many more. New edges could be added as well between the already implemented nodes, such as key vault access permissions, app role permissions, Azure AD roles and so on. Summarised, the Microsoft cloud is continuously changing as new features are added; thus, the iteration of finding new attack paths is an ongoing task. In addition, the generation of attack graphs and the identification of relevant attack paths are a manual process that would benefit from automation. In future research, we can look into automating this procedure and ranking the identified attack paths based on relevancy using graph algorithms.

Moreover, the BloodHound extensions that were added during this research would require further adaptations. These include the modification of the import functionality in the BloodHound web GUI and the adaptation in general of the web GUI to support the newly added node types. After these changes are completed, we foresee a release of the modified BloodHound version to the public. This would benefit anyone that is interested in using graphs to assess their Azure AD tenant. Furthermore, we are committed to run a comparative analysis study using our approach against existing methods, which should offer a better understanding of the limitations and advantages of our approach. Careful consideration will be given to important factors such as the scope of the analysis, the criteria for evaluation, and the source of the data.

Data availability The data generated and/or analysed in the current study can be obtained at https://github.com/m8r1us/BloodHoundAz.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Al-Mohannadi, H., Mirza, Q., Namanya, A., Awan, I., Cullen, A., Disso, J.: Cyber-attack modeling analysis techniques: An overview. In: 2016 IEEE 4th international conference on future internet of things and cloud workshops (FiCloudW), pp. 69–76. IEEE, 2016
2. Ali, M., Khan, S.U., Vasilakos, A.V.: Security in cloud computing: opportunities and challenges. Inf Sci **305**, 357–383 (2015)
3. Azure. Github - azure/stormspotter: Azure red team tool for graphing azure and azure active directory objects. https://github.com/Azure/Stormspotter, 2020. Accessed: 14-10-2022
4. Bechberger, D., Perryman, J.: Graph Databases in Action. Manning Publications, New York (2020)
5. Besta, M., Peter, E., Gerstenberger, R., Fischer, M., Podstawski, M., Barthels, C., Alonso, G., Hoefler, T.: Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. arXiv preprint arXiv:1910.09017, 2019
6. Bouillot, L., Gras, E.: Chemins de Contrôle en Environnement Active Directory, 2014
7. Brath, R., Jonker, D.: Graph Analysis and Visualization: Discovering Business Opportunity in Linked Data. John Wiley & Sons, New York (2015)
8. Burrough, M.: Pentesting Azure Applications: The Definitive Guide to Testing and Securing Deployments. No Starch Press, San Francisco (2018)
9. Dunagan, J., Zheng, A. X., Simon, D. R.: Heat-ray: combating identity snowball attacks using machinelearning, combinatorial optimization and attack graphs. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles pp. 305–320, 2009
10. Duncan, R.: A multi-cloud world requires a multi-cloud security approach. Comput. Fraud Secur. **2020**(5), 11–12 (2020)
11. Elmiger, M.: Attack path automation powershell. https://gist.github.com/m8r1us/5babd1d63c25c0199520a0a2f8e4f2e4, 2022. Accessed: 14-10-2022
12. Elmiger, M.: Bloodhoundaz. https://github.com/m8r1us/BloodHoundAz, 2022. Accessed: 14-10-2022
13. Fernandes, D.A., Soares, L.F., Gomes, J.V., Freire, M.M., Inácio, P.R.: Security issues in cloud environments: a survey. Int. J. Inf. Secur. **13**(2), 113–170 (2014)
14. Fosaaen, K.: How to escalate azure privileges with the log analytics contributor role, 2021. Accessed: 14-10-2022
15. Fritzson, P.: Electron-build cross platform desktop apps with javascript, html, and css. *URL:* http://electron.atom.io/, 2016
16. Gosnell, D., Broecheler, M.: The Practitioner's Guide to Graph Data: Applying Graph Thinking and Graph Technologies to Solve Complex Problems. O'Reilly Media Inc, Sebastopol (2020)
17. Grobauer, B., Walloschek, T., Stocker, E.: Understanding cloud computing vulnerabilities. IEEE Secur. Privacy **9**(2), 50–57 (2010)
18. Ho, G., Dhiman, M., Akhawe, D., Paxson, V., Savage, S., Voelker, G. M., Wagner, D.: Hopper: Modeling and detecting lateral movement. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3093–3110, 2021
19. Holm, H., Sommestad, T., Almroth, J., Persson, M.: A quantitative evaluation of vulnerability scanning. Inf. Manage. Comput. Secur. **19**, 231–247 (2011)
20. Hyllienmark, E.: Evaluation of Two Vulnerability Scanners Accuracy and Consistency in a Cyber Range, 2019
21. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), pp. 121–130. IEEE, 2006
22. Johnson, A., Faust, S.: Cloud post exploitation techniques infiltrate. https://sec.today/events/talk/c559e549-35ad-4122-b64f-2ef7f78e1d2e/, 2017. Accessed: 14-10-2022
23. Kheir, N., Mahjoub, A.R., Naghmouchi, M.Y., Perrot, N., Wary, J.-P.: Assessing the risk of complex ICT systems. Annals Telecommun. **73**(1), 95–109 (2018)
24. Kunz, B.: Blue cloud of death: Red teaming azure. https://speakerdeck.com/tweekfawkes/blue-cloud-of-death-red-teaming-azure-1, 2018. Accessed: 14-10-2022
25. Kyriakidis, A., Maniatis, K., You, E.: The majesty of Vue. js. Packt Publishing, Birmingham (2016)
26. Lambert, J.: Defenders Think in Lists. Attackers Think in Graphs. As Long as this is True, Attackers Win, 2015
27. Lassila, O., Swick, R. R.: et al. Resource Description Framework (RDF) Model and Syntax Specification. Citeseer, 1998
28. Lenk, A., Klems, M., Nimis, J., Tai, S., Sandholm, T.: What's inside the cloud? An architectural map of the cloud landscape. In: 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, pp. 23–31. IEEE, 2009
29. Linkurious. Linkurious| investigation solutions | bringing criminal activity to light. Accessed: 14-10-2022
30. Mell, P., Grance, T., et al.: The NIST Definition of Cloud Computing. National Computer Security Division, Information Technology Laboratory (2011)
31. Mescheryakov, S., Shchemelinin, D., Izrailov, K., Pokussov, V.: Digital cloud environment: present challenges and future forecast. Future Internet **12**(5), 82 (2020)
32. Microsoft. Azure active directory feature deployment guide. https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-deployment-checklist-p2. Accessed: 14-10-2022
33. Microsoft. Azure active directory powershell for graph module. https://learn.microsoft.com/en-us/powershell/module/azuread/?view=azureadps-2.0. Accessed: 14-10-2022
34. Microsoft. Azure ad built-in roles - global administrator. https://learn.microsoft.com/en-us/azure/active-directory/roles/permissions-reference#global-administrator. Accessed: 14-10-2022
35. Microsoft. Azure devops services rest api reference. https://learn.microsoft.com/en-us/rest/api/azure/devops/?view=azure-devops-rest-7.1. Accessed: 14-10-2022
36. Microsoft. Azure Monitor workbooks for reports - Microsoft Entra. https://learn.microsoft.com/en-us/azure/active-directory/reports-monitoring/howto-use-azure-monitor-workbooks
37. Microsoft. Microsoft 365 developer subscriptions in visual studio subscriptions. https://docs.microsoft.com/en-us/visualstudio/subscriptions/vs-m365. Accessed: 14-10-2022
38. Microsoft. Microsoft graph rest api v1.0 endpoint reference. https://learn.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0. Accessed: 14-10-2022
39. Microsoft. Shared responsibility in the cloud - Microsoft Azure. https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility
40. Microsoft. View audit report for Azure resource roles in Privileged Identity Management (PIM) - Azure AD - Microsoft Entra. https://

learn.microsoft.com/en-us/azure/active-directory/privileged-identity-management/azure-pim-resource-rbac
41. Microsoft. What is identity secure score? - Azure Active Directory - Microsoft Entra. https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/identity-secure-score
42. Mollema,D.: Abusing azure ad SSO with the primary refresh token, 2020. Accessed: 14-10-2022
43. Needham, M., Hodler, A.E.: Graph Algorithms: Practical Examples in Apache Spark and Neo4j. O'Reilly Media, Sebastopol (2019)
44. Neo4j. Cypher shell - operations manual. https://neo4j.com/docs/operations-manual/4.4/tools/cypher-shell/
45. Neo4j. Installation - Neo4j Desktop. https://neo4j.com/docs/desktop-manual/1.4/installation/
46. Noel, S., Wang, L., Singhal, A., Jajodia, S.: Measuring security risk of networks using attack graphs. Int J Next-Gener Comput **1**, 135–147 (2010)
47. Ou, X., Govindavajhala, S., Appel, A.W., et al.: Mulval: a logic-based network security analyzer. In: USENIX Security Symposium. vol 8, pp. 113–128. Baltimore (2005)
48. Paxton, N. C.: Cloud security: a review of current issues and proposed solutions. In: 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), pp 452–455. IEEE, 2016
49. Pitropakis, N., Darra, E., Vrakas, N., Lambrinoudakis, C.: It's all in the cloud: Reviewing cloud security. In: 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing, pp 355–362. IEEE, 2013
50. Quasar. Quasar framework - build high-performance vuejs user interfaces in record time. https://quasar.dev/. Accessed: 14-10-2022
51. Robbins, A.: Introducing the adversary resilience methodology - part one, 2018. Accessed: 14-10-2022
52. Robbins, A.: Azure privilege escalation via azure API permissions abuse, 2021. Accessed: 14-10-2022
53. Robbins, A., Schroeder, W., Vazarkar, R.: Github - bloodhoundad/bloodhound: Six degrees of domain admin. https://github.com/BloodHoundAD/BloodHound, 2016. Accessed: 14-10-2022
54. Robinson, I., Webber, J., Eifrem, E.: Graph Databases: New Opportunities for Connected Data. O'Reilly Media Inc, Sebastopol (2015)
55. Sheyner, O., Wing, J.: Tools for generating and analyzing attack graphs. In: International Symposium on Formal Methods for Components and Objects, pp. 344–371. Springer, Berlin (2003)
56. Soh, J., Copeland, M., Puca, A., Harris, M., Soh, J., Copeland, M., Puca, A., Harris, M.: Microsoft azure and cloud computing. In: Microsoft Azure: Planning, Deploying, and Managing the Cloud, pp. 3–20. Apress, Berkeley (2020)
57. Sommestad, T., Sandström, F.: An empirical test of the accuracy of an attack graph analysis tool. Inf. Comput. Secur. **23**(5), 516–531 (2015)
58. Swiler, L. P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01, vol 2, pp. 307–321. IEEE, 2001
59. Yi, S., Peng, Y., Xiong, Q., Wang, T., Dai, Z., Gao, H., Xu, J., Wang, J., Xu, L.: Overview on attack graph generation and visualization technology. In: 2013 International Conference on Anti-Counterfeiting, Security and Identification (ASID), pp. 1–6. IEEE, 2013
60. Zeng, J., Wu, S., Chen, Y., Zeng, R., Wu, C.: Survey of attack graph analysis methods from the perspective of data and knowledge processing. Secur. Commun. Netw. **2019**, 1–16 (2019)