ADSynth: Synthesizing Realistic Active Directory Attack Graphs

Nhu Long Nguyen
Computer and Mathematical Sciences
University of Adelaide
Adelaide, Australia
nhulong.nguyen@adelaide.edu.au

Nickolas Falkner
Computer and Mathematical Sciences
University of Adelaide
Adelaide, Australia
nickolas.falkner@adelaide.edu.au

Hung Nguyen
Computer and Mathematical Sciences
University of Adelaide
Adelaide, Australia
hung.nguyen@adelaide.edu.au

***Abstract*—Active Directory (AD), a directory service for Windows domain networks, is a common target for attackers due to its widespread use and the confidential data it contains. According to Microsoft, 95 million AD accounts are attacked every day and new attacks involving AD are a common occurrence. Despite frequent attacks against Active Directory and its critical role in network security, there are no publicly available datasets and tools for generating realistic AD graphs. This absence hinders the development and testing of novel methods for protecting AD systems. Realistic AD datasets are also essential for training and up-skilling human AD defenders.**

In this work, we develop ADSynth, a scalable and realistic AD attack graph generator. ADSynth uses metagraphs to model design principles of realistic AD systems, relying on three novel ideas: (1) metagraph abstractions of best practices in AD organizational design, (2) metagraph abstractions of security design principles in AD systems, and (3) a random metagraph model of common security misconfigurations. Our experiments demonstrate ADSynth's scalability in creating realistic AD graphs under various security settings. We apply ADSynth to some recent research on AD security and demonstrate that data from ADSynth significantly benefit these studies. ADSynth has been released to the community[1] [2].

***Index Terms*—Active Directory, ADSynth, metagraph, realistic**

I. Introduction

Active Directory (AD) [1] is a proprietary centralized directory service that manages network information and settings in numerous organizations worldwide. It has become a common target of many recent attacks [2]. Microsoft recorded more than 25 million brute force attacks against Azure Active Directory systems in 2021 alone [3]. This has driven research and commercial activities to identify potential risks and attack paths in Active Directory (AD) to enhance the security of all AD systems [4], [5], [6].

Challenges. Due to the confidential and sensitive information stored in Active Directory, there is a lack of realistic datasets available to researchers. Although tools exist to generate AD attack graphs, such as DBCreator [7], BadBlood [8], ADSimulator [9], they are limited in how well they model large AD graphs of millions of nodes observed in large organizations. In addition, graphs generated by these tools generally do not match the structures of graphs that are observed in real AD systems.

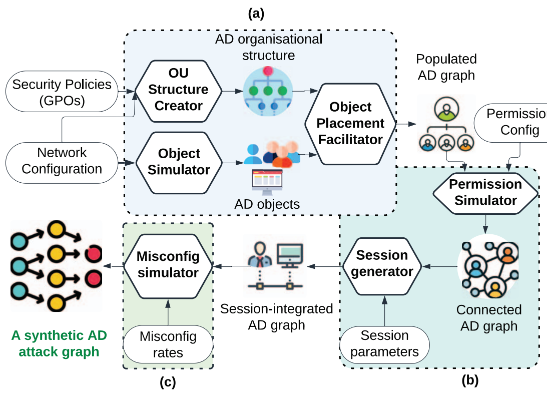

Fig. 1. ADSynth architecture: (a) Node generation stage; (b) Edge generation stage; (c) Violation/Misconfiguration generation stage

Solution. We develop ADSynth, a scalable and synthetic Active Directory attack graph generator. ADSynth starts with a simple observation that real AD systems follow design principles and best practices such as those issued by Microsoft and professional organizations [10], [11], [12]. These guidelines often operate on sets of users and/or computers, with policies defined on set-to-set mappings. To model these, ADSynth uses a set-to-set mapping abstraction called *metagraph* [13] and generates the organizational structure and nodes of the attack graphs. In the next phase, the tool creates edges between nodes to represent relationships between sets and access permissions specified in best practices and security policies of real AD systems [10], [11]. In addition, ADSynth simulates a wide range of misconfigurations to replicate the effects of human errors in system administration. The software architecture of ADSynth is illustrated in Fig. 1. The performance of ADSynth is evaluated against DBCreator and ADSimulator on graphs of similar sizes and on a real AD graph of a University.

Benefits. ADSynth data can benefit AD security in multiple ways. First, ADSynth data enables the development of novel algorithms for managing AD attack graphs. In the application, by utilizing ADSynth realistic data, we can prove the working of a novel algorithm [14] when applied in practice and display

[1]https://adsynthesizer.github.io/
[2]https://github.com/adsynthesizer/ADSynth.git

2158-3927/24/$31.00 ©2024 IEEE
DOI 10.1109/DSN58291.2024.00021

limitations of another algorithm [4] in realistic AD systems, thereby prompting further refinement to enhance its efficacy. Second, ADSynth data allows testing and validation of various commercial solutions for AD security. In particular, we apply ADSynth data on an attack path management tool [15] to validate the claim made by the author regarding the improvement of AD defenders' strategy in eliminating attack paths. In addition, ADSynth graphs could also be used for graph theory research in metagraphs, learning and training purposes, allowing students and professionals to learn and practice AD security concepts.

Our primary contributions in this paper are as follows:

- A scalable tool to generate synthetic Active Directory (AD) attack graphs that resemble realistic AD systems and follow the design guidelines and best practices.
- We benchmark ADSynth against existing tools such as DBCreator and ADSimulator. We also compare data generated from ADSynth with a University AD system to illustrate the realism of data produced by ADSynth.
- We apply ADSynth-generated data to existing reference works on AD security and show the values of these realistic datasets in improving these works.

II. Background and Related Work

A. Active Directory components

An Active Directory (AD) is essentially a centralized database that stores network information, where the data is stored as objects. There are 6 basic types of AD objects: (1) **Domain**: A Domain is a structural component, containing all AD objects; (2) **Users**: A user represents an individual user's AD account, which is identified by a username and authenticated through a password to ensure protection against unauthorized access to network resources; (3) **Computers**: A computer represents a physical machine in the AD network; (4) **Groups**: A group functions as a container-like entity that stores *references* to users, computers, and other groups, to coordinate and manage the permissions granted to a set of objects in an AD system. Members of a group inherit all permissions granted to that group; (5) **Organisational Units (OUs)**: An OU is a container within AD systems to store users, computers, groups, and other OUs. OUs are utilized to reflect the structure of an organization, which can be based on locations, departments, object types, and other design criteria; (6) **Group Policy Objects (GPOs)**: virtual collections of policy settings [16] to regulate the behaviours of AD objects.

AD permissions are a set of rules to grant an object certain control over other objects. For example, Domain Admins group has full control over all AD objects, a local administrator can view local users' credentials or reset their passwords, among the many authorized actions. These are best modeled using a set-to-set mapping abstraction described below.

B. AD Attack graph models and generators

An attack graph model depicts the potential paths an attacker could take to exploit security vulnerabilities in a system. There are currently more than 90 attack graph models [17].

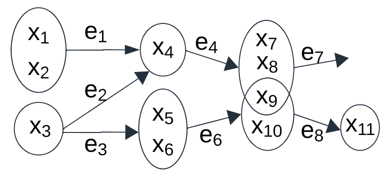

Fig. 2. A metagraph. The generating set of this metagraph is $\{x_1, x_2, x_3, \ldots, x_{11}\}$. The edge e_1 can be described by the pair $\langle\{x_1, x_2\}, \{x_4\}\rangle$, which are the invertex and outvertex of the edge.

Among these, the Active Directory attack graph is one of the most widely recognized and frequently employed. An Active Directory (AD) system is inherently complex due to the increasingly numerous interconnected components it contains. The aggregate intrinsic complexity causes misconfigurations in AD management and administration, which leads to a serious type of cyber attacks called *identity-based snowball attacks* [18]. In a snowball attack, after compromising a machine, the attackers can steal the credentials of users currently logged in to perform lateral movements and compromise other machines. If these users have administrative rights, the attackers can launch a privilege escalation attack to compromise critical assets of the system. AD attack graph models for identity snowball attacks and defense are currently commercialized by the popular Bloodhound [19] software, recommended by the US Department of Homeland Security.

Most research and industry work related to the security of Active Directory (AD) uses AD graphs generated by tools such as DBCreator [7] and ADSimulator [9]. These tools, however, do not follow how ADs are built in real organizations, including how to construct AD structures and delegate security permissions between objects. In particular, the access control assignment within these graphs is random, which deviates from the standard [20]. In addition, the current tools do not feature a tier model which separates objects based on their privileges. The tier model is a practice strongly endorsed by Microsoft to enhance the overall security of an Active Directory system [20]. Scale is also challenging. ADSimulator, one of the well-known tools used in many recent works [5], [14], [21], struggles to generate networks of 100000 nodes. While this seems large, a medium-sized commercial AD graph could have millions of nodes.

C. Metagraph Model of AD

A graph, G, is traditionally defined as $\langle V, E\rangle$, where V is the set of vertices, and E is the set of edges that connect the vertices $v_i \in V$. Metagraphs extend this abstraction to model mapping between sets to sets. Formally, a metagraph [13] is defined as: A metagraph $S = \langle X, E\rangle$ is a graphical construct specified by a generating set X and a set of edges E. A generating set is a set of elements $X = \{x_1, x_2, \ldots, x_n\}$, which represent variables of interest. In the AD context, these could be users and computers in an AD environment. An edge e is a pair $e = \langle V_e, W_e\rangle \in E$ consisting of two sets, an invertex $V_e \subset X$ and an outvertex $W_e \subset X$. Fig. 2 provides an example of a metagraph. Each of the sets in the metagraph such as $\{x_1, x_2\}$ could be used to represent a group in AD and the edge from $\{x_1, x_2\} \to \{x_4\}$ could be used to represent a permission setting from group $\{x_1, x_2\}$ to computer $\{x_4\}$.

III. ADSynth - Model and Implementation

A. From Metagraph to AD Attack Graphs

Metagraph nodes - Organisation units (OUs) and Groups. In Active Directory, OUs contain AD objects and Groups store *references* to them. Each OU and group functions as a set, encompassing multiple objects. The uniqueness of objects within a set is determined by a globally unique identifier (GUID) [22]. OUs define the hierarchical structure of an AD system. Beginning with a domain representing the entire organization, the AD structure extends into nested OUs. An example organizational structure adhering to best practices with 3 tiers is illustrated in Fig. 3, with two distinct departments (IT and HR), each with offices in City A and B.

Groups in Active Directory are divided into 2 types, *distribution groups* and *security groups* [23]. Distribution groups are used to distribute emails to group members [23], [12], [24]. Security groups are employed to grant access to root folders [23], [12], [24]. They are also used for delegating permissions to multiple principals [24]. Objects such as users, computers, etc. are included in groups as member entities and inherit all permissions the groups have. Groups are situated within specific OUs.

Metagraph Edges - Security permissions As AD typically contains millions of objects, instead of specifying the control of a principal over every single object, they apply permissions on the OU containing a set of objects, and these permissions are passed down to child objects [25] using inheritance.

To grant OU permissions to multiple users, AD admins configure permissions on groups, enabling group members to gain permissions over OUs [25] and objects contained within OUs, specified by the inheritance type. The resulting structure therefore represents the set-to-set mappings of a metagraph, as demonstrated in Fig. 2.

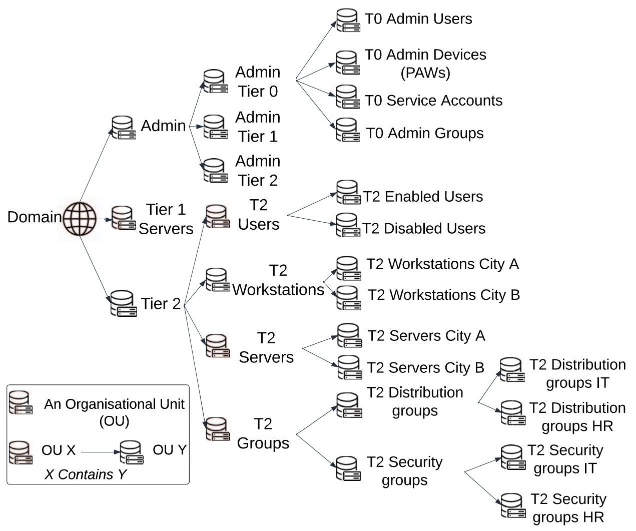

Fig. 3. ADSynth network integrated with the Tier Model

Metagraph Modelling Formally, we can model an AD and permissions setting within an AD using a metagraph as follows. We define X as a set of nodes representing all entities in an AD system and E as edges representing permissions between them. $O = \{o_1, o_2, \dots, o_k\}$ denotes a set of organisational units with $o_i \subset X$. In the above scenario, $V \subset X$ is a group containing users, $W \subset X$ is a set of objects, and $Q \subset O$ is an OU containing those objects such that $Q \equiv W$. An edge $e = \langle V, Q \rangle \in E$ with an invertex V and an outvertex Q represents a permission between the set V and the organizational unit Q. The permission type is included as an edge label in the attribute list P_e of the edge.

AD permissions - Access Control Lists (ACLs). An example of permission setting in AD is the application of Access Control Lists. Each AD object has a security descriptor containing a list of entries, detailing what entity has permissions on the object and the type of permissions. The type may include the rights for other entities to modify the object's properties such as its owner (*WriteOwner*), password (*ForceChangePassword*) among other properties, and full control over the object (*GenericAll*). These permissions are generally granted to groups of users [26] on OUs containing objects. In metagraphs, let V be a group of users, Q be an OU containing objects and p be the ACL permission type. A metagraph edge $e = \langle V, Q \rangle \in E$ with $p \in P_e$ denotes the ACL permission between a set of users and a set of objects.

There are permissions not specified in ACL and are designated as non-ACL permissions, mostly permissions on computers. They include the rights for a user to remotely log on to a computer (*CanRDP*), execute commands on the machine (*ExecuteDCOM*), and other operations. In metagraphs, a set V represents the user group, Q denotes an OU containing computers and p represents the non-ACL permission type.

Fig. 4 demonstrates these AD permissions in a metagraph model of the company in Fig. 3. To simplify the visualization, we reduce the number of tiers to 1 tier and only consider user and group objects as admin entities in the Admin OU. The example AD system has 2 administrative users, 6 regular or non-administrative users, and some computers.

B. Metagraph generation process in ADSynth

1) Node generation: There are three steps required to generate metagraph nodes for an AD system: (1) generating OU architecture and groups, (2) assigning objects to OUs, and (3) adding members to groups. OUs and groups are sets of AD objects, representing nodes in a metagraph.

Step 1: Using OUs, ADSynth generates a tier structure implementing the guidelines in [20]. The number of tiers is a parameter in ADSynth. A tier structure containing 3 tiers is illustrated in Fig. 3. As recommended by best practices [10], [23], [27], administrative and regular objects are separated into different OUs. In the tier administrative model [20], Tier 0

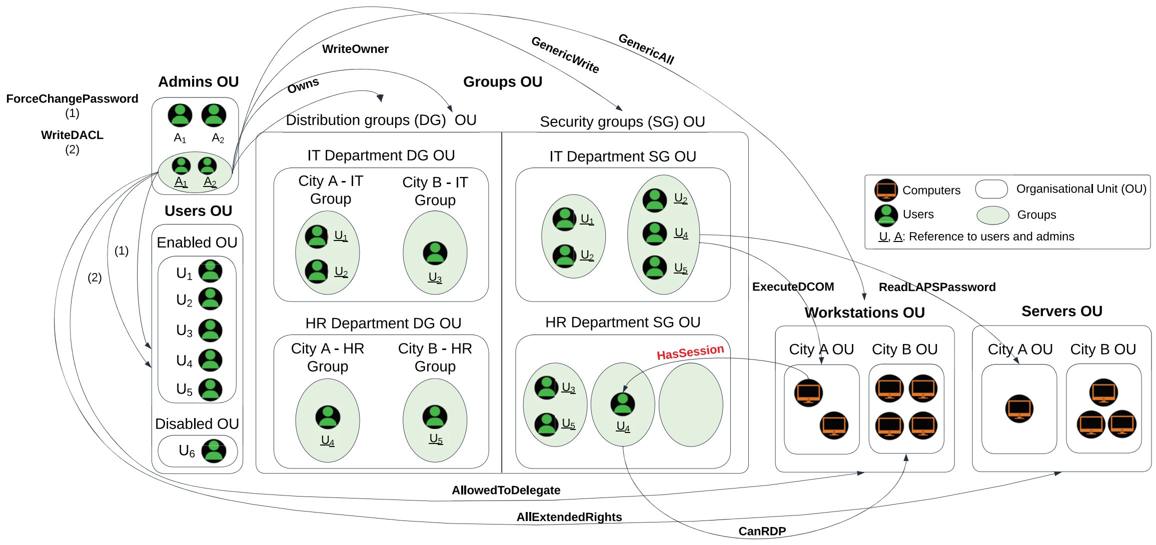

Fig. 4. The metagraph representation of a company's AD system, representing the set-to-set mapping - an intrinsic nature of Active Directory

contains the highest-privileged objects that have administrative control of the entire AD system. Tier 1 is dedicated to enterprise servers, containing sensitive business data. Regular objects, including non-administrative users, groups, and computers (workstations and servers), are placed in tier 2 onwards.

ADSynth uses lists of departments in an enterprise, branch locations, and the number of root folders containing data of the organization to generate distribution groups for each department at every location following [10], [23], [12], [24]. In addition, within a department, a number of security groups equivalent to the number of root folders are created with access rights, or NTFS permissions [28], [29]. The groups are placed in the corresponding OUs, as illustrated in Fig. 3. AD objects are later added to groups as group members.

Step 2: Assigning users and computers to OUs. In ADSynth, there are 3 types of users: admin, enabled, and disabled user accounts. Users of each type are equally distributed to the equivalent OUs at every tier. Similarly, computers are separated into different types based on their nature, such as privileged access workstations (PAW), regular workstations, and servers. Computers of each type are then uniformly allocated to the according OUs within each tier.

Step 3: Adding members to groups. For each user, ADSynth randomly samples several groups at the same tier and assigns the user to these groups as group members. This incorporates the principle of Least Privilege, part of the Tier Model of Microsoft [30], which prevents users from gaining privileges of admin groups at higher tiers.

2) Edge generation: After generating metagraph nodes, ADSynth creates metagraph edges to connect the nodes.

Control and Management. In a tier model, administrators have control over resources at their tier and in the tiers below, based on their roles [31]. In contrast, restrictions are in place to prevent less privileged users from taking control of crucial assets in higher tiers [31].

Algorithm 1 simulates these interactions by generating edges $e = \langle V, Q \rangle \in E$ with administrative groups as the invertex V and potential resources, including sets of users, computers and groups, as the outvertex Q. The algorithm iterates through each tier. At every tier, the maximum number of resources allocated to permissions per admin group would be capped at a proportion p_r of the total number of all possible resources which is calculated by $total\_resources(i, k, is\_acl)$ (line 3). For each permission, the target set Q is randomly chosen from possible resources within the permitted range of tiers using function $random\_resources(i, k, is\_acl)$ (line 6). The type of permission, which is either an ACL or non-ACL permission, is determined with user-specified probabilities by the function $random\_permission(is\_acl)$.

Logon restrictions. In a tier model, the following restrictions are enforced on cross-tier connections [32], [33]:

- *Credentials from a higher-privileged tier (e.g. Tier 0 Admin) must not be exposed to lower-tier systems (e.g. Tier 1 or Tier 2 systems)*
- *Lower-tier credentials can use services provided by higher-tiers, but not the other way around.*

Users are restricted to logging on to computers at their assigned tier and the higher tiers if necessary for work purposes [31]. In our metagraphs, a user or a computer is represented as a set of size 1. Algorithm 2 simulates user sessions *including allowed cross-tier connections* by generating edges $e = \langle V, Q \rangle \in E$ with computers as the invertex V and users as the outvertex Q. A metagraph edge $e = \langle V, Q \rangle$ denotes that a user has an interactive session on a computer. Algorithm 2 traverses through all tiers in the AD structure to generate sessions for users at each tier. It limits the number of user sessions by a proportion p_s of allowed computers (line

Algorithm 1: Generate edges for AD permissions

```
Input: k - number of tiers
       O - sets of organisational units
       AG - sets of admin groups, AG(t) is a set of admin
       groups at tier t
       $p_r$ - the percentage of possible target resources
1  Function generate_control(k, AG, O, $p_r$, is_acl)
2  for $t \leftarrow 0$ to $k-1$ do
3      $n_r \leftarrow total\_resources(i, k, is\_acl) \times p_r$
4      for $g \in AG(t)$ do
5          for $iterator \leftarrow 1$ to $n_r$ do
6              $target \leftarrow random\_resources(i, k, is\_acl)$
7              $permission\_type \leftarrow random\_permission(is\_acl)$
               /* Edge generation */
8              Generate $e = \langle g, target\rangle, permission\_type \in P_e$
9          end
10     end
11 end
12 generate_control(k, AG, O, $p_r$, True) /* ACL */
13 generate_control(k, AG, O, $p_r$, False) /* non-ACL */
```

3). The algorithm randomly samples the according quantity of computers (line 4) and generates edges between the user and these computers (line 5).

Algorithm 2: Generate sessions

```
Input: k - number of tiers
       $\mathcal{C}(t,k)$ - a set of computers at tier t and higher tiers
       $\mathcal{U}(t)$ - a set of users at tier t
       $p_s$ - the maximum proportion of computers a user
       can log on at each tier
1  for $t \leftarrow 0$ to $k-1$ do
2      for $u \in \mathcal{U}(t)$ do
3          $num\_sessions \leftarrow random(0, p_s \times |\mathcal{C}(t,k)|)$
4          for $c \in random\_sample(\mathcal{C}(t,k), num\_sessions)$ do
               /* Edge generation */
5              Generate $e = \langle c, u\rangle$, $e$ labelled HasSession
6          end
7      end
8  end
```

3) Misconfiguration generation: The third critical component of ADSynth is simulating misconfigurations through exceptions and violations of best practices [20], which are common in realistic AD systems due to the aggregate intrinsic complexity of Active Directory [18].

The first type of misconfiguration occurs when a highly privileged user is allowed to log on to lower-tier computers with their credentials (a higher-privileged user should access computers at lower tiers using a different account created only for this purpose). When this misconfiguration occurs and a high-privilege user authenticates to a lower-tier computer, a malicious actor at lower tiers could easily harvest credentials by injecting malicious code into the authentication process of these computers and obtain the administrative credentials [32], [33] to make lateral movements to critical parts of the system.

Algorithm 3 simulates these misconfigurations by creating edges $e = \langle V, Q\rangle \in E$ with the invertex V representing computers and the outvertex Q denoting users. A metagraph edge $e = \langle V, Q\rangle$ signifies a violated session of a user on a computer. The algorithm first determines the number of breached sessions in the entire Active Directory based on user input (line 1). For each violation, it randomly samples a user (regular or administrative role) at any tier (lines 3-5) and a computer at lower tiers (lines 6-7). An edge, labeled as *HasSession*, is then generated from the computer to the user.

Algorithm 3: Misconfiguration in user sessions

```
Input: k - number of tiers
       u - total number of users
       perc_misconfig_sessions - the percentages of users
       with misconfigured sessions

1  $num\_misconfig \leftarrow perc\_misconfig\_sessions \times u$
2  for $i \leftarrow 1$ to $num\_misconfig$ do
       /* Determine user's role and tier */
3      $is\_admin \leftarrow random(True, False)$
4      $user\_tier \leftarrow random(0, k-1)$ /* excluding the
           last tier */
5      $user \leftarrow random\_user(is\_admin, user\_tier)$
       /* Determine computer's tier */
6      $comp\_tier \leftarrow random(user\_tier + 1, k)$
7      $comp \leftarrow random\_comp(comp\_tier)$
       /* Edge generation */
8      Generate $e = \langle comp, user\rangle$, $e$ labelled HasSession
9  end
```

The second class of misconfiguration arises when permissions are granted incorrectly. For example, a regular user at a lower tier is allowed to execute code on privileged computers at a higher privilege level via DCOM (ExecuteDCOM) and perform other operations (aka non-ACL permissions). These permission misconfigurations allow attackers to escalate from lower tiers to higher tiers. We model these misconfigurations in metagraphs by using edges $e = \langle V, Q\rangle$ with users as the invertex V and computers as the outvertex Q. The edges show that users have misconfigured permissions on computers.

Algorithm 4 generates a user-specified number of misconfigured permissions (line 1). For each permission, a regular user and a computer at higher tiers are randomly sampled (lines 3-6). A misconfigurd edge is generated from the user to the computer (line 8) with random non-ACL permission (line 7).

ADSynth security parameters The security level of the generated Active Directory graph is specified by 2 parameters. The parameter $perc\_misconfig\_sessions$ in Algorithm 3 and the parameter $perc\_misconfig\_permissions$ in Algorithm 4 determine the numbers of misconfigured user sessions and permissions in the AD system respectively. High values for these parameters lead to an increased number of misconfigurations, raising the risk of attackers escalating privileges and reaching Domain Admins, thus making the network vulnerable. Conversely, low parameter values reduce misconfigurations, resulting in a secure system.

ADSynth Ouput The output of ADSynth is an Active Directory (AD) attack graph in a JSON format of Neo4J [34], which can be loaded and processed in BloodHound [19]. The default generated graph is a set-to-set mapping AD attack

Algorithm 4: Misconfiguration in permissions

```
Input: k - number of tiers
       u - total number of users
       perc_misconfig_permissions - the percentage of users
       with misconfigured permissions

1 $num\_misconfig \leftarrow perc\_misconfig\_permissions \times u$
2 for $i \leftarrow 1$ to $num\_misconfig$ do
      /* Determine regular user's tier */
3     $user\_tier \leftarrow random(1, k)$ /* excluding the
         first tier */
4     $user \leftarrow random\_user(False, user\_tier)$
      /* Determine computer's tier */
5     $comp\_tier \leftarrow random(0, user\_tier)$
6     $comp \leftarrow random\_comp(comp\_tier)$
      /* Edge generation */
7     $permission \leftarrow random\_permission(False)$
8     Generate $e = \langle user, comp \rangle, permission \in P_e$
9 end
```

graph. ADSynth has a parameter to convert the default graph into an element-to-element AD attack graph.

IV. Experiments and Results

We compare the performance of ADSynth with DBCreator and ADSimulator. In addition, we validate the realism of ADSynth graphs with an AD system of a University. The University AD system has 100K nodes and 1.2 million edges.

Experiment Setup. We perform experiments across diverse graph sizes, spanning from 1000 nodes to a million nodes. Utilising ADSynth, we simulated different levels of security, encompassing highly secure and vulnerable AD systems. For graphs with 100K nodes, which are similar to the size of the realistic AD system, we label them as AD100.

A. Execution time

The existing tools DBCreator and ADSimulator use a simple graph model and rely on Neo4J database [35] to generate Active Directory attack graphs. The metagraph model allows us to generate nodes and edges using groups of entities, significantly reducing the complexity of the graph. In addition to the metagrah model, during the execution of DBCreator and ADSimulator, we identify a large number of data transactions, leading to a very significant latency. ADSynth eliminates the latency by implementing a local graph database with functions replicating Neo4J. ADSynth's database facilitates insertion and retrieval operations for nodes and edges at a constant time while maintaining optimal storage efficiency.

Both the metagraph model and the engineering effort contribute to a significant reduction in the running time of ADSynth. Table I displays the average running times obtained from 20 executions for each tool across varying graph sizes. The standard deviations for multiple executions are included following the +/- sign. In particular, ADSynth constructs a 100K-node AD graph, which is equivalent to the scale of a University AD system, within approximately 21 seconds. ADSimulator took 31 minutes while DBCreator could not produce graphs of this size.

TABLE I
RUNNING TIMES OF DBCREATOR, ADSIMULATOR, AND ADSYNTH

$\|V\|$	DBCreator[s]	ADSimulator[s]	ADSynth[s]
1000	27.296±1.89	12.479±1.775	0.027±0.031
5000	257.046±1.608	70.492±0.855	0.993±0.105
10000	819.433±1.678	190.439±1.579	1.886±0.153
50000	-	757.135±5.361	9.774±0.201
100000	-	1849.273±30.748	21.304±0.958
500000	-	-	121.725±7.281
1000000	-	-	257.387±12.28

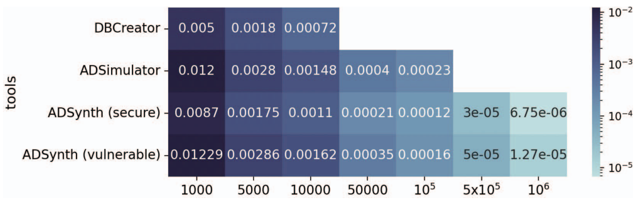

Fig. 5. Graph density

B. Active Directory Graph Properties

We evaluate ADSynth using common metrics in Active Directory (AD), such as graph density, numbers of user sessions, and other metrics [36], [37]. We compare these against the University AD system.

Graph density is defined as $\frac{|E|}{|V| \cdot (|V|-1)}$, with $|V|$ as the number of nodes and $|E|$ as the number of edges in a graph. Fig. 5 illustrates ADSynth's capability to generate networks with varying densities under different network settings. In particular, the secure system AD100 by ADSynth has a density of 0.00012, closely resembling the University AD system at 0.00008. By contrast, vulnerable AD systems by ADSynth possess greater density due to many violated connections. Assuming that there are $|X|$ violations, equivalent to $|X|$ edges in the attack graph, the graph density of a vulnerable system is $\frac{|E|+|X|}{|V| \cdot (|V|-1)}$, greater than the density of a secure system without violations by $\frac{|X|}{|V| \cdot (|V|-1)}$. DBCreator and ADSimulator graphs also exhibit notably high density, but this originates from the generation of permissions (ACL and non-ACL) that diverges from established rules and best practices [38], [39], [40].

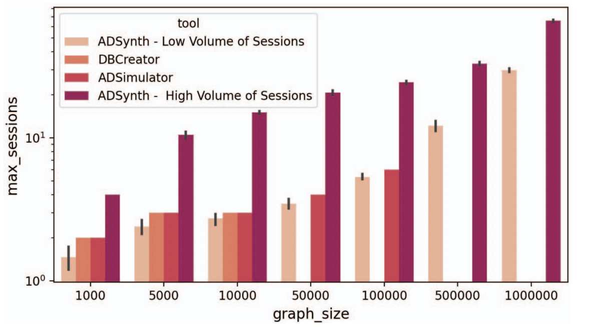

Fig. 6. Comparison of volumes of user sessions for increasing size of the networks produced by DBCreator, ADSimulator, and ADSynth.

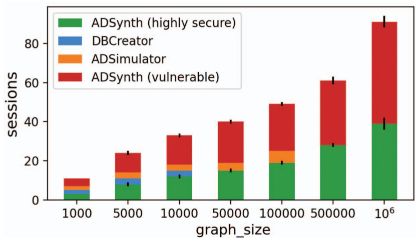

Fig. 7. Comparison of volumes of user sessions on various AD systems

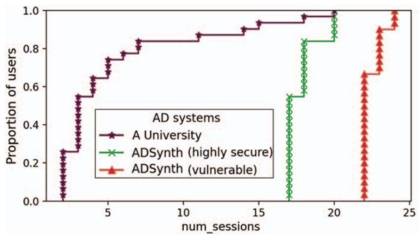

Fig. 8. Comparison of user session distribution between ADSynth and the University AD system

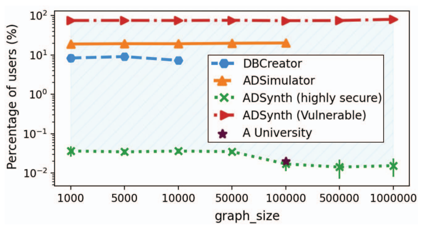

Fig. 9. Proportions of users granted access to DA, Y-axis set to a log scale

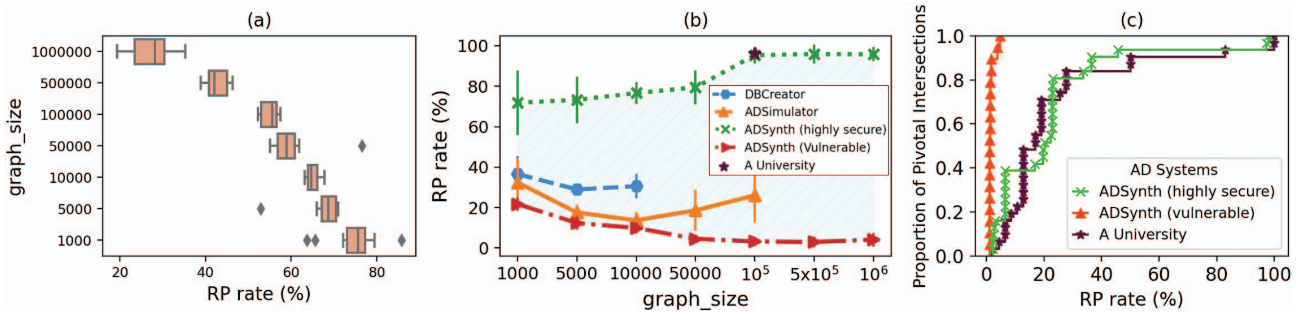

Fig. 10. (a) Relationship between the highest Route Penetration Rate (RP rate) and graph sizes ; (b) Peak RP rates generated by DBCreator, ADSimulator, and ADSynth; (c) RP rates between the AD system of a University and ADSynth networks

User sessions. Fig. 6 presents the highest session counts per user, denoted as user sessions, in various networks. The figure showcases ADSynth's ability to generate a range of user logons, which none of the other tools can do.

Fig. 7 illustrates the peak user session counts in ADSynth networks under various security settings and in networks generated by the other tools. In detail, the session counts in the vulnerable networks by ADSynth surpass those of other networks, attributed to the prevalence of violated cross-tier sessions. Additionally, the figure highlights a similarity between our secure network AD100 and the University AD system with approximately 20 sessions per user at peak. ADSynth achieves this result through a parameter to tune the maximum number of sessions per user.

Fig. 8 depicts the distributions of user sessions among the top 30 users in the University AD system and ADSynth networks. In particular, ADSynth displays a constrained spread. The explanation is as follows. In a realistic University Active Directory (AD) system, there are many user roles and users can access multiple computers depending on their roles. While most normal users typically log on to 1-2 machines, teaching staff may utilize 3-4 for various tasks such as experiments, presentations, communication, and other tasks. A small percentage of staff may access 5, 10, or even 15 machines, shaping the distribution of user sessions in Figure 8. The distribution also indicates that among 30 users with the most sessions out of 30K users of the University AD system, many have fewer than 2 sessions, emphasizing that the majority of remaining users also have minimal session activity as expected above. However, in ADSynth graphs, users can log onto any number of permitted computers, regardless of their organizational roles. Figure 8 shows that in the secure ADSynth graph, the highest upper bound of user sessions is approximately 20. The numbers of sessions of the top 30 users already exceed 17, nearing the upper bound of 20. With roughly 30K users in the AD system, the figure indicates that a significant number of users have high session counts. The same pattern is observed in the vulnerable ADSynth graph. Addressing this mismatch in the distribution of user sessions is a focus for future development of ADSynth.

C. Security metrics in Active Directory

We focus here on the proportions of regular users having attack paths to Domain Admins, as depicted in Fig. 9. By manipulating the two misconfiguration parameters in algorithms 3 and 4, ADSynth can simulate networks across a spectrum of security levels. This ranges from a vulnerable system harboring an excessive proportion of concerned users to a highly secure system with only a minimal presence of such users. Notably, the secure network AD100 by ADSynth closely mirrors the University AD system, with 0.02% of regular users possessing access to Domain Admins.

Route Penetration Rate. We define the *Route Penetration Rate*, or *RP rate* as the percentage of shortest paths from regular users to Domain Admins passing through that node. Nodes with large RP rates are recognized as *choke points*, indicating a focal point of concentrated network traffic [41].

Fig. 10a illustrates that with constant security settings, an increase in the number of nodes leads to a decrease in the peak RP rate. The number of violated sessions is directly proportional to the graph sizes following the violation rate $perc\_misconfig$ in Algorithms 3 and 4. As the network expands, more nodes and violated connections are created. With more emerging escalated paths from regular users to Domain Admins, the traffic flow is spread, resulting in a reduction of RP rate at choke points.

The comparison of the peak RP rates between ADSynth, DBCreator, and ADsimulator is depicted in Fig. 10b. Specifically, ADSynth demonstrates the capacity to generate vulnerable systems with low RP rates and realistic secure networks with high RP rates at choke points, closely mirroring the University AD system. Conversely, graphs produced by DBCreator and ADSimulator consistently display a moderate RP rate within the range of 20-40%, thereby proving incapable of reproducing the behavior of realistic AD systems.

In Fig. 10c, we analyze the distributions of the top 30 nodes with the highest RP rates in ADSynth graphs and the University AD system. It shows a similar distribution pattern between the secure network by ADSynth and the University AD system. Specifically, secure network AD100 by ADSynth holds some choke points with more than 80% of paths towards Domain Admins passing through, while the other nodes experience a low RP rate. This mimics the behavior in the University AD system. Conversely, in the vulnerable network, no significant choke points are identified.

V. Applications

Several recent research studies [14], [21], [4] and industry solutions [19], [15] about attack path management, have developed solutions to improve the security of Active Directory (AD), but these methodologies have only been tested with limited real datasets or datasets generated by DBCreator and ADSimulator. With ADSynth datasets, the applicability and generalization of these works can be evaluated on various AD configurations, and improvements can be made to enhance their performance when applied in real-world AD systems. Furthermore, AD admins could also find the tool useful in generating attack graphs for practice, training, and simulation.

Commercial attack path management tools - BloodHound [19] and GoodHound [15]: BloodHound is an AD mapping and attack path management tool [42]. The BloodHound Attack Path Management solution is proprietary, we do not have access to the code. AD admins with a license to BloodHound Enterprise could use our datasets to evaluate the effectiveness of BloodHound under various scenarios to help guide their decisions. GoodHound is an open-source tool that identifies the weakest edges in an AD system. This information allows AD defenders to eliminate edges with substantial attack traffic in a prioritized order, effectively reducing the number of attack paths. We evaluated GoodHound on datasets generated by ADSynth and ADSimulator, each containing 100K nodes. We observed a significant disparity between the 2 datasets. In this experiment, after identifying the weakest links in a prioritized manner for each graph, we determined the number of links that needed to be removed to eliminate all shortest attack paths. Figure 11 shows that the typical number of

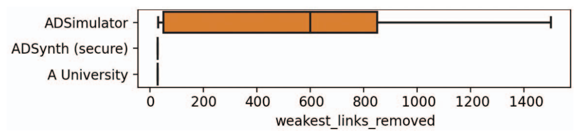

Fig. 11. Application of GoodHound on various datasets

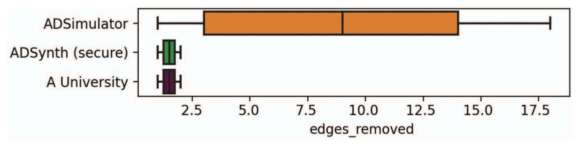

Fig. 12. Testing the Double Oracle algorithm on various datasets

weakest links removed necessary to eradicate projected attack paths is approximately 600. However, in the ADSynth graph (secure), only 29 weakest links require removal, mirroring the outcome observed in the realistic University AD graph.

Scalable Double Oracle Algorithm [14]: An optimal strategy is suggested to eliminate attack paths in AD environments through edge cutting at a fast speed. We applied the algorithm on a realistic dataset generated by ADSynth and observed the major difference compared to experiments on ADSimulator datasets in the paper. The datasets comprised approximately 100K nodes each. As shown in Figure 12, while in the ADSimulator graph, the median number of edge cuts required to fully eliminate attack paths of the shortest length is roughly 8, the minimum level of edge removal in the ADSynth graph (secure) does not exceed 2 - resembling the result on the realistic University AD system.

Scalable Edge Blocking Algorithms [4]: The work was developed on a scenario when defenders block a set of edges and attackers take the shortest unblocked path. The paper introduces algorithms for defenders to maximize the length of the unblocked attack path. We ran the algorithm IP (Integer program based on Kernelization) and Iterative LP on graphs of ADSimulator, ADSynth, and a University AD system. Each of the graphs contains approximately 100K nodes. For the ADSimulator graph, the resulting success rates of attackers after defenders conduct edge blocking are 0.149 (IP) and 0.093 (IterLP). When running on the ADSynth graph (secure) and the University AD system, the algorithms report an error in the graph setup. We conjecture that the algorithm fails on more realistic graphs.

VI. Conclusion

Security solutions and attack response tools rely on accurate data and models, but AD datasets are confidential, and no current tool can produce realistic, large AD graphs that meet the design criteria. We develop *ADSynth* to generate synthetic Active Directory attack graphs that meet these realistic design and scalability requirements. *ADSynth* uses metagraph abstractions to represent the intrinsic nature of Active Directory, set-to-set mappings. Experiments show its scalability and the realism of the generated attack graphs.

References

[1] A. Binduf, H. O. Alamoudi, H. Balahmar, S. Alshamrani, H. Al-Omar, and N. Nagy, “Active directory and related aspects of security,” in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, 2018, pp. 4474–4479.

[2] T. Matthews. (2019, August) Active directory: a new product for a new millennium. [Online]. Available: https://www.exabeam.com/information-security/cybersecurity-calendar-active-directory/

[3] V. Jakkal. (2022) Cyber signals: Defending against cyber threats with the latest research, insights, and trends. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2022/02/03/cyber-signals-defending-against-cyber-threats-with-the-latest-research-insights-and-trends/

[4] M. Guo, J. Li, A. Neumann, F. Neumann, and H. Nguyen, “Practical fixed-parameter algorithms for defending active directory style attack graphs,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 9360–9367.

[5] M. Guo, M. Ward, A. Neumann, F. Neumann, and H. Nguyen, “Scalable edge blocking algorithms for defending active directory style attack graphs,” *Proc. AAAI 2023*, 2023.

[6] I. F. et al. (2021) Best practices for securing active directory. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/best-practices-for-securing-active-directory

[7] A. R. W.S and V. Rohan. (2020, april) Bloodhound database creator. [Online]. Available: https://github.com/BloodHoundAD/BloodHound-Tools/tree/master/DBCreator

[8] D.Rowe. Badblood by secframe. [Online]. Available: https://www.secframe.com/badblood/

[9] N. Carolo. (2022, Jun.) adsimulator: a realistic simulator of active directory domains. [Online]. Available: https://github.com/nicolas-carolo/adsimulator

[10] D.Ruest and N.Ruest, *Windows Server 2003: best practices for enterprise deployment*. McGraw-Hill Osborne Media, 2003.

[11] Microsoft. (2023, May) Ad ds design and planning. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/ad-ds-design-and-planning

[12] M. M. Alan Bruchill. (2010) Active directory structure guidelines – part 1.

[13] A. Basu and R. W. Blanning, *Metagraphs and their applications*. Springer Science & Business Media, 2007, vol. 15.

[14] Y. Zhang, M. Ward, M. Guo, and H. Nguyen, “A scalable double oracle algorithm for hardening large active directory systems,” in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 993–1003.

[15] A. Morris, “Goodhound,” https://github.com/idnahacks/GoodHound, accessed: 2022-12-14.

[16] Microsoft. (2018, May) Group policy objects. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/policy/group-policy-objects

[17] H. S. Lallie, K. Debattista, and J. Bal, “A review of attack graph and attack tree visual syntax in cyber security,” *Computer Science Review*, vol. 35, p. 100219, 2020.

[18] J. Dunagan, A. X. Zheng, and D. R. Simon, “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*, 2009, pp. 305–320.

[19] W. S. Andy Robbins, Rohan Vazarkar, “Bloodhound: Six degrees of domain admin,” https://bloodhound.readthedocs.io/en/latest/index.html, accessed: 2022-12-14.

[20] Microsoft. (2014) Mitigating pass-the-hash and other credential theft, version 2.

[21] H. Q. Ngo, M. Guo, and H. Nguyen, “Near optimal strategies for honeypots placement in dynamic and large active directory networks,” in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2023, pp. 2517–2519.

[22] Microsoft. (2020, August) Object names and identities. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/ad/object-names-and-identities

[23] S. Majumder. Active directory : Design considerations and best practices. [Online]. Available: https://social.technet.microsoft.com/wiki/contents/articles/52587.active-directory-design-considerations-and-best-practices.aspx

[24] K. Joyce. (2023, April) Active directory security groups vs distribution groups. [Online]. Available: https://blog.netwrix.com/2023/04/28/active-directory-security-groups-vs-distribution-groups/

[25] F. Malaeb. (2021) Understanding get-acl and ad drive output. [Online]. Available: https://devblogs.microsoft.com/powershell-community/understanding-get-acl-and-ad-drive-output/

[26] Microsoft. (2023, July) Access control overview. [Online]. Available: https://learn.microsoft.com/en-us/windows/security/identity-protection/access-control/access-control

[27] R. Smith. Protect privileged active directory credentials using a tiered administrative model. [Online]. Available: https://www.youtube.com/watch?v=t4I2saNpoFE&list=WL&index=73&t=1182s

[28] C. Russel, S. Crawford, and J. Gerend, *Microsoft windows server 2003 administrator’s companion*. Microsoft Press, 2003.

[29] S. Parkinson and A. Crampton, “A novel software tool for analysing nt file system permissions,” *arXiv preprint arXiv:1312.2804*, 2013.

[30] Microsoft. (2023, March) Enhanced security admin environment. [Online]. Available: https://learn.microsoft.com/en-us/security/privileged-access-workstations/esae-retirement?source=recommendations

[31] R. Smith. (2017, October) Why you should use microsoft’s active directory tier administrative model. [Online]. Available: https://petri.com/use-microsofts-active-directory-tier-administrative-model/

[32] Microsoft. (2023) Credentials processes in windows authentication. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/credentials-processes-in-windows-authentication

[33] S. Metcalf, “Unofficial guide to mimikatz & command reference,” *Active directory security*, 2018.

[34] N. Labs. (2023) Export to json. [Online]. Available: https://neo4j.com/labs/apoc/4.1/export/json/#export-database-json

[35] Neo4j, Inc., “Neo4J graph database,” https://neo4j.com/product/neo4j-graph-database/, 2023, version 5.13.

[36] F. Force. (2023, May) Bloodhound - calculating ad metrics 0x01. [Online]. Available: https://falconforce.nl/bloodhound-calculating-ad-metrics-0x01/

[37] ——. (2023, May) Bloodhound - calculating ad metrics 0x02. [Online]. Available: https://falconforce.nl/bloodhound-calculating-ad-metrics-0x02/

[38] Microsoft. (2023, March) Enterprise access model. [Online]. Available: https://learn.microsoft.com/en-us/security/privileged-access-workstations/privileged-access-access-model

[39] ——. (2023, March) Privileged access: Strategy. [Online]. Available: https://learn.microsoft.com/en-us/security/privileged-access-workstations/privileged-access-strategy?source=recommendations

[40] ——. (2023) Tier model for partitioning administrative privileges. [Online]. Available: https://learn.microsoft.com/en-us/microsoft-identity-manager/pam/tier-model-for-partitioning-administrative-privileges

[41] A. Robbins. (2021, October) Why choke-point analysis is essential in active directory security. [Online]. Available: https://www.darkreading.com/application-security/why-choke-point-analysis-is-essential-in-active-directory-security

[42] J. Dibley. (2022) Finding abusable active directory permissions with bloodhound. [Online]. Available: https://blog.netwrix.com/2022/12/09/bloodhound-active-directory/#:~:text=BloodHound%20is%20an%20Active%20Directory,in%20a%20source%20Windows%20domain.