Scalable Active Directory Defense with α-Metagraph

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 developed for Windows domain networks, is a frequent target for attackers due to its widespread adoption and the sensitive information it manages. Most existing attack path management solutions in ADs rely on simplistic node-to-node graphs, disregarding dependencies between edges. Specifically, in AD systems, there exists policy-defining edges that represent permissions on a set of objects. A single defensive action to remove such an edge can eliminate multiple permissions associated with all objects within the set - effectively removing all other edges connecting nodes in this set. In this paper, we propose a rigorous model that formalizes this concept of *node-to-set* mapping using α-metagraph - a novel high-order graph model for capturing dependencies in AD systems. We present an algorithm for constructing the α-metagraph from Active Directory data. Furthermore, we extend the current state-of-the-art AD defensive solution algorithm - Spiral - to operate with the α-metagraph model, taking advantage of policy-defining edges in AD network defense. Our extensive experiments demonstrate that the proposed α-Spiral Algorithm, applied to α-metagraphs, delivers timely and superior defense strategies, mitigating more attack sources within time and budget constraints than the original Spiral algorithm on node-to-node graphs.**

I. Introduction

Simple directed graphs are commonly used in traditional attack graphs, including the Active Directory attack graph model [1]. In these graphs, nodes represent objects such as users, computers, groups, organizational units (OUs), and other network resources. Each edge represents a possible step that attackers can exploit to move laterally through the network from one entity to another. In these graphs, all security vulnerabilities that allow lateral movements are represented using a simple graph notion of single node-to-node connections. For example, security permission between a user and a set of objects is represented by a set of independent edges between the user and every element in the set. We call this model the *node-to-node* mapping. Several software tools have been developed for analyzing and visualizing this model in Active Directory attack graphs, including the popular BloodHound tool [2].

AD attack graphs are used in practice to guide remedy defensive solutions that remove potential attack paths from low-privilege nodes (also called entry nodes) to high-value targets. Paths are removed through the elimination of edges - or the security loopholes that allow the edges to be abused. Removing an edge in an AD graph is a costly and time-intensive process, as each removal requires manual examination and approval to ensure it does not interfere with standard operations or trigger cascading effects. These evaluations, typically conducted by IT operational teams, can take several days or even weeks to complete. As a result, there is a growing body of defense strategies to cut attack paths to high-value targets from entry sources while minimizing the number of necessary removal actions both in academia [3]–[6] and industry [7]. These optimization problems are known NP-hard problems on attack graphs. Apart from simple heuristics that do not guarantee the removal of all threats, iterative algorithms such as the Spiral algorithm [4] and the Bloodhound Enterprise solution [8] depend heavily on the size of the AD attack graphs.

Current approaches to building AD attack graphs by focussing on simple entity-to-entity relationships, unfortunately, tend to result in unnecessarily large graphs that ignore dependencies between the edges. It is common for a medium organization to have millions of nodes and edges with billions of attack paths in these models [9]. Consider a simple AD attack as shown in Figure 1. In this example, the Group has Permission-A control over an Object set containing User 1 and User 2. Control over an object set inherently grants the group authority over all individual elements within that set. Consequently, the Group can exploit Permission B associated with User 1 to gain access to the Domain Admins (DA) account. Similarly, the Group can achieve the same objective by leveraging Permission C associated with User 2. To mitigate this attack, we only need to revoke the Permission A of the Group over the Object set. However, simple graph models cannot capture these dependencies as they have no abstract representations for sets in their graphs. Most existing models would approximate these relationships by including an edge between the Group and the Object set and additional edges directly linking the Group to User 1 and User 2, reflecting the group's control over individual elements in the set. Consequently, eliminating Permission A in this simple graph model requires the removal of three edges: one between the Group and the Object set and two additional between the Group and the users. This results in a higher number of defense actions than what is really needed in a realistic AD system.

In order to reduce both the size of the graphs and the number of defensive actions, we need a model that captures the polyadic relationships in security policies. No such model currently exists for AD systems. We develop such a model in this work. In our model, we use a high-order graph abstraction to model the security settings that map one element to a set of elements. For the example in Figure 1, we use one single edge to represent permission A. But the destination node of this edge is now a set of elements - not a single element. This necessitates the development of a new graph abstraction that we call α-metagraph. With this new abstraction, edges that define the security permission of an element over every element in another set are called *policy-defining edges*. In the above example, by removing the policy-defining edge for Permission A, we can revoke the Group permissions not only over the Object set but also over every element in the set, thereby severing all attack paths from the Group to the users and the DA - resulting in significantly fewer defensive actions than the simple graph model. We assume a realistic threat model where an attacker exploits existing permissions and group memberships in Active Directory to escalate privileges from low-privilege entry nodes toward high-value targets such as Domain Admins. By modeling these chained attack paths with the α-metagraph and strategically removing minimal policy-defining edges, our approach directly targets and disrupts the attacker's ability to traverse these escalation routes.

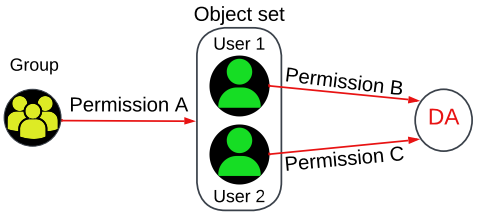


Fig. 1: Attack paths in a realistic Active Directory configuration. Assuming attackers control the Group, they can exploit Permission A to access the Object set and its elements. Subsequently, attackers may proceed toward Domain Admins (DA) by exploiting Permission B or Permission C of two users in the set.

Our key contributions in this paper are:

- We develop a novel α-metagraph abstraction model that can faithfully capture the polyadic relationships in AD permission configurations (node-to-set mapping); together with an algorithm to build α-metagraph model for AD systems from data collected by SharpHound [10], an open-source AD data collector.
- We show that there exists a graph transformation to map α-metagraphs to the digraph model - preserving the security properties of the original AD system. This analysis indicates that current AD models while sub-optimal can still be used to secure AD systems - albeit with more defensive actions than necessary.
- We propose the α-Spiral algorithm that can eliminate a greater number of attack sources by taking advantage of α-metagraphs than any existing algorithms. We also provide a mathematical proof supporting its effectiveness.

Our experiments on realistic datasets show that our α-Spiral algorithm on the proposed α-metagraph model provides optimal and timely defense strategies compared to the state of the art Spiral algorithm on the node-to-node mapping model.

II. Background

A. Existing AD Attack Graph Models

Active Directory [11] is a proprietary directory service that manages network resources and permissions. AD stores resources as objects, including users, computers, groups, and other entities. Permissions in AD are rules that grant specific control or access rights over objects. For example, Domain Admins group has administrative rights on all computers, or a user is allowed to log on to a computer at a specific tier. In a medium-sized Active Directory (AD) network, there are typically hundreds of objects and millions of permissions interconnecting these objects [9]. The inherent complexity of the AD system often leads to misconfigurations, which can create multiple attack paths to Domain Admins, the most privileged account within AD.

An attack graph for Active Directory was first developed in [1]. In this model, every object in an AD system is a node, the directed edge connecting 2 nodes $e = \langle u, v \rangle \in E$ represents a form of control or misconfiguration that allows an attacker to traverse from u to v via the edge e. The graph captures identity-based snowball attacks where after compromising a machine, the attackers can steal the credentials of users currently logged in to perform lateral movements and compromise other machines. AD attack graph management tools built on this model are leveraged and commercialized by the popular Bloodhound [2] software, recommended by the US Department of Homeland Security for hardening AD systems.

BloodHound uses SharpHound data of AD objects as input to create an AD attack graph. SharpHound operates by leveraging native Windows API functions and LDAP queries to extract sensitive data from domain controllers and domain-joined systems from a domain joined user account [12]. SharpHound collects various types of information, including security group memberships, domain trusts, abusable permissions on AD objects, OU tree structure, Group Policy links, and relevant properties of computer, group, and user objects [12]. Additionally, it attempts to gather data on local group memberships and active user sessions from domain-joined Windows computers [12]. Each object of SharpHound data has a property indicating which entities have what permissions on the object [2]. Using information about the relationship between objects/nodes, BloodHound builds directed edges to connect these nodes. We refer to this simple graph model of AD system as the node-to-node mapping model.

Attack Path Management Solutions

The primary application of AD attack graphs in defensive security is for the elimination of edges to minimize the number of attack paths and entry nodes that have access to Domain Admins [3], [4], [6], [13]. Removing an edge from the attack graph means the revocation of a permission between two entities in the actual AD network. This is a costly process and many optimization solutions exist to minimize the number of defensive actions. Among these, the state-of-the-art approach in [4] guarantees the removal of all attack paths, albeit with potentially exponential worst-case run time.

The **Spiral algorithm** in [4] is an anytime Inter-Linear Programming (ILP) algorithm for the NP-Hard problem of eliminating paths from entry nodes to a privileged node where an acceptable defense strategy is always provided within a time limit. Spiral first formulates the AD optimization problem as an ILP [4] and subsequently solves this ILP equation. Instead of applying the ILP formulation to the entire graph, which is computationally expensive due to the large number of variables involved, the algorithm processes subgraphs. A k-subgraph in the algorithm is defined as a graph containing exactly k entry nodes with attack paths to Domain Admins. Solving the ILP on these subgraphs is equivalent to finding defense strategies to eliminate k attack resources. The algorithm proceeds by iterating over all subgraphs of $k = 1$ before advancing to $k = 2, 3, \ldots$. The algorithm runs until all nodes and edges are iterated or the time out is reached. The last k is returned as the maximum number of entry nodes that can be isolated from DA within a particular budget and time constraint. Figure 2 illustrates the workflow of the Spiral algorithm. It is important to note that the Spiral Algorithm is the only algorithm that can solve this NP-Hard Problem [4].

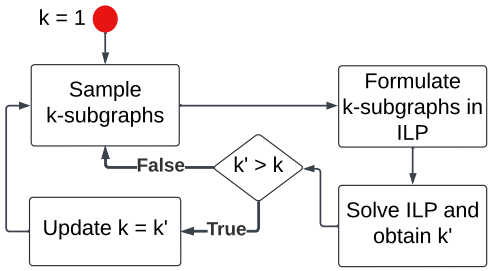


Fig. 2: Steps in the Spiral Algorithm [4].

There is one major limitation with the current algorithms for attack path elimination. They have no abstract representation of security policies (such as access control policies) between an entity and a set of entities. They approximate these policies by breaking them down into multiple entity-to-entity edges. This approximation leads to both the explosion of the number of nodes and edges in the graph but also the extra defensive actions that are oblivious to the inter-dependencies between these approximated edges.

It is common in AD environments for system admins to grant permissions from a privileged principal to a set of objects. For instance, to grant an IT Admin Group full control over a set of users, known as *Organisational Unit (OU)* in ADs, permission is assigned from the Group to the user OU, rather than individually to each user. Through AD's inheritance mechanism, this permission is automatically propagated to all objects within the OU. Consequently, this type of permission assignment inherently operates on a node-to-set mapping principle where permission edges between the principal and the OU are called policy-defining edges. However, in the simple graph model, such security permissions are represented using a node-to-node mapping approach. Specifically, for a set of n elements, the model introduces an edge from the Group node to the set node, and n additional edges from the Group node to each individual element within the set. Thus, the representation of a single permission requires $n + 1$ edges.

Furthermore, in practice, removing the permission between a principal and an OU also revokes the permission between the principal and every element in the OU. In other words, a single action to remove the policy-defining edge can lead to the elimination of multiple individual permissions due to their dependencies on the policy-defining edge.

Our work seeks to improve Spiral and other simple graph algorithms by developing a better graph model that can yield shorter computation time and requires fewer defensive actions. Our model natively captures the polyadic relationships in security policies by using high-order graph models - discussed below.

B. High-Order Attack Graph Models

There are several graph models that could be used to capture security policies that map nodes to sets, including AND/OR attack graphs, metagraphs, directed hypergraphs, and petrinets.

An AND/OR graph captures sets of preconditions and postconditions of a cyber attack. In this graph, preconditions and postconditions are represented as condition nodes, each signifying a specific state or fact about network entities, such as a user X having full control over a computer Y. Exploits or the steps taken by attackers are depicted as exploit nodes. AND/OR attack graphs are formally defined in [14].

A more general model for AND/OR graphs is a directed hypergraph or alternatively metagraph [15]–[18]. Directed hypergaphs are semantically identical to metagraphs. Petrinets further extend metagraphs/hypergraphs to capture polyadic dependencies in cyber security [19], [20], especially to take into account temporal dependencies. We focus on metagraphs in this work. A metagraph extends the AND/OR model into a more general model for mapping between sets and sets. Formally, a metagraph $S = \langle X, E \rangle$ [15] is defined as 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. An edge e is a pair $e = \langle V, W \rangle \in E$ consisting of two sets, an invertex $V \subset X$ and an outvertex $W \subset X$. We call $e \in E$ a *meta-edge*. An example of metagraphs is in Figure 3. In this graph, the relationship between 2 sets $\{x_1, x_2\}$ and $\{x_4, x_5\}$ can be modeled using the edge $e_1 = \langle \{x_1, x_2\}, \{x_4, x_5\} \rangle$. A meta-edge can also connect a node and a set where a

node is considered as a set of size 1. In Figure 3, the meta-edge $e_2 = \langle\{x_3\}, \{x_4, x_5\}\rangle$ connects the node x_3 and the set $\{x_4, x_5\}$.

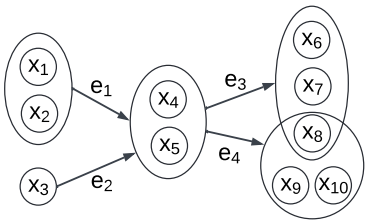


Fig. 3: An example of metagraphs. A meta-edge $e_1 = \langle\{x_1, x_2\}, \{x_4, x_5\}\rangle$ connects two sets $\{x_1, x_2\}$ and $\{x_4, x_5\}$. The graph has 4 edges - representing 4 security dependencies.

A node-to-node approximation of the metagraph in Figure 3 is shown in Figure 4. The node-to-node graph has many more edges than the metagraph as it expands all the mappings between nodes and sets into node-to-node mappings. This model not only fails to capture dependencies between edges but also over-complicates the attack graphs. To address this problem, we propose to model the node-to-set mapping nature of Active Directory using a variant of metagraphs where $\forall e = \langle V, W\rangle$, $|V| = 1$. We denote such edges as $e = \langle v, W\rangle \in E$ where v is a single element in the generating set, while W represents a set of elements. Details of the model are provided in the next section.

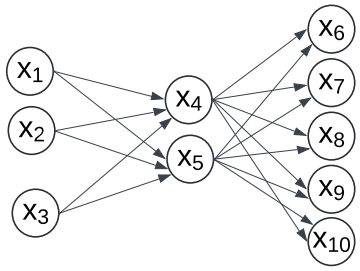


Fig. 4: A node-to-node approximation of the metagraph in Figure 3. This graph has 16 edges, compared to the 4 permission edges in the metagraph model.

III. α-Metagraph-based mapping model for AD systems

In this section, we develop a novel α-Metagraph model that captures the node-to-set mappings prevalent in realistic AD security settings. We also explain how these models can be constructed using data extracted from AD systems through management protocols such as the Lightweight Directory Access Protocol (LDAP).

A. AD Modelling using α-Metagraphs

As mentioned previously, in Active Directory, security permissions are configured on sets of entities. For instance, to grant a principal specific control over a set or organizational unit (OU) of users within the Business Department, system admins configure the permission at the OU level, rather than establishing individual permissions between the principal and every single user. The inheritance mechanism of AD will propagate this permission to all users within the OU and its nested OUs. Using this approach, when there is an urgent need to eliminate the direct connection between the principal and all users in the OU, defenders only need to remove the security permission on the OU level. In α-metagraphs, objects such as users, computers, or individual network resources are represented as nodes. Organizational Units (OUs), which contain objects and nested OUs, can be modeled as sets comprising element nodes and nested sets. Consequently, the permissions between a principal and an OU can be represented as edges connecting a node to a set. Formally,

Definition III.1 (α-*metagraph*)**.** *An α-metagraph $S = \langle X, E\rangle$ is a metagraph with a generating set X and a set of edges E satisfying the following condition. Let $e_1 = \langle u_1, V_1\rangle, e_2 = \langle u_2, V_2\rangle$ be two edges of the α-metagraph and let $e^{\wedge} = V_1 \cap V_2$ be the intersection of the two outvertex sets, then*

$$\forall e_1, e_2 \in E : \left(e^{\wedge} \neq \emptyset \Rightarrow V_1 \subseteq V_2 \vee V_2 \subseteq V_1\right) \qquad (1)$$

α-metagraph nodes. A generating set of elements $X = \{x_1, x_2, \ldots, x_n\}$ represents individual AD objects including users, computers, and groups. Organisational Units (OUs) are modeled as sets or containers $Q \subset X$ containing these element nodes. This reflects the real configuration where OUs can hold individual objects and nested OUs [21]. Note that there is another aggregate structure in AD called *groups* that are used to manage multiple elements at the same time. However, a group in Active Directory is a *virtual* container that does not physically contain any objects. Instead, it stores references to its members in a list property, which may include users, computers, or other groups. As a result, any attack on groups does not grant attackers control over group members or enable traversal to other nodes via the member references. Therefore, we model groups as nodes in α-metagraphs.

α-metagraph edges. Every permission in an AD is represented by an edge $e = \langle v, W\rangle \in E$ consisting of a node invertex $v \in X$ and a node/set outvertex $W \subset X$ where $|W| \geq 1$. It can be interpreted as a permission or control granted to a principal v over an individual AD object W with $|W| = 1$ or a set of objects (contained in OUs) W with $|W| > 1$. These meta-edges $e \in E$ are policy-defining edges, determining the permission of v over every element within the set W. In essence, a meta-edge e encapsulates the permission dependencies between v and all $x \in W$, a relationship that node-to-node graphs are incapable of representing. The removal of e effectively eliminates all permissions between v and the elements in W. For instance, granting an admin full control over an OU and its elements can be achieved by assigning control at the OU level, which automatically propagates permissions to all elements within the OU. Conversely, revoking this control at the OU level (a single action)

simultaneously removes all permissions between the admin and the objects contained in the OU. However, in a node-to-node mapping representation, this revocation would require individually removing the control between the admin and each element within the OU, necessitating multiple actions.

The condition in (1) ensures that the model adheres to the Active Directory implementation of OU sets [22] that an object such as a user or a computer cannot exist simultaneously in multiple OUs unless the OUs are nested.

Figure 5 provides an example of an α-Metagraph, where individual objects are represented using nodes $x_1, x_2, \ldots, x_{10}$, and OUs are denoted using sets such as $\{x_4, x_5\}, \{x_8, x_9, x_{10}\}$. The edge $e_4 = \langle \{x_5\}, \{x_8, x_9, x_{10}\} \rangle$ could be used to represent the security permission between the user x_5 and a set/OU of computers $\{x_8, x_9, x_{10}\}$.

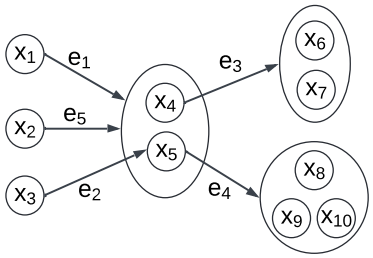


Fig. 5: An example of α-Metagraphs. The meta-edge $e_4 = \langle \{x_5\}, \{x_8, x_9, x_{10}\} \rangle$ connects the node $\{x_5\}$ and the set $\{x_8, x_9, x_{10}\}$.

Note here that in real AD systems, the overlapped sets in the out-vertices such as $\{x_6, x_7, x_8\}$ and $\{x_8, x_9, x_{10}\}$ in Figure 3 are not allowed. We model them with two disjoint sets $\{x_6, x_7\}$ and $\{x_8, x_9, x_{10}\}$ as in Figure 5. We would like to highlight that α-Metagraph is very similar to F-hypergraph [18] with one exception that is represented in Condition (1). That is, in an F-hypergraph, the outvertex sets of any two F-hypergraph edges cannot overlap.

B. Construction of α-metagraphs from data collected by SharpHound

AD security information can be extracted in different ways [12], [23]. These methods differ in the queries they make and the format of their data but provide the same information for the attack graphs. In this work, we present an algorithm to construct α-metagraphs from the data collected by the dominant ingestor SharpHound [12]. Details of data generated by SharpdHound can be found in [12].

We parse the ingestion data from SharpHound into an attack graph using the α-metagraph model by following the process in Algorithm 1. Ingestion data typically includes JSON files [12], each file containing information about a type of AD objects. In our model, we focus on users, computers, groups, and OUs. OUs represent a container or a set in the AD system.

Algorithm 1 starts by parsing JSON files into AD objects of each type, including users, computers, groups, and OUs (Line 1). Next, the Algorithm creates sets of users and computers

Algorithm 1: Construct α-metagraphs from data collected by SharpHound

Input: files - JSON files collected by SharpHound, containing information about an AD system

```
1  Users, Computers, Groups, OUs ← parse_ad_json(files)
2  MG ← create_α_metagraph
3  for ou ∈ OU do
4      set ← MG.generate_set()
5      elements ← get_child_elements(ou)
6      for object ∈ elements do
7          set.add_element(object)
8  for g ∈ Groups do
9      parent_set ← get_parent(g.dn)
10     parent_set.add_element(g)
11     for member ∈ g.members do
12         MG.create_edge(g, member, MemberOf)
13 populate_object_props(Users, Computers, Groups, OUs)
14 ad_permissions ←
       get_permissions(Users, Computers, Groups, OUs)
15 for p ∈ ad_permissions do
16     if p.inherited = False then
17         MG.create_edge(p.principal, p.target, p.type)
18 for comp ∈ Computers do
19     for user ∈ comp.sessions do
20         MG.create_edge(comp, user, HasSession)
21 return MG
```

based on the OU structure (Lines 3-7). After this step, we add groups to their according container/set using their distinguished names (Lines 8-10). A distinguished name is unique for any AD object [24]. It specifies the location of a group in the AD structure [24]. Next, we add members to the groups (Lines 11-12). An edge specifying the group membership is labeled *MemberOf*. Subsequently, we populate the nodes/sets with their properties (Line 13). After building the structure of an AD system, we construct meta-edges representing the security permissions. To do this, we retrieve permissions applied onto all AD objects (Line 14). For every permission between v and w_i (Lines 15-17), the $inherited = True$ means that the permission applied onto the object is inherited from parent containers/sets. This implies that w_i is an element inside a set and the permission is already defined on the meta-edge $e = \langle v, W \rangle$ where $w_i \in W$. Therefore, we do not need to create any extra edge $\langle v, w_i \rangle$. If $inherited = False$, the permission is defined directly onto the object, and we create a meta-edge (Line 17) signifying that $p.principal$ has the permission $p.type$ on the $p.target$. $p.target$ can either be an individual element or a set. In the last step (Lines 18-20), we create edges connecting users and computers, representing user sessions on these computers.

C. Relationships between α-Metagraphs and Digraph Models

In this section, we establish the relationships between the two attack graph models - the traditional digraphs and our proposed α-metagraphs. We first show that there is a polynomial time algorithm to transform an α-metagraph into a digraph which is compatible to run in [1]–[6] without introducing additional attack paths or new security weaknesses. Subse-

quently, in Section IV, we will demonstrate that the node-to-set mapping nature inherent to α-metagraphs, even when represented as digraphs, offers significant advantages over the traditional node-to-node mapping model, enabling the removal of a greater number of attack sources.

1) α-Metagraph to Digraph Transformation: Consider an α-Metagraph with attack paths from a collection of entry nodes S_m to Domain Admins passing through a meta-edge e_γ and a meta-edge connects a node v and a set W as shown in Figure 6. q is a node where attack paths from $w_i \in W$ to DA intersect before heading to DA. For graphs where node q does not exist, we can create an intermediate node q representing an intersection of attack paths from $w_i \in W$ to DA because an intermediate node q does not create any additional attack paths from $w_i \in W$ to DA.

The goal is to transform any such meta-edge $e = \langle v, W \rangle$ into a directed edge of digraphs.

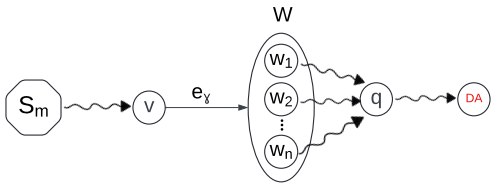


Fig. 6: Attack paths from source to DA via an α-Metagraph edge. Regular arrows represent directed links between 2 nodes or sets. Wavy arrows denote all paths between 2 nodes or sets.

Given any α-Metagraph MG, we can transform it to a digraph using Algorithm 2. Intuitively, the algorithm consists of two steps. In **Step 1**, for any $e_\gamma = \langle v, W \rangle$, we create an intermediate node w_T between v and W, dividing the meta-edge e_γ into 2 segments, as in Figure 7. The security permission represented by the edge e_γ from node v to set W is consequently denoted by the edge e'_γ from node v to node w_T where w_T indicates the node version of the set W. In **Step 2**, we generate directed edges from w_T to every element $w_i \in W$, signifying the ownership of the set W over these element nodes, as in Figure 8.

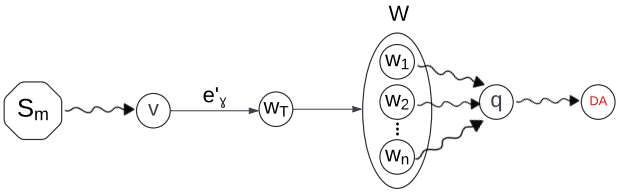


Fig. 7: Attack paths from Source to DA with an intermediate node Q_T between M and Q

In Algorithm 2, we iterate through every meta-edge in the α-Metagraph. If any edge connects two individual nodes (Lines 4-5), we add it to the digraph. Otherwise, for any edge connecting a node and an OU set, we create a node representing the OU (Line 7) and generate edges between

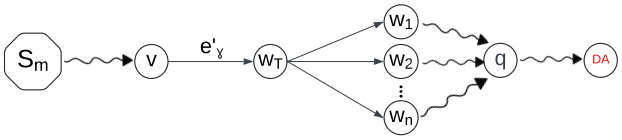


Fig. 8: Attack paths from Source to DA in a digraph. This is the final output of the graph transformation.

that node and every element node in the OU set, signifying the ownership of OU over those elements (Lines 9-10). The security permission from a principal to the OU is added in Line 8.

Algorithm 2: Transform an α-Metagraph into a digraph

Input: MG - an α-Metagraph
1 $meta\_edges \leftarrow MG.meta\_edges$
2 $G \leftarrow create\_digraph()$
3 **for** $me \in meta\_edges$ **do**
4 **if** $|me.outvertex| = 1$ **then**
5 $G.add\_edge(me.invertex, me.outvertex)$
6 **else**
7 $ou\_node \leftarrow G.create\_node(me.outvertex.id)$
8 $G.add\_edge(me.invertex, ou\_node, me.type)$
9 **for** $element \in me.outvertex$ **do**
10 $G.add\_edge(ou\_node, element)$

Figure 9 illustrates an example of transforming an α-metagraph into a digraph for a realistic AD security configuration. Assuming a Group in an AD system has a permission over an OU of computers. In an α-metagraph (Figure 9a), the permission is modeled as a meta-edge $e = \langle v, W \rangle$ where v represents the Group, W denotes the OU set, and element nodes $w_i \in W$ are computers within the OU. To convert this meta-edge to directed edges in a digraph, we apply the graph transformation in Algorithm 2. We first create a node w_T that represents the OU. We then generate an edge connecting the Group to that node w_T, denoting the security permission. To signify the ownership of the OU over computers, we create directed edges $e = \langle w_T, w_i \rangle$ connecting the OU and every computer within it, as demonstrated in Figure 9b. By contrast, in the simple digraph (node-to-node mapping model) that the majority of AD research works rely on [3]–[6], to signify the permission that Group has on the OU and every computer within it, we have to generate directed edges $\langle v, w_i \rangle$ to connect the Group and each computer in the OU, as shown in Figure 10.

2) Attack Path Management on α-metagraphs versus on a Digraph: We show here that the α-metagraph model and its transformed graph model have the same set of attack paths and they admit the same cut solution.

Proposition 1. *The α-metagraph MG and the equivalence digraph G after the transformation in Algorithm 2 have the following properties: (1) No additional attack path from a set of entry nodes S_m to DA is created on $MG \rightarrow G$. (2) An*

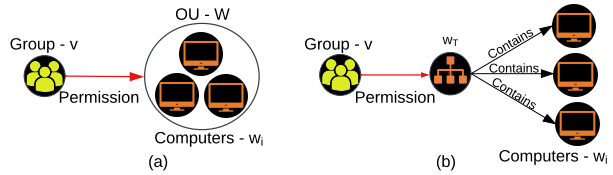


Fig. 9: (a) An AD permission in an α-Metagraph. (b) The permission in a transformed α-Metagraph

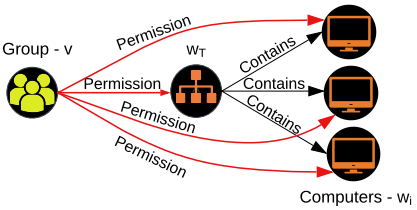


Fig. 10: An AD permission in the node-to-node mapping model

edge cut on e_γ in MG and the equivalent edge cut on e'_γ in G eliminate the same set of entry nodes having attack paths to Domain Admins (DA).

Proof. For the first property, we need to prove that the transformation does not introduce additional attack paths from the set of entry nodes S_m to DA, but rather modifies existing attack paths to convert meta-edges in α-metagraphs to directed edges in digraphs.

Let $P(X, Y)$ denote the set of attack paths from X to Y in an attack graph, and $P(X, Y, \alpha)$ represent those that pass through an edge α within the same graph. The attack paths from S_m to DA in an α-metagraph (Figure 6) before the transformation can be represented as follows:

$$P(S_m, DA, e_\gamma) = P(S_m, DA, \langle v, W\rangle) \quad (2)$$
$$= P(S_m, v) \times \{\langle v, W\rangle\} \times P(W, q) \times P(q, DA) \quad (3)$$
$$= P(S_m, v) \times \{\langle v, W\rangle\} \times \left\{P(w_i, q) \mid {}_{i=1}^{n}\right\} \times P(q, DA) \quad (4)$$

where $P(W, q) = \left\{P(w_i, q) \mid {}_{i=1}^{n}\right\}$.

After Step 1 in the Transformation (Figure 7), $e_\gamma \rightarrow e'_\gamma$, $P(S_m, DA, e_\gamma)$ becomes $P(S_m, DA, e'_\gamma)$. The number of attack paths in $P(S_m, DA, e_\gamma)$ remains unchanged in $P(S_m, DA, e'_\gamma)$ as we only introduce one intermediate node w_T, which converts the $e_\gamma = \langle v, W\rangle$ to $\langle v, w_T\rangle \times \langle w_T, W\rangle$. Therefore, we have:

$$P(S_m, DA, e'_\gamma) = P(S_m, v) \times \{\langle v, w_T\rangle\} \times \{\langle \mathbf{w_T}, \mathbf{W}\rangle\} \times \left\{\mathbf{P}(\mathbf{w_i}, \mathbf{q}) \mid {}_{\mathbf{i=1}}^{\mathbf{n}}\right\} \times P(q, DA) \quad (5)$$

In (5), we expand $\{\langle w_T, W\rangle\} \times \left\{P(w_i, q) \mid {}_{i=1}^{n}\right\}$ as follows:

$$\{\langle w_T, W\rangle\} \times \{P(w_i, q) \mid {}_{i=1}^{n}\} = \{P(w_T, w_i) \times P(w_i, q) \mid {}_{i=1}^{n}\} \quad (6)$$

The $P(w_T, w_i)$ in (6) represents attack paths from w_T to w_i. However, after Step 2 of the Transformation process, $P(w_T, w_i)$ is represented as a directed edge connecting w_T and $w_i \in W$ (Figure 8). $P(S_m, DA, e'_\gamma)$ becomes $P'(S_m, DA, e'_\gamma)$ as follows:

$$P'(S_m, DA, e'_\gamma) = P(S_m, v) \times \{\langle v, w_T\rangle\} \times \left\{\langle \mathbf{w_T}, \mathbf{w_i}\rangle \times \mathbf{P}(\mathbf{w_i}, \mathbf{q}) \mid {}_{\mathbf{i=1}}^{\mathbf{n}}\right\} \times P(q, DA) \quad (7)$$

As in (7), the set W is no longer required in the representation of attack paths from S_m to DA via e'_γ, it is therefore removed from MG. In addition, the Transformation process does not create any additional attack paths, but rather modifies existing ones to convert meta-edges in α-metagraphs to directed edges in digraphs. As a result, the number of attack paths passing through every edge e_γ remains invariant post-transformation.

For the second property, supposing blocking an edge $e_\gamma = \langle v, W\rangle$ in MG separates a set of entry nodes $S_B = \{s_1, s_2, \ldots, s_b\} \subset S_m$ from DA, meaning that every node $s_i \in S_B$ has to go through v before reaching DA. To prove that blocking the equivalent edge $e'_\gamma = \langle v, w_T\rangle$ in G also eliminates the same set of S_B as in G, we use contradiction. Assuming that blocking $e'_\gamma = \langle v, w_T\rangle$ in G only separates a subset $S'_B \subset S_B$ from DA, meaning that some nodes in S_B must have other paths to attack DA, in addition to paths passing through v. This is contradictory to the observation discussed in the first property that the transformation does not create additional attack paths. Therefore, we can conclude that an edge cut in G eliminates the same set of attack sources as in MG.

□

D. Integration of α-metagraphs to BloodHound

BloodHound employs SharpHound to collect Active Directory (AD) data from an organization, which it then uses to construct graphs based on a node-to-node mapping model. In contrast, our approach leverages AD data to construct an α-metagraph using Algorithm 1. Since BloodHound supports only directed graphs, we apply Algorithm 2 to transform the α-metagraph into a digraph prior to importing it into BloodHound.

IV. Attack Path Management with α-Metagraphs

Once an α-metagraph is built for an AD system, we can leverage the powerful abstraction of its node-to-set mapping nature to develop efficient attack path management solutions for the targetted AD system. We develop one such solution in this work - the α-Spiral algorithm.

A. α-Spiral Algorithm

In an unsecured Active Directory system, a set of low-privilege nodes (non-administrative entities) have attack paths to Domain Admins (DA) due to misconfigurations and bad cyber hygiene accumulated over some time. These entry nodes are typically user accounts, computers or Internet-facing services. To protect AD systems from snowball attacks, it is essential to protect DA from these nodes by removing edges that help eliminate attack paths from these entry nodes to

DA. Following [4], we use ILP formulation to optimize the number of edges to cut in α-metagraphs. Our key innovation is leveraging policy-defining edges that simultaneously grant accesses to multiple entities. We call our algorithm the α-Spiral algorithm.

α-Spiral Algorithm Our proposed algorithm improves *the sampling of k-subgraphs* of the original Spiral Algorithm in [4]. At each layer $k = 1, 2, \ldots$, during the subgraph sampling, each iteration aims to uncover new edges by sampling a k-subgraph α-T_k that contains undiscovered edges. The α-T_k subgraphs are composed of the shortest paths between k entry nodes in S and the target node t. This is achieved by the procedure $T'(MG, S, t, k)$ which executes Dijkstra's algorithm to construct α-T_k. Before executing Dijkstra's Algorithm, we retrieve all meta-edges $e \in E$ in a given attack graph and assign them with the weight of negative infinity. This ensures that attack paths containing meta-edges are significantly more likely to be selected. The reason is that these edges can expedite the elimination of attack sources. In particular, as outlined in Section III-A, meta-edges (or policy-defining edges) represent security permissions between a principal and an OU. These edges define the permission of the principal over all elements within the OU. Removing such edges simultaneously revokes all permissions granted to the principal for every element in the OU, achieving a more extensive elimination of permissions compared to removing a single permission between the principal and an individual element within the OU. In essence, the removal of policy-defining edges can effectively eliminate multiple attack paths in a single action, thereby significantly suppressing potential attack sources. In other words, it accelerates the increase of k, a.k.a the elimination of k attack sources in urgent cyberattack situations.

Second, in the original Spiral algorithm [4], if Dijkstra's algorithm only samples already explored edges, the process transitions to randomly selecting z edges from the remaining unexplored edges. In contrast, the α-Spiral algorithm begins by retrieving a pool of unexplored edges, where initially, each edge has an equal probability of being selected. As we iterate through this pool, for every meta-edge encountered, its probability of selection is scaled by a multiplier $\beta > 1$. This adjustment prioritizes meta-edges, increasing their likelihood of being included in the defense strategy. Once an edge is explored, its weight is incremented by 1, enabling the selection of new edges in subsequent iterations. If the edge is a meta-edge with a value of negative infinity, its weight will be reset to 1. The pseudocode for the k-subgraph sampling in α-Spiral algorithm is given in Algorithm 3. The workflow of the improvement is illustrated in Figure 11.

Algorithm 3: α-Spiral Algorithm

Input: G_{MT} - a transformed α-metagraph
S - a set of attack sources
t - the target node (Domain Admins)
k - the layer

Function *α_edge_sampling(k)*
  for $e \in E$ **do**
    if e *is meta-edge* **then** $w(e) \leftarrow -\infty$
    else $w(e) \leftarrow 0$
  unexplored $\leftarrow \{\}$
  if $\exists e \in E : w(e) \leq 0$ **then**
    α-$T_k \leftarrow T'(G_{MT}, S, t, k)$
    for $e \in \alpha$-T_k **do**
      if $w(e) \leq 0$ **then**
        unexplored.add(e)
    if *unexplored* $= \emptyset$ **then**
      sample_pool $\leftarrow E - E^*$ `/* E* - explored edges so far */`
      probs $\leftarrow \{1$ **for** *edge* **in** *sample_pool*$\}$
      for $e \in$ *sample_pool* **do**
        if $w(e) = -\infty$ **then**
          probs[e] $\leftarrow \beta$
      unexplored $\leftarrow$ weighted_sampling($E - E^*$, probs, z)
    for $e \in$ *unexplored* **do**
      if $w(e) \geq 0$ **then** $w(e) \leftarrow w(e) + 1$
      else $w(e) \leftarrow 1$
  return *unexplored*

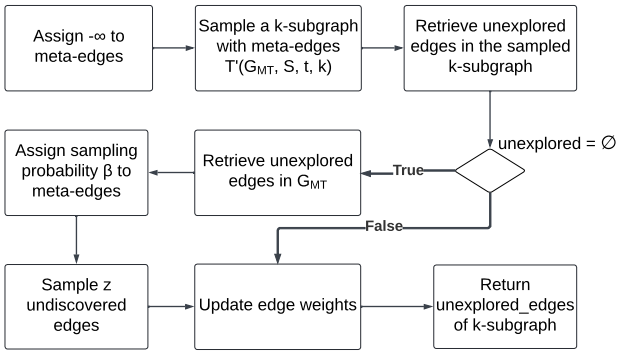


Fig. 11: Block diagram of the k-Subgraph Sampling component in the α-Spiral Algorithm

B. Properties of α-Spiral

We show in this section that α-Spiral algorithm that leverages the polyadic structures (node-to-set mapping) in α-metagraph provides better defensive solutions for AD systems than their simple graph counterparts (node-to-node mapping).

We first formally define the graph construction process for simple graphs which reflects the node-to-node mapping model. In an AD system, there are 2 representations that we have discussed so far: *node-to-set* mapping model and *node-to-node* mapping model. The α-metagraph representing the node-to-set mapping can be transformed into a digraph without creating additional attack paths or new security weaknesses, as shown in Section III-C. To generate a node-to-node graph for the same AD system, we present Algorithm 4. Given a transformed α-metagraph G_{MT}, the Algorithm 4 returns a simple graph G_{NN} where for every edge connecting a node and an OU/set, we create directed edges connecting the node and every element within the set, resulting in the node-to-node mapping model. Figure 12 illustrates a subgraph of the returned graph G_{NN} in Algorithm 4. In the context of Active

Directory, if a user is granted security permissions over an OU, equivalent permissions are correspondingly established for all objects in the OU. In this algorithm, G_{MT} and G_{NN} have the Proposition 2.

Algorithm 4: Construct the node-to-node mapping graph from an α-metagraph post-transformation

Input: G_{MT} - an α-metagraph post-transformation

1 $G_{NN} \leftarrow G_{MT}$
2 **for** $e \in G_{NN}.edges$ **do**
3 **if** $e.outvertex.type$ *is* OU **then**
  /* Retrieve all nodes in the OU (set) */
4 $set\_elements \leftarrow$ $get\_set\_elements(G_{NN}, e.outvertex)$
5 **for** $node \in set\_elements$ **do**
6 $G_{NN}.add\_edge(e.invertex, node)$

Proposition 2. *Let p_a and p_b respectively represent the numbers of attack paths from a set of entry nodes S_m to Domain Admins (DA) in G_{MT} and G_{NN}. Then:*

$$p_a < p_b.$$

Proof. In the α-metagraph G_{MT} as illustrated in Figure 8, attack paths passing through a node v and $w_i \in W$ via an edge $e'_\gamma = \langle v, w_T \rangle$ is defined as $P'(S_m, DA, e'_\gamma)$ in (7). In G_{NN}, directed edges connecting nodes and elements in sets are created (Lines 2-6 in Algorithm 4), which means v and elements w_i in the set represented by w_T are connected via an edge $\langle v, w_i \rangle$ as illustrated in Figure 12. As a result, additional attack paths from S_m to DA via these edges $\langle v, w_i \rangle$ are formed and are defined as $P'(S_m, DA, \langle v, w_i \rangle)$. This proves that G_{NN} has more attack paths from S_m to DA than G_{MT}, or $p_a < p_b$. □

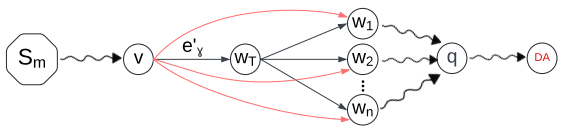


Fig. 12: The *node-to-node* graph G_{NN} in Algorithm 4. Added edges are colored orange.

Lemma 1 formalizes the advantages of α-Spiral over the Spiral solution in [4].

Lemma 1. Let G_{MT} represent the α-metagraph model of an AD attack graph after the Transformation process in Section III-C1, and G_{NN} denote the node-to-node mapping graph of the same AD system, obtained using Algorithm 4. Given a finite budget and time constraints, let A and B be the numbers of remaining entry nodes with attack paths to the Domain Admin (DA) after applying the α-Spiral Algorithm on G_{MT} and the Spiral Algorithm on G_{NN}, respectively. Then:

$$A \leq B.$$

Lemma 1 can be interpreted as follows. Given an Active Directory system with 2 representations G_{MT} and G_{NN}, applying the α-Spiral Algorithm on the transformed α-metagraph G_{MT} eliminates more attack sources compared to applying the Spiral Algorithm on the simple graph G_{NN}, thereby resulting in an AD system with *fewer* remaining attack sources ($A \leq B$). The reason why α-Spiral and Spiral Algorithms might not eliminate all attack paths from all entry nodes to DA can be explained as follows. In Active Directory (AD) mitigation, defenders face constraints due to the high cost of edge removals, which require manual reviews to prevent operational disruptions and cascade effects, often taking days or weeks to complete [4]. With limited resources, it may not be feasible to eliminate all attack sources. Additionally, in urgent situations, defenders have limited time to devise strategies to maximize the elimination of entry nodes from DA. This problem is NP-Hard [4] with exponential running time, making it challenging to achieve optimal solutions for realistic AD graphs which are typically large graphs potentially containing hundreds of nodes and millions of edges.

The proof for Lemma 1 leverages the ILP formulation in [4]. In an AD graph with a set S_m of entry nodes having attack paths to Domain Admins (DA), let $n(u)$ and $cut(\langle u, z \rangle)$ be binary variables for nodes and edges in the AD attack graph. t represents the Domain Admins node. If a node u cannot reach t, its variable $n(u)$ becomes 0, otherwise, $n(u) = 1$. As the node DA can always reach itself, then $n(t) = 1$. If an edge $\langle u, z \rangle$ is cut, its corresponding variable $cut(\langle u, z \rangle)$ becomes 1, $cut(\langle u, z \rangle) = 0$ otherwise. In the original ILP formulation [4], the objective function aims to maximize the number of entry nodes that do not have attack paths to DA. This is equivalent to minimizing the number of entry nodes that can traverse to DA $\sum_{s \in S_m} n(s)$, where $n(s)$ with $s \in S_m$ is a binary variable, indicating whether an entry node s in S_m can reach DA. b represents the budget constraint, specifying the maximum number of edges to cut. We present the leveraged ILP formulation as follows.

$$\text{minimize} \quad \sum_{s \in S_m} n(s) \tag{8}$$

$$\begin{aligned} \text{subject to} \quad & n(u) \geq n(z) - cut(\langle u, z \rangle), \quad \forall \langle u, z \rangle \in E \\ & n(t) = 1 \\ & \sum_{\langle u, v \rangle \in E} cut(\langle u, v \rangle) \leq b, \end{aligned} \tag{9}$$

As shown in Algorithm 4, G_{NN} retains the same nodes and edges as G_{MT} (Line 1). Therefore, the ILP formulation for the Spiral Algorithm on G_{NN} can be applied to the α-Spiral Algorithm on G_{MT}. However, the difference is that the ILP formulation for G_{NN} in the Spiral Algorithm requires additional constraints to account for the extra edges in G_{NN} (Lines 5-6), reflecting the node-to-node mapping model. These constraints are described as follows.

Let $E^{\sim}$ indicate a set of transformed meta-edges in G_{MT} where $\forall e = \langle v, w_T \rangle \in E^{\sim}$, node w_T represents a set. G_{NN} also possesses $E^{\sim}$ (refer to Line 1, Algorithm 4).

In G_{NN}, $\forall w_i$ that belongs to the set represented by w_T, we have the following constraint for extra edges $\langle v, w_i \rangle$ as illustrated in Figure 12 using Equation (9):

$$n(v) \geq n(w_i) - \text{cut}(\langle v, w_i \rangle) \tag{10}$$

where $\langle v, w_i \rangle$ represents the edge between v and an element w_i in the set denoted by w_T.

The proof for Lemma 1 consists of 3 steps as outlined below. Intuitively, the Spiral Algorithm on G_{NN} and α-Spiral on G_{MT} both share the same part of ILP formulation, as discussed above. However, as G_{NN} has additional edges $\langle v, w_i \rangle$, the ILP formulation in the Spiral Algorithm will include an additional constraint for every entry node $s \in S_m$, involving the added edges $\langle v, w_i \rangle$. Therefore, ILP formulation of the α-Spiral Algorithm has a larger feasible region due to fewer constraints, allowing more options for optimization. Adding constraints in the ILP formulation of the Spiral Algorithm shrinks this feasible region, potentially excluding the optimal solution of the α-Spiral ILP. Thus, the α-Spiral ILP can always achieve equal or better optimization compared to Spiral ILP.

Proof. **Step 1.** In the transformed α-metagraph G_{MT}, given an attack path P' of length greater than 0 from an entry node x to a target node t, the dependency between node x and all other nodes and edges on the path using the constraint (9) is shown as follows:

$$n(x) \geq n(t) - \sum_{\langle u,z \rangle \in P_{x \to t}} cut(\langle u, z \rangle) \tag{11}$$

Due to space limits, the proof for (11) can be found in the Appendix B.

Step 2. For every additional edge $\langle v, w_i \rangle$ in G_{NN}, we have at least one more constraint for every entry node $s \in S_m$ that has attack path(s) to v, using Equations (10) and (11):

$$\begin{aligned} n(s) &\geq n(v) - \sum_{\langle u,z \rangle \in P_{s \to v}} \text{cut}(\langle u, z \rangle) \\ &\geq [n(w_i) - \text{cut}(\langle v, w_i \rangle)] - \sum_{\langle u,z \rangle \in P_{s \to v}} \text{cut}(\langle u, z \rangle) \\ &= n(w_i) - \sum_{\langle u,z \rangle \in P_{s \to w_i}} \text{cut}(\langle u, z \rangle) \end{aligned}$$

where $P_{s \to v}$ and $P_{s \to w_i}$ are two sets of edges along two attack paths from an entry node s to an element v and w_i in the OU set represented by w_T respectively.

Step 3. Feasiblity regions. Let C_1 be the set of constraints for a node-to-set mapping graph G_{MT}, C_2 be the set of constraints for the equivalent node-to-node mapping graph G_{NN}. As demonstrated in Step 2, the ILP formulation for G_{NN} includes additional constraints. Hence, $C_1 \subseteq C_2$.

Let $F1$ be the feasible region defined by the set of constraints C_1, $F2$ be the feasible region defined by C_2. Since C_2 includes all the constraints in C_1 and additional constraints as specified in Step 2, $F_2 \subseteq F_1$. This means that every feasible solution for C_2 is also a feasible solution for C_1, but the reverse is not necessarily true.

Optimal solution comparison. Let a^* be the optimal solution for constraints C_1 in F_1, $A = ILP(a^*)$ represents the optimal value of the ILP objective function (8) for C_1. Let b^* be the optimal solution for constraints C_2 in F_2, $B = ILP(b^*)$ represent the optimal value of the ILP problem for C_2.

To prove $A \leq B$, we use contradiction. Assuming that $A > B$. From the above observation of Feasibility regions, a solution for C_2, b^*, is also a feasible solution for C_1. Based on the inductive assumption $A > B$, we can imply that b^* minimizes greater numbers of entry nodes having attack paths to DA, compared to the solution a^*. In other words, b^* yields a better solution for constraints C_1. This contradicts the fact that a^* is the optimal solution for constraints C_1.

Therefore, we can conclude that $A \leq B$ and that using the node-to-set mapping model G_{MT} on ADs, we can achieve equal or lower numbers of remaining attack sources with attack paths to DA, compared to the node-to-node mapping model G_{NN} for the same Active Directory configurations. □

C. Application in real world AD installations

Our method models Active Directory security configurations as an α-metagraph, where node-to-set mapping reduces the number of nodes and edges compared to traditional node-to-node graphs. This abstraction leads to significantly fewer variables in the ILP formulation, enabling faster solving and making large-scale privilege escalation analysis tractable. Combined with the α-Spiral algorithm—which acts as an anytime algorithm capable of quickly producing acceptable, though potentially suboptimal, solutions—this approach supports practical deployment in enterprise AD environments by achieving broader attack surface reduction with lower cost and effort.

The proposed method naturally accommodates dynamic changes in Active Directory. Since the α-metagraph is constructed directly from the current AD state, any configuration update automatically leads to a reconstructed graph that reflects the new relationships and permissions. Thanks to the reduced graph size and lower ILP complexity, re-running the α-Spiral algorithm over the updated graph remains computationally feasible, even in large-scale environments. This allows defenders to recompute effective minimal edge removals in response to evolving AD configurations, supporting near real-time defense without excessive computational overhead, as demonstrated in our experiments.

D. Advantages and compatibility of the α-metagraph model compared to the node-to-set mapping model in digraphs

While it is true that the α-metagraph can be converted back to a digraph without losing attack-relevant information, the key advantage of the α-metagraph lies in its more compact and expressive representation of complex AD structures—such as nested groups and permission sets—that are cumbersome or highly redundant in traditional digraphs. This compactness significantly reduces the security problem size and computational complexity when applying to anytime security algorithms, which translates into faster and more scalable analysis. The conversion capability ensures compatibility and interoperability with existing algorithms and tools that operate on

the traditional node-to-node mapping model of digraphs [3]–[8], and provides a migration path toward improved models without discarding prior work.

V. Experiments

In this section, we present the evaluation of our α-metagraph models and α-Sprial algorithms on Active Directory attack graphs from ADSynth [25] - a recent AD attack graph generator. Due to the confidential nature of real-world Active Directory (AD) data, publicly available datasets are scarce. We obtained one real AD dataset consisting of approximately 100,000 nodes and 1.2 million relationships, reflecting the scale and structural complexity of a large enterprise environment. However, applying the α-Spiral Algorithm to this dataset resulted in zero attack sources eliminated. While the algorithm successfully executed on this large-scale graph, the dataset itself is heavily secured and contains no identifiable attack sources, making it unsuitable for assessing the effectiveness of attack source elimination.

To thoroughly evaluate our approach under various realistic and complex AD conditions, we employed ADSynth, a synthetic AD data generator specifically designed to mimic realistic AD environments while including representative security vulnerabilities. ADSynth produces AD graphs with features such as deeply nested group memberships, delegated administrative privileges, multiple privileged groups, diverse distributions of users and computers, along with a range of other parameters (refer to ADSynth [25]) that capture the heterogeneity and historical complexities typical of enterprise AD deployments. These comprehensive configurations ensure that the synthetic datasets reflect not only the size but also the structural characteristics and security challenges of real-world AD environments. In our experiments, the ADSynth data include various sizes: 3k, 6k, 12k, and 30k nodes. ADSynth graphs have several levels of security settings. In our experiments, the security levels of the AD systems range from vulnerable configurations, where over 75% of regular users have attack paths to Domain Admins, to secure configurations, where fewer than 0.5% of such users exist. To account for variability in AD configurations, each graph size includes 10 instances and we report the average performance over these 10 runs. We execute the algorithm under different budget constraints, specifically 5, 10, 15, and 20. The entry nodes in our experiments are regular users and low-privileged computers in AD systems.

We focus in particular on the comparative performances of the node-to-set mapping model of α-metagraphs (N-to-S) using the α-Spiral algorithm over the existing node-to-node mapping model (N-to-N) with the Spiral algorithm. We use following metrics in our evaluation:

- The number of entry nodes or attack sources to be eliminated, representing the effectiveness of defense strategies in eliminating risks.
- Execution time of the algorithms, representing the effectiveness of the mitigation strategies under realistic time constraints.

ADSynth data is in JSON Lines format [26], where each line contains information about an AD object or describes a relationship between two entities. We apply Algorithm 5 (due to space limits, please refer to the Appendix) to parse this file into an α-Metagraph, which reflects the node-to-set mapping model. To produce the equivalent node-to-node mapping digraph for comparison, we use the Algorithm 4.

To evaluate the performance of the algorithm under different time constraints, we study two distinct security scenarios:

- Long-term mitigation: We allow the algorithms to run for a maximum of 36 hours. This resembles periodic system checks where AD defenders have an extended duration to analyze and remove vulnerabilities of an AD system [27], [28].
- Rapid incident response: We evaluate the results after one hour to assess the effectiveness of the graph models in addressing urgent security incident responses.

A. Long-term Mitigation

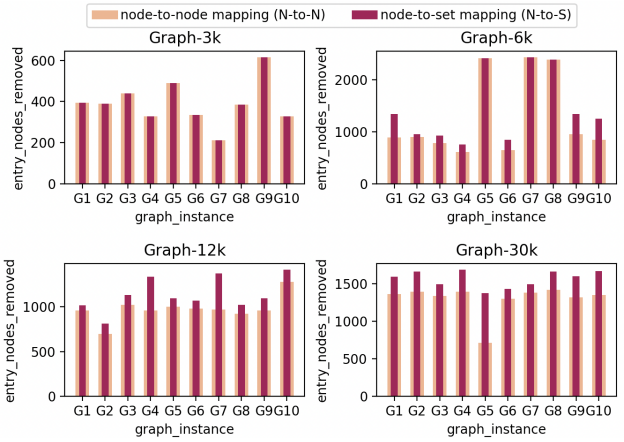


Fig. 13: Long-term mitigation in Vulnerable AD systems with budget of 20.

1) Effectiveness of defense strategies: Figures 14 and 13 show the number of entry nodes eliminated under vulnerable and secure settings respectively. As shown, for all AD settings, the number of attack sources to be eliminated in the node-to-set α-Metagraph model is greater than or equal to that of the node-to-node mapping model. Recall that both α-Sprial and Spiral require a budget b, which is the number of edges to be cut. The figures emphasize that under different budgets, α-Spiral algorithm applied on the node-to-set mapping produces defense strategies that mitigate more attack sources compared to the Spiral algorithm applied on the node-to-node graphs. In vulnerable settings, the observations remain consistent across different budgets, such as the budget of 20 (Figure 13). However, for secure graphs, we observe distinct patterns in the behavior of the security algorithms under varying budgets and graph sizes. Specifically, for small secure graph sizes (3k and 6k nodes), as shown in Figure 14, the number of attack sources mitigated in both mapping models tends to be similar, starting

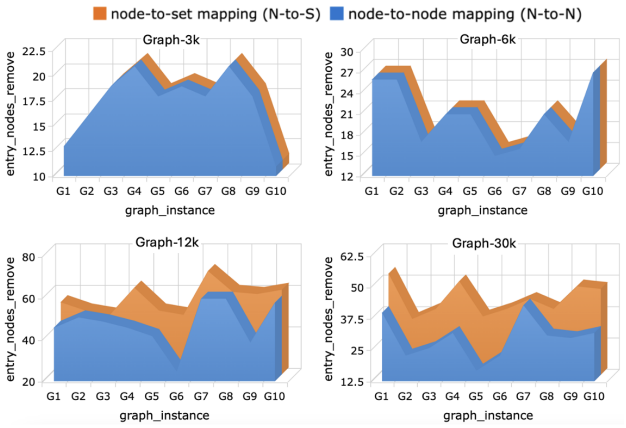


Fig. 14: Long-term mitigation in Secure AD systems with budget of 10.

from a budget of 10. This similarity results from the lower number of attack paths in these graphs. Although the N-to-N mapping generates additional attack paths (Proposition 2), they remain within a proportion that can be managed by the available budgets. Consequently, even though the number of attack paths in this N-to-N model exceeds that of the N-to-S mapping model, the effectiveness of the defense strategies applied to them is still comparable. However, for secure graphs of larger sizes (12k and 30k nodes), a similar phenomenon only emerges when the budget is increased to 15 and 20 (Due to space limits, refer to Figure 18 in the Appendix). The reason is that the larger graph sizes lead to an increase in the number of attack paths. For the N-to-N model, it also has additional attack paths. The lower budget of 10 is therefore insufficient to enable the N-to-N model to eliminate the same number of attack sources as the N-to-S model (Figure 14). Only higher budgets can eliminate these additional attack paths in the N-to-N model.

Furthermore, we observe the *considerable variability* in the number of entry nodes removed across graph instances. For instance, in Figure 13, for the graph size of 6k nodes, the number of attack sources mitigated in G1 of N-to-S is 1250, but in G5, it could be as high as roughly 2250, or only around 800 in G6. This phenomenon can be explained as follows. By analyzing the graph generator, we find that it introduces a significant degree of randomness such as the number of computers a user can access, the group memberships of users, the misconfigurations allowing regular users to modify the passwords of privileged computers, and other factors. For example, if a regular user at Tier 3 is erroneously granted full control over a device in Tier 1, they can escalate the privileges to move from Tier 3 to Tier 1. Due to the randomness of graph generation, attack paths from this device to Domain Admins (DA) may or may not exist, directly influencing the user's potential to carry out an attack against DA. As a result, such random attackability accumulated determines that, for the same AD configuration and a specific budget, some graphs may contain a large number of removable entry nodes, while others may have far fewer such nodes.

2) Execution Time - Computational Complexity: We show in Figure 15 the execution times for the α-Spiral algorithm on the N-to-S model and Spiral Algorithm on the N-to-N model across all AD sizes. The results demonstrate that α-Spiral on the N-to-S model requires consistently lower computation time than Spiral on the N-to-N model across all graph sizes. Especially, α-Spiral (N-to-S) is several orders of magnitude faster in vulnerable AD settings. In general, the running time of an ILP solution is directly proportional to the number of decision variables that need to be optimized. Since each edge in a graph corresponds to a variable in the ILP formulation, the complexity of the problem increases with the number of edges. The large numbers of edges in the N-to-N mapping model translate directly to large numbers of decision variables, thereby leading to longer computational times compared to the N-to-S model. Table I shows the number of variables in the ILP for Spiral on the N-to-N model and α-Spiral on the N-to-S model. The standard deviations for 10 graph instances per graph size are included following the $\pm$ sign.

Security	Graph size	Vars in N-N	Vars in N-S
Vulnerable	$3k$	24168 ± 45	13879 ± 57
	$6k$	49562 ± 139	23580 ± 89
	$12k$	99126 ± 1945	47258 ± 236
	$30k$	236676 ± 5784	105679 ± 534
Secure	$3k$	23569 ± 47	13165 ± 48
	$6k$	47840 ± 121	22647 ± 97
	$12k$	94128 ± 1175	46179 ± 261
	$30k$	215764 ± 2109	103249 ± 517

TABLE I: Numbers of variables in ILP formulation of Spiral and α-Spiral Algorithms

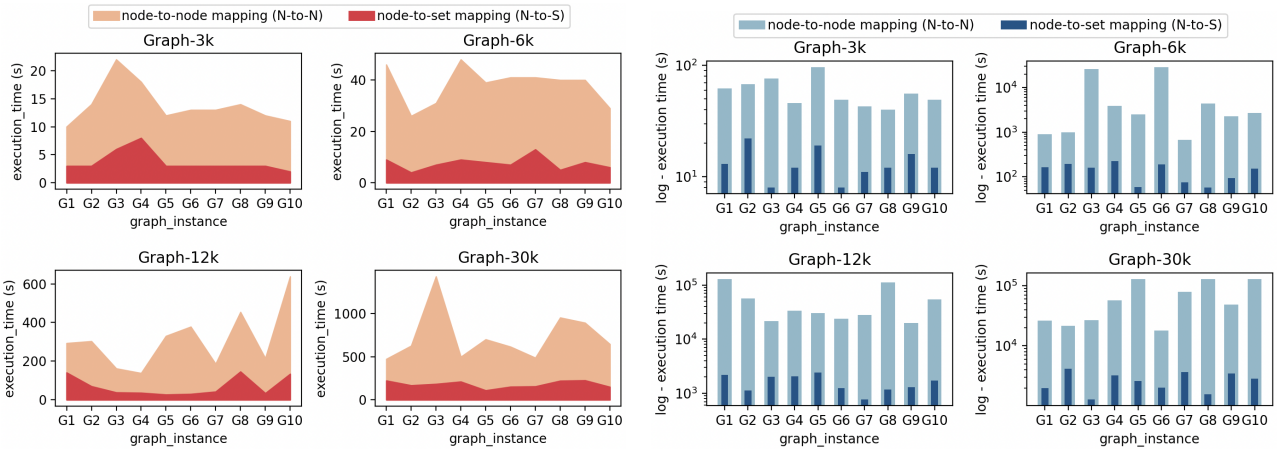
An example of a graph instance with a significant disparity in the running time between two algorithms on two respective models is the vulnerable graph G5 of size 30k (Figure 15b). The N-to-S model requires approximately 21 minutes (2617s) to reach the optimal solution, whereas the N-to-N mapping model takes nearly 36 hours (129600s) to achieve a suboptimal outcome, eliminating only 710 attack sources in contrast to 1373 attack nodes in the N-to-S model (Figure 13).

Note that the long period of 36 hours allows both algorithms to achieve optimality for defense strategies within a wide range of budgets considered in this section.

B. Rapid Mitigation In Incident Responses

To test the algorithms under scenarios where quick decisions are needed, we set a time limit of 1 hour for both Spiral and α-Spiral. We only focus on vulnerable settings as under secure settings, all graph instances of both models can reach an optimal solution within an hour.

Figure 16 illustrates that after one hour of execution, the number of attack sources removed in the N-to-S model is consistently greater than or equal to that of the N-to-N model. For the small graph of size 3k, the results between 2 models are similar. For other graph sizes, the greater number of attack



(a) Secure AD system

(b) Vulnerable AD system

Fig. 15: Execution time in Secure and Vulnerable AD systems with budget = 20, capped at 36 hours

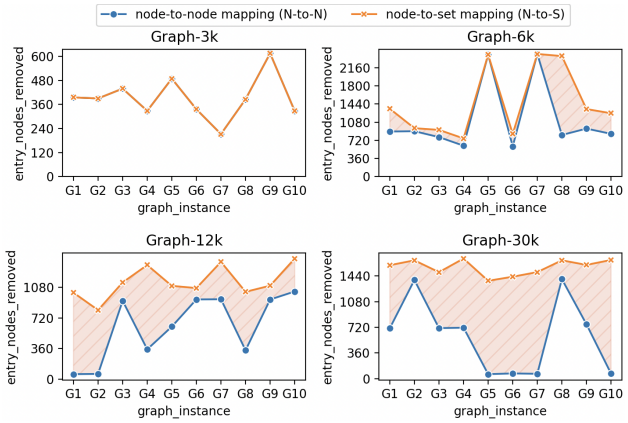


Fig. 16: Rapid mitigation in Vulnerable AD systems: Effectiveness of defense strategies under budget of 20

sources achieved in the N-to-S model can be explained based on the inner working of the two algorithms, as shown in Section II-A. At each layer k, the number of subgraphs in the N-to-S model is smaller than in the N-to-N mapping model due to the lower quantity of edges in the N-to-S model. Furthermore, the α-Spiral Algorithm prioritizes meta-edges in its edge-cut strategies, which aims to achieve a larger suppression of attack sources. As a result, the progression through the subgraphs for each k in the α-Metagraph is more rapid. Conversely, the N-to-N model contains a larger number of edges, which in turn leads to a greater number of subgraphs for each k of attack sources. This increased complexity causes slower progression through the values of k. Consequently, after one hour of execution, the N-to-S model typically reaches a higher value of k than the N-to-N model, thereby isolating a greater number of attack sources within the same time frame. We also observe significant variability across different graph instances. This variability is consistent with the explanation for the variability observed in the long-term mitigation. This result highlights the efficiency of the α-Spiral Algorithm on the N-to-S mapping model in removing attack sources under time constraints.

VI. Conclusion

Current commercial and academic works to protect AD systems only use the node-to-node mapping model, overlooking permission dependencies applied on OU and its elements, which are inherent in AD security configuration. To address these limitations, we propose a new abstraction for AD attack graphs, α-Metagraphs, that align with how realistic AD systems are configured. To leverage AD security algorithms in digraphs, we demonstrate a graph transformation to convert α-Metagraphs to digraphs and prove the preservation of attackability information, thereby taking advantage of current novel works. Furthermore, we introduce the α-Spiral Algorithm that leverages α-metagraphs in eliminating greater numbers of attack sources, compared to the Spiral Algorithm on the node-to-node mapping model. Our experiments demonstrate that the α-Spiral Algorithm on the α-metagraph model offers more effective and prompt defense strategies in comparison to the original Spiral Algorithm on node-to-node graphs, which results in a higher number of attack sources mitigated under time and budget constraints.

References

[1] 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.

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

[3] 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*. New York, NY, United States: Association for Computing Machinery, 2023, pp. 993–1003.

[4] Y. Zhang, M. Ward, and H. Nguyen, "Practical anytime algorithms for judicious partitioning of active directory attack graphs," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, K. Larson, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2024, pp. 7074–7081, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2024/782

[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] 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. Washington, DC, USA: AAAI Press, 2022, pp. 9360–9367.

[7] SpectorOps. What is attack path management? [Online]. Available: https://specterops.io/what-is-attack-path-management/

[8] ——. (2023) Bloodhound enterprise: Securing active directory using graphs. [Online]. Available: https://posts.specterops.io/bloodhound-enterprise-securing-active-directory-using-graph-theory-1d0730e4c741

[9] A. Robbins, "The attack path management manifesto," https://posts.specterops.io/the-attack-path-management-manifesto-3a3b117f5e5, accessed: 2022-12-14.

[10] W. S. Andy Robbins, Rohan Vazarkar, "Sharphound," https://github.com/SpecterOps/SharpHound, 2024, accessed: 2024-08-14.

[11] 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)*. Piscataway, NJ, USA: IEEE, 2018, pp. 4474–4479.

[12] A. R. W.S and V. Rohan. (2020, april) Sharphound. [Online]. Available: https://bloodhound.readthedocs.io/en/latest/data-collection/sharphound.html

[13] W. S. Andy Robbins, Rohan Vazarkar, https://specterops.io/bloodhound-overview/, 2024, accessed: 2024-12-14.

[14] L. Wang, A. Liu, and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Computer communications*, vol. 29, no. 15, pp. 2917–2933, 2006.

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

[16] B. Wang, Y. Sun, T. Q. Duong, L. D. Nguyen, and N. Zhao, "Security enhanced content sharing in social iot: A directed hypergraph-based learning scheme," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4412–4425, 2020.

[17] D. Ranathunga, M. Roughan, and H. Nguyen, "Verifiable policy-defined networking using metagraphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 482–494, 2020.

[18] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, "Directed hypergraphs and applications," *Discrete applied mathematics*, vol. 42, no. 2-3, pp. 177–201, 1993.

[19] M. H. Henry, R. M. Layer, K. Z. Snow, and D. R. Zaret, "Evaluating the risk of cyber attacks on scada systems via petri net analysis with application to hazardous liquid loading operations," in *2009 IEEE Conference on Technologies for Homeland Security*. IEEE, 2009, pp. 607–614.

[20] M. D. Petty, T. S. Whitaker, E. M. Bearss, J. A. Bland, W. A. Cantrell, C. D. Colvett, and K. P. Maxwell, "Modeling cyberattacks with extended petri nets," in *Proceedings of the 2022 ACM Southeast Conference*, 2022, pp. 67–73.

[21] Microsoft. (2024) Create an organizational unit (ou) in a microsoft entra domain services managed domain. Microsoft. [Online]. Available: https://learn.microsoft.com/en-us/entra/identity/domain-services/create-ou

[22] B. Sheresh and D. Sheresh, *Understanding directory services*. Sams Publishing, 2002.

[23] A. R. W.S and V. Rohan. Powerview. [Online]. Available: https://powersploit.readthedocs.io/en/latest/Recon/

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

[25] N. L. Nguyen, N. Falkner, and H. Nguyen, "Adsynth: Synthesizing realistic active directory attack graphs," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Piscataway, NJ, USA: IEEE, 2024.

[26] I. Ward. (n.a.) Documentation for the json lines text file format. [Online]. Available: https://jsonlines.org/

[27] Microsoft. (2024, May) Configure scheduled quick or full microsoft defender antivirus scans. [Online]. Available: https://learn.microsoft.com/en-us/defender-endpoint/schedule-antivirus-scans

[28] Tenable. (2024, n.a.) Nessus 10.8.x user guide: Scan and policy settings. [Online]. Available: https://docs.tenable.com/nessus/Content/AdvancedScanSettings.htm

Appendix A
Constructing an α-Metagraph from an ADSynth output file

We parse the file into nodes, edges representing ownership of OUs over its child objects and edges indicating security permissions (Lines 2-4). Next, we create nodes in the α-Metagraph (Lines 5-7) and add element nodes to its corresponding sets (Lines 8-10). Finally, we build the meta-edges between those sets and return the complete α-Metagraph.

Algorithm 5: Constructing an α-Metagraph from an ADSynth output file

Input: file - an ADSynth output file, containing information about AD entities and their relationships

1 $MG = create\_metagraph()$
2 $row\_node = parse\_node(file)$
3 $row\_edge\_ownership = parse\_edge\_ownership(file)$
4 $row\_edge\_permission = parse\_edge\_permission(file)$
5 **for** $node$ *in* row_nodes **do**
6 $MG.create\_node(node.properties)$
7 **for** $edge$ *in* $row\_edge\_ownership$ **do**
8 $add\_to\_set(MG, edge.start, edge.end)$
9 **for** e *in* $row\_edge\_permission$ **do**
10 $MG.create\_meta\_edge(MG, e.start, e.end, e.type)$
11 **return** MG

Appendix B
Dependencies of nodes and edges in the ILP Formulation

In the transformed α-metagraph G_{MT}, given an attack path P of length greater than 0 from an entry node x to a target node t, the dependency between node x and all other nodes and edges on the path using the constraint (9) is shown as follows:

$$n(x) \geq n(t) - \sum_{\langle u,z\rangle \in P_{x\to t}} cut(\langle u,z\rangle) \tag{12}$$

Proof by Induction

Base Case:

For a path with a single edge, say $x \to y$:

$$n(x) \geq n(y) - \text{cut}(\langle x, y \rangle) \tag{13}$$

This matches the required form for the path with a single edge.

Inductive Step:

Assume the inequality holds for a path with k edges, say $x \to q$:

$$n(x) \geq n(q) - \sum_{\langle u,z \rangle \in P_{x \to q}} \text{cut}(\langle u, z \rangle) \tag{14}$$

where $P_{x \to q}$ is a set of edges on an attack path from x to q.

We need to prove that the inequality holds for the attack path $x \to \cdots \to q \to w$ with $k+1$ edges as follows:

$$n(x) \geq n(w) - \sum_{\langle u,z \rangle \in P_{x \to w}} \text{cut}(\langle u, z \rangle) \tag{15}$$

Applying the constraint to the edge $q \to w$, we have:

$$n(q) \geq n(w) - \text{cut}(\langle q, w \rangle)$$

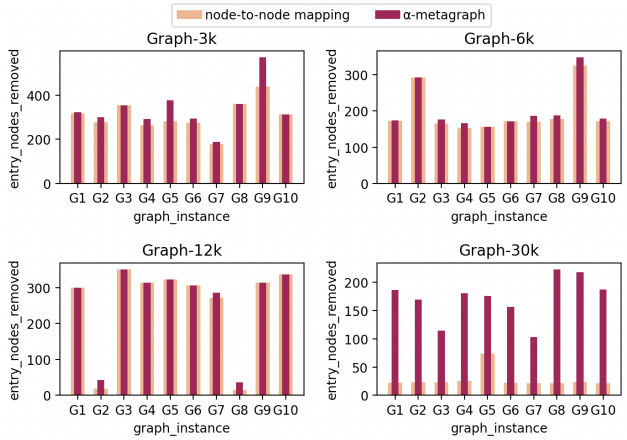
Substitute $n(q)$ in (13) into (14), we have:

$$\begin{aligned} n(x) &\geq n(q) - \sum_{\langle u,z \rangle \in P_{x \to q}} \text{cut}(\langle u, z \rangle) \\ &\geq [n(w) - \text{cut}(\langle q, w \rangle)] - \sum_{\langle u,z \rangle \in P_{x \to q}} \text{cut}(\langle u, z \rangle) \\ &= n(w) - [\text{cut}(\langle q, w \rangle) + \sum_{\langle u,z \rangle \in P_{x \to q}} \text{cut}(\langle u, z \rangle)] \\ &= n(w) - \sum_{\langle u,z \rangle \in P'_{x \to w}} \text{cut}(\langle u, z \rangle) \end{aligned}$$

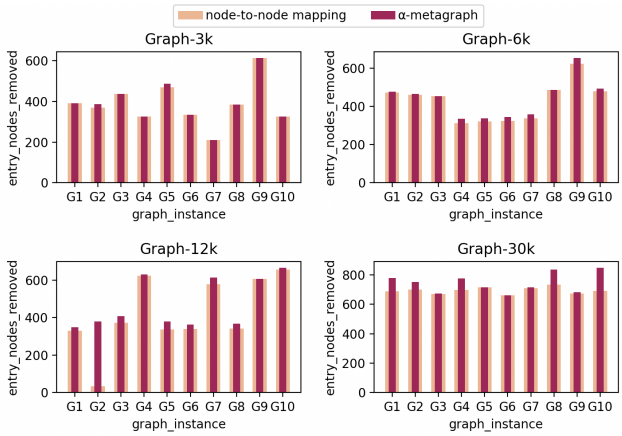
Thus, the inequality holds for the path with $k+1$ edges.

By induction, the inequality holds for any path from x to t.

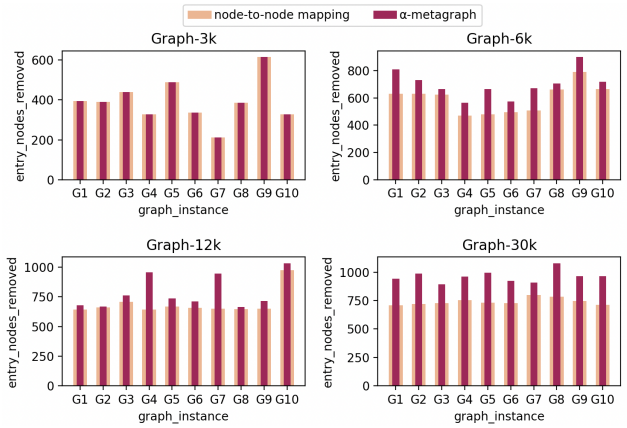
APPENDIX C
COMPLETE EXPERIMENTS FOR VARIOUS GRAPH SIZES AND BUDGETS



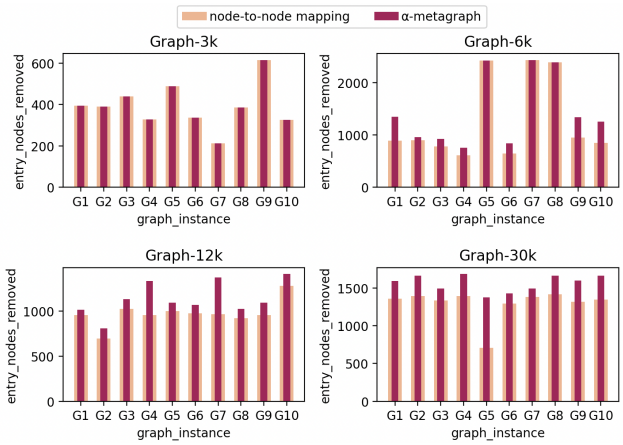
(a) budget = 5



(b) budget = 10

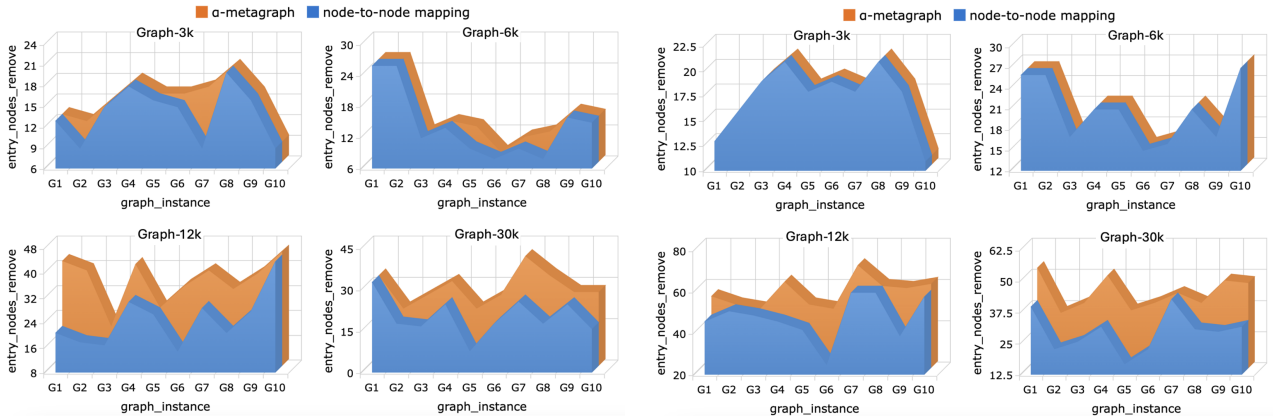


(c) budget = 15



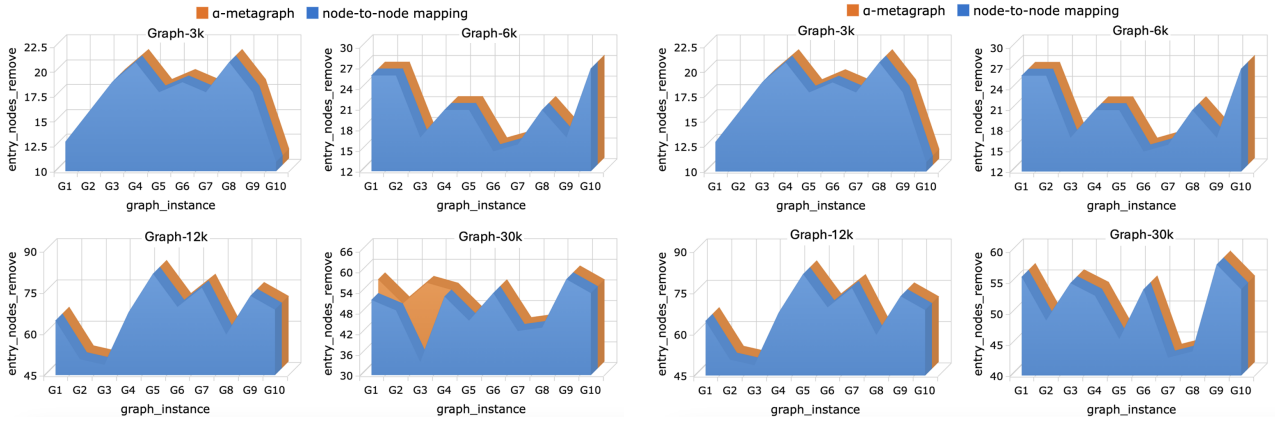
(d) budget = 20

Fig. 17: Long-term mitigation in Vulnerable AD systems: Effectiveness of defense strategies under various budgets



(a) budget = 5

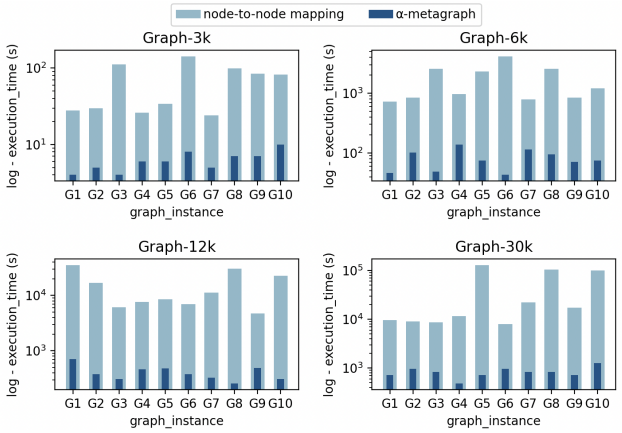
(b) budget = 10



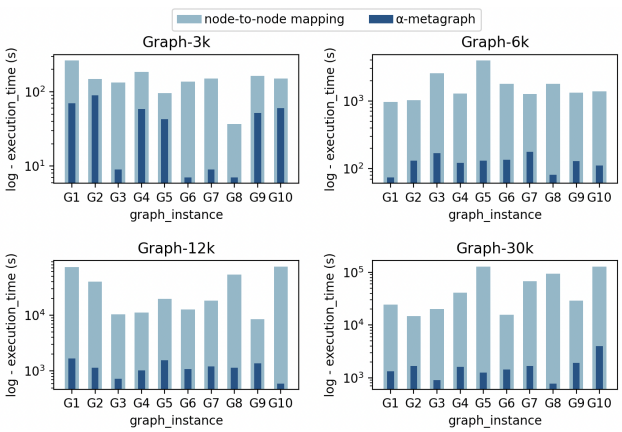
(c) budget = 15

(d) budget = 20

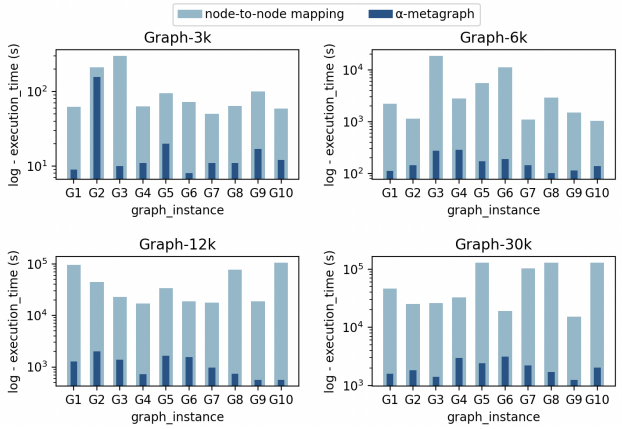
Fig. 18: Long-term mitigation in Secure AD systems: Effectiveness of defense strategies under various budgets



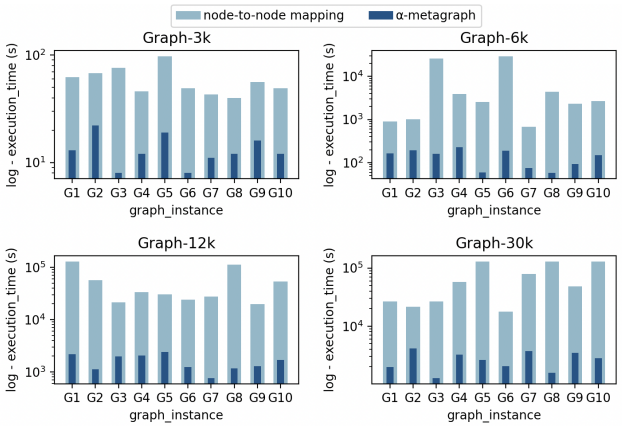
(a) budget = 5



(b) budget = 10

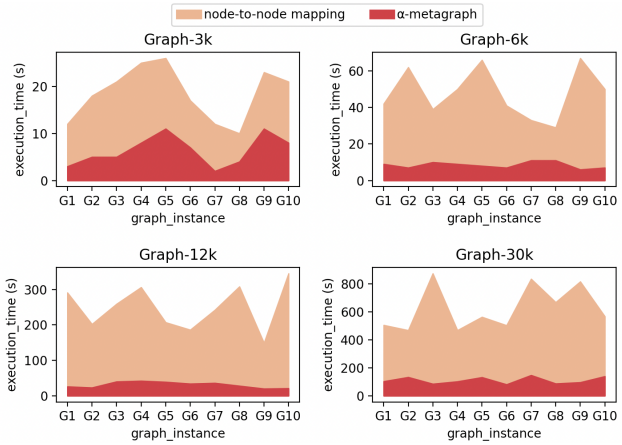


(c) budget = 15

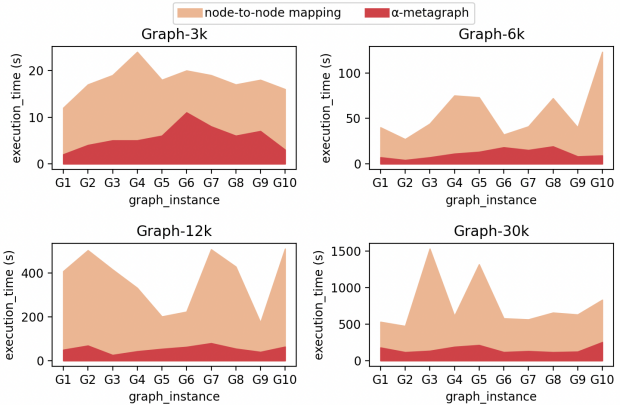


(d) budget = 20

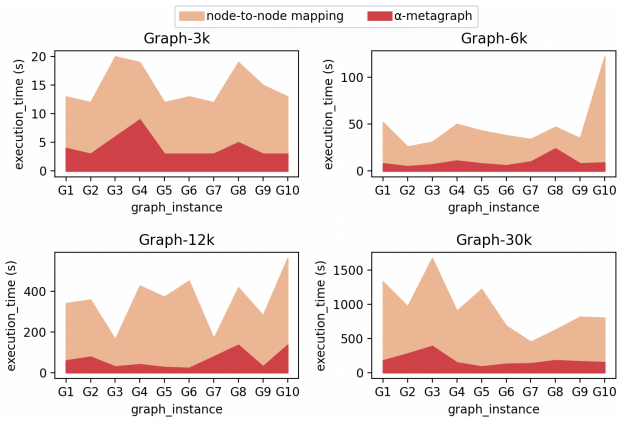
Fig. 19: Execution time in Vulnerable AD systems, capped at 36 hours



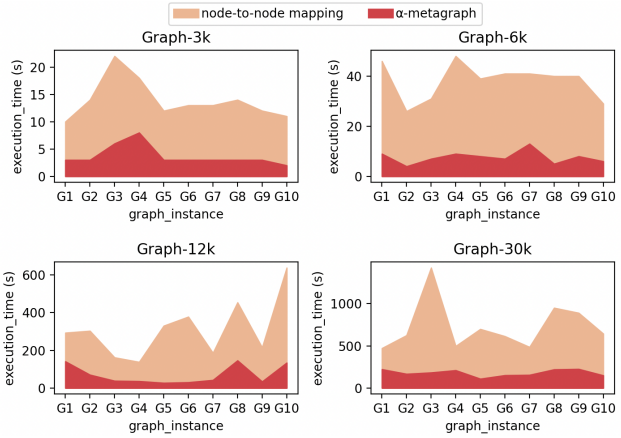
(a) budget = 5



(b) budget = 10

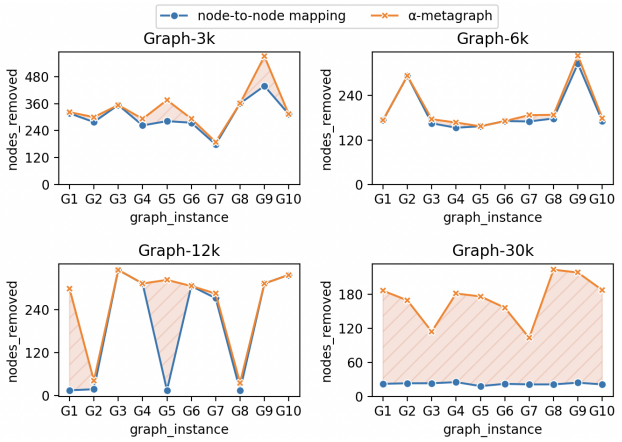


(c) budget = 15

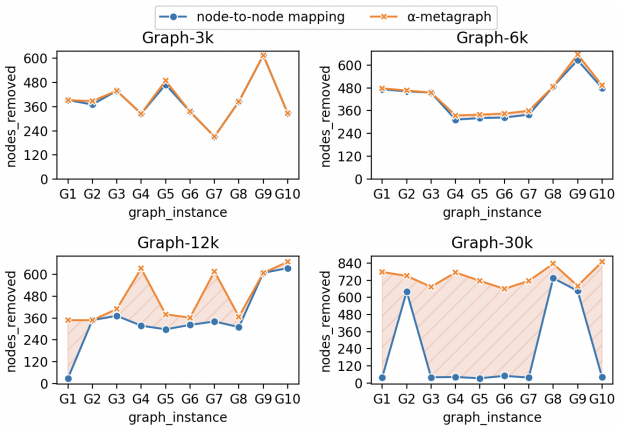


(d) budget = 20

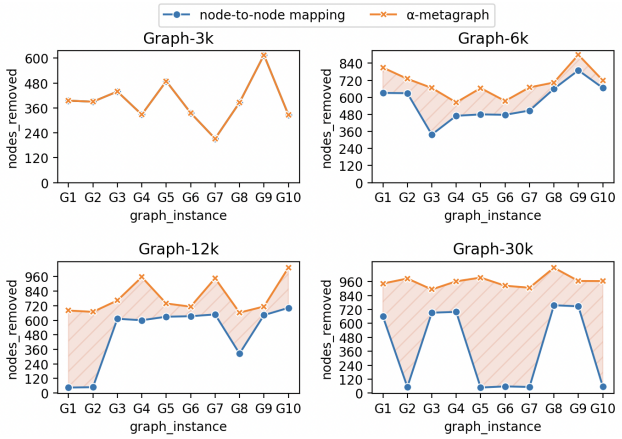
Fig. 20: Execution time in Secure AD systems



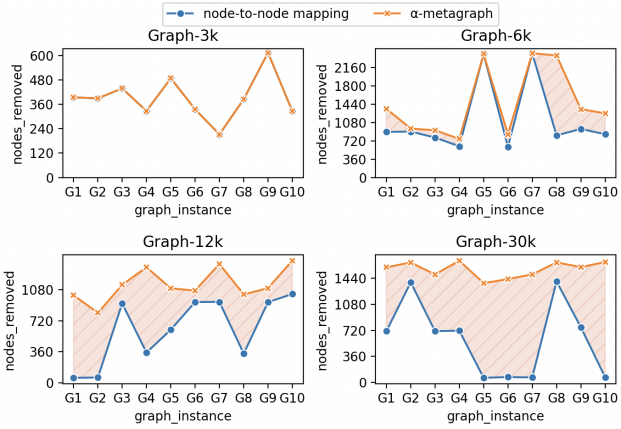
(a) budget = 5



(b) budget = 10



(c) budget = 15



(d) budget = 20

Fig. 21: Rapid mitigation in Vulnerable AD systems: Effectiveness of defense strategies under various budgets