RESEARCH-ARTICLE

A Scalable Double Oracle Algorithm for Hardening Large Active Directory Systems

YUMENG ZHANG, The University of Adelaide, Adelaide, SA, Australia

MAX WARD, The University of Western Australia, Perth, WA, Australia

MINGYU GUO, The University of Adelaide, Adelaide, SA, Australia

HUNG X NGUYEN, The University of Adelaide, Adelaide, SA, Australia

Open Access Support provided by:

The University of Adelaide

The University of Western Australia

Published: 10 July 2023

Citation in BibTeX format

ASIA CCS '23: ACM Asia Conference on Computer and Communications Security
July 10 - 14, 2023
VIC, Melbourne, Australia

Conference Sponsors:
SIGSAC

A Scalable Double Oracle Algorithm for Hardening Large Active Directory Systems

Yumeng Zhang
yumeng.zhang01@adelaide.edu.au
The University of Adelaide
Adelaide, Australia

Max Ward
max.ward@uwa.edu.au
University of Western Australia
Perth, Australia

Mingyu Guo
mingyu.guo@adelaide.edu.au
The University of Adelaide
Adelaide, Australia

Hung Nguyen
hung.nguyen@adelaide.edu.au
The University of Adelaide
Adelaide, Australia

ABSTRACT

Active Directory (AD) is a popular information security management system for Windows domain networks and is an ongoing common target for cyber attacks. Most real-world Active Directory systems consist of millions of entities and links, and there are currently no efficient and effective solutions for hardening Active Directory systems of such scale. In this paper, we propose a novel and scalable double oracle-based algorithm for hardening large AD systems. We formulate the problem as a Stackelberg game between the defender and the attacker on a weighted AD attack graph, where the defender acts as the leader with a budget, and the objective is to find an optimal defender's pure strategy. We show that our double oracle-based solution has significantly improved speed and scalability compared with previous solutions for hardening AD systems. Lastly, we compare with GoodHound weakest links and show that our solution provides better recommendations for targeting the elimination of optimal attack paths.

CCS CONCEPTS

• **Networks** → **Network security**; • **Mathematics of computing** → **Paths and connectivity problems**; • **Theory of computation** → **Network games**.

KEYWORDS

Active Directory, Network security, Attack graph, Stackelberg game, Double oracle

ACM Reference Format:
Yumeng Zhang, Max Ward, Mingyu Guo, and Hung Nguyen. 2023. A Scalable Double Oracle Algorithm for Hardening Large Active Directory Systems. In *ACM ASIA Conference on Computer and Communications Security (ASIA CCS '23), July 10–14, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3579856.3590343

ASIA CCS '23, July 10–14, 2023, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0098-9/23/07…$15.00
https://doi.org/10.1145/3579856.3590343

1 INTRODUCTION

Cyber attack graphs model the chain of events (conceptual or physical) that lead to successful cyber attacks. Currently, there are more than 80 attack graph/tree models and over 90 different definitions of attack graphs [13]. Among these, the most prominent and actively used attack graph model is the Active Directory (AD) attack graph, first proposed in [4, 6] and then commercialised by the BloodHound tool [22]. In these graphs, nodes represent components including user accounts, computers and security groups in a network. A directed edge from node A to B represents that an attacker can reach B from A via existing access or a known exploit. Attackers use the AD attack graphs to map out attack paths from a compromised node to a target node with each step corresponding to one or more exploits. An example of such AD attack graph is shown in fig. 1, in which the red node denotes a compromised user account and the golden node marks a high-value asset targeted in the attack. BloodHound is one of the most popular tools used by hackers and AD attack graphs provided by BloodHound have been used in many recent large-scale cyber-attacks including Conti Ransomware [21].

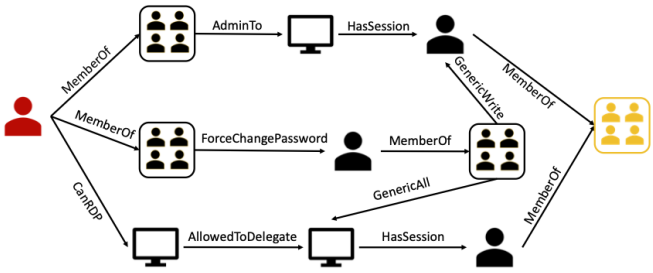

Figure 1: Example of an AD attack graph

Securing AD networks require methods to eliminate attack paths in the AD attack graphs—a general concept often referred to as *network hardening*. There are currently no good solutions for cutting AD attack graphs to stop these attacks. Graph interdiction solutions [12] typically work on small graphs of up to hundreds of nodes. Real AD networks are very large and complex. A typical AD network for a medium size organisation contains hundreds of thousands of nodes, millions of edges and billions of attack paths in their AD networks [20]. There are more than 20 types of nodes and 12 types of edges of various properties in each of these graphs.

Because of the scale and complexity of exploitable connections, the topological distribution of a real-world AD attack graph can be elusive to formalise, which increases the difficulty of identifying optimal edges for cutting. Furthermore, some nodes and edges cannot be cut as they are required for critical and legacy services. Despite decades of research, current algorithmic solutions for hardening AD networks are still not practical for real systems [20]. Without scalable algorithms, commercial solutions such as GoodHound [15] rely on heuristics that do not optimally protect the networks, leaving significant gaps in current cyber defence.

Recent advances in fixed parameter and double oracle analyses have led to new algorithms for hardening large AD networks [10]. Double oracle is a heuristic for optimisation problems that can be modelled as a game between two players. An oracle generates a best response action for a player given the opponent's strategy space. Double oracle allows the opponent's strategy space considered by an oracle to be restricted, therefore improving the efficiency when compared with single oracle solutions, which often struggle to converge due to each player's strategy space being exponentially large. Linear programming (LP) provides the foundation for these optimisation problems. Generally, components where defence resources are potentially allocated are created as variables, over which an objective function is defined to model a security-related parameter that a security engineer aims to strengthen against an attack. However, LP often suffers from the problem of explosion in the number of variables when dealing with large attack graphs, even in double oracle solutions. RUGGED [12] is a double oracle-based solution for a network interdiction problem, in which the defender's oracle utilises a Mixed Integer Linear Program (MILP) for finding an optimal set of edges over which defence resources are allocated, and the objective is to find an optimal defender's mixed strategy in a Nash equilibrium. Nevertheless, RUGGED has all edges in an attack graph defined as variables, which causes a large deterioration in speed on graphs with merely hundreds of nodes. To resolve this issue, we look for ways to narrow down the edge candidates over which security-hardening measures are deployed, so as to mitigate the explosion in the number of variables and improve the convergence speed. In the algorithms we propose in this paper, this is achieved by first identifying representative attack paths in which optimal edges are potentially located, and only creating variables for edges among the set of paths. As a result, the number of variables is greatly reduced in the LP and the scalability of the algorithm is improved.

In this paper, we study the security hardening problem of large AD systems through limited edge removal. In our double oracle approach, we aim for finding the optimal defender's pure strategies, which are exact links cutting which would give the best attack path elimination result. We believe the defender's optimal pure strategies can provide more practical and concrete guidance for the security administrator, as an optimal pure strategy informs exactly which links to cut that would effectively reduce the attacker's chance of gaining the highest privileges in the system. Moreover, unlike many network security problems, the environment of an AD system is completely transparent to an attacker, the existence of a link is deterministic and can be quickly obtained with SharpHound[1]. Therefore, having a mixed defender's strategy is not felicitous in this context.

Previous solutions are not ideal for our problem due to their different definitions of models and objectives as well as drawbacks with scalability. Our main contributions are:

- Adaptation of the double oracle algorithm for finding an optimal defender's pure strategy of a Stackelberg game, which provides an AD hardening solution with better efficiency and scalability.
- Improvement in the computational efficiency of the double oracle algorithm by modelling the attacker's oracle to be polynomial-time solvable, and narrowing down the space from which the defender's oracle generates a best response action.
- An evaluation of GoodHound's method for finding weakest links.

Through a series of experiments with an array of graphs under variously randomised settings, we show that the double oracle-based algorithm empirically has an outstanding speed and scalability, efficient even on graphs of real-world network sizes. When running on graphs of almost 100,000 nodes, the double oracle algorithm converges within seconds, while the fastest previous solution takes over an hour. By comparing with GoodHound weakest links, we argue that attack paths giving the attacker the greatest chance of success are not necessarily those having the smallest number of hops, and removing edges with an intention of eliminating most number of paths is not guaranteed to reduce the attacker's chance of reaching their target. For these reasons, the double oracle algorithm can be more suitable for targeting the elimination of optimal attack paths.

2 RELATED WORK

2.1 Attack Graph Management

Since the seminal paper [24], there have been more than 2000 papers written on the topic of attack graph generation, management and applications. Of most relevant to our work on managing attack graphs are [3, 5, 7, 19, 23, 26]. In both [26] and [7], attack graphs are logic-based for modelling sequences of actions of a successful attack. In this type of attack graphs, nodes denote conditions, actions and effects in an attack and edges represent causal relationships between events. Also logic-based, Dewri *et al.* [5] used a tree structure instead of a graph to reduce the complexity and better reflect the hierarchical structure of the attacker's goals. In [19], Poolsappasit *et al.* adopted a Bayesian attack graph to model the state of network properties effectuating with probabilities that an attacker could exploit, so as to quantify the probabilities of different network compromise levels for risk management. Sawilla and Ou [23] modelled a dependency attack graph that embodies vulnerabilities and privileges assigned with logical types as vertices and their inner dependencies. Both the vertices and edges are assigned weights for representing vertices' inherent values and success likelihood respectively. With slight differences, Borbor *et al.* [3] modelled the attack graph based on an extended resource graph which has network resource instances as nodes instead of vulnerabilities, and edges represent causal relationships of attacker's exploitation. In our paper, we extend from the model designed by

[1] https://bloodhound.readthedocs.io/en/latest/data-collection/sharphound.html

BloodHound, which is similar to the attack graph in [23] and [3] that has vulnerabilities and resources in an AD system represented as nodes. We assume that each attack path is a sequence of successful exploitation of components connected with "AND" logic, and each successful exploitation is an independent event. Instead of assigning weight values on both vertices and edges, we only assign on edges for computing the attacker's success probability.

Attack graphs can be used for the derivation of security metrics and scoring systems for various types of networks [17, 18]. However, in this paper, our use of attack graphs prioritises the derivation of security-hardening solutions rather than the quantification of security level.

AD attack graph was first explained in [6]. In that paper, Dunagan *et al.* proposed a machine learning-based algorithm to cut the AD attack graph into disconnected components. In our study, we assume that the attacker's goal is to compromise Domain Admins (DA)—a group of accounts of the highest privileges, from a set of compromised user accounts of low privileges. The corresponding security-hardening measure is to remove a set of edges within a defence budget that would minimise the attacker's chance of successfully reaching DA.

Similar to our approach for hardening the security of an AD network, BloodHound Enterprise [22] offers attack path remedial solutions by way of identifying choke points, which are critical edges that provide most user accounts and computers in the network with pathways to high-value assets. However, we cannot study its performance in our model, as it is not open-source.

Developed based on BloodHound for solving a similar problem, GoodHound [15] is an open-source AD security monitoring tool that aims at reducing the number of exposed users by flagging the busiest paths and weakest links. Attack paths are identified by finding a shortest path from each Group node to high-value assets, then displayed in a prioritised order based on a heuristic risk score reflecting an exploitation cost of the path and the number of non-admin users exposed. Weakest links are recognised to be those most commonly shared among the paths, which can be used for guiding the mitigation of attack paths. However, as we shall point out in section 6.2, removing edges with an intention of eliminating the most number of attack paths does not necessarily reduce the attacker's chance of successfully reaching their target, and our double oracle-based algorithm could provide better suggestions for prioritising the elimination of optimal attack paths.

2.2 Security Games and Double Oracle Based Solutions

One common approach for developing security-hardening solutions is the use of game theory for optimising defence strategies. Nguyen *et al.* [16] studied the problem of moving target defence on multi-stage attack graphs, in which the defender's strategy proactively responds to dynamics in network configurations. In [1], Abdallah *et al.* added prospect theory into the game model and took into account misperceptions the defender may have about the attacker's success probabilities when optimising the allocation of defence resources. In this paper, the security-hardening approach we study is the optimal cutting of links, and our model seeks an overall increase in exploitation difficulty by way of worsening the best attacking conditions for an attacker to the greatest extent possible.

In [10], Guo *et al.* studied the shortest path interdiction problem modelled as a Stackelberg game between the defender and the attacker on AD attack graphs. The defender's task is to block a set of edges under a defence budget so as to maximise the expected shortest path length of the attacker, where the path length is measured in terms of the number of hops. Later in [11], Guo *et al.* further explored the problem with the attacker's success probabilities assigned as edge weights, and proposed a faster solution with the utilisation of LP. The double oracle algorithm we propose in this paper is based on an AD attack graph model similar to that in [8, 9, 11], however with all AD edge types added into the attack graph. In section 6.2, we show that our double oracle-based solution has a better speed and scalability on AD attack graphs of a larger scale.

Double oracle was first proposed by McMahan *et al.* in [14], where it was used to efficiently solve the problem of robot navigation among detection adversaries. In [12], Jain *et al.* introduced RUGGED, which is a double oracle-based approach for a road network security problem modelled as a two-player zero-sum game on undirected graphs. The objective is to find the defender's mixed strategy in a Nash equilibrium. The key idea of their approach is that in each optimising iteration, a player's oracle generates a best response action against the opponent's mixed strategy in a local Nash equilibrium given the opponent's restricted strategy space. The best response actions are used for updating restricted strategy space iteratively until terminating conditions are met which signifies the convergence to a global Nash equilibrium. In this way, the computational costs associated with the enumeration of a large number of cost vectors can be reduced in most cases.

Similar to [12], we are using the double oracle approach for a network security problem modelled as a game, in which we aim to minimise the impact of an adversary. In contrast to a conventional security game model, the objective of our problem is to find the optimal defender's pure strategy, and we have each player's utility determined by edges in attack paths rather than targets. As edges giving the attacker a greater utility have smaller weight values in our model, the attacker's problem of finding an optimal strategy is converted to a shortest path problem as opposed to an NP-hard problem as in RUGGED, therefore is polynomial-time solvable. Furthermore, in our double oracle-based solution, the defender's strategy space from which a best response is generated is scaled down in line with the attacker's restricted strategy space under consideration. Compared with having a full defender's strategy space as in RUGGED, the number of variables over which the defender's utility is optimised can be immensely reduced. With the above adaptations, our double oracle-based algorithm has a significantly improved speed and scalability.

3 PROBLEM FORMULATION

3.1 Attack graph

We define an AD attack graph to be a directed, weighted graph $G = (V, E)$. A set of vertices V represents all physical and virtual components in an AD network. Directed edges E denote all AD link

types modelling security dependencies and administrative privileges between components, which represent opportunities for an attacker to make lateral movements.

We assume that the attacker's goal is to compromise DA as it would provide the attacker with full access to the system, including establishing persistent backdoors. Corresponding to how an attack is started in an Identify Snowball Attack [6], we define a set of entry nodes $S = \{s_1, s_2, \ldots, s_n\}$, $S \subset V$ as source nodes which the attacker has already compromised at the start of an attack, and one terminal node $t \in V$ as DA.

Each edge has an attacker's failure probability caused by multiple factors including the existence of a link at the time of exploit, security measures deployed and difficulty of exploitation [2]. To simplify the assessment of attack plans, we define a function $Pr : E \to \mathbb{R}$ that assigns a success probability $Pr(e) \in [0, 1]$ for $e \in E$ that accommodates all factors causing a certain level of attack failure.

Each attack path gives the attacker a chance of success. Let $p = (v_0, v_1, \ldots, v_k)$ be an attack path from v_0 to v_k where $v_i \in V$ for $i = 0, 1, \ldots, k, v_0 \in S$ and $v_k = t$. We assume that each edge being successfully exploited is an event independent of that for other edges on the same path. Under this assumption, the attacker's success probability given by an attack path p is calculated as

$$\begin{aligned} Pr(p) &= Pr(v_0, v_1) \cdot Pr(v_1, v_2) \cdot \ldots \cdot Pr(v_{k-1}, v_k) \\ &= \prod_{i=0}^{k-1} Pr(v_i, v_{i+1}) \end{aligned} \tag{1}$$

Taking the logarithm of (1) yields

$$\begin{aligned} \ln(Pr(p)) &= \ln(Pr(v_0, v_1) \cdot Pr(v_1, v_2) \cdot \ldots \cdot Pr(v_{k-1}, v_k)) \\ &= \ln(Pr(v_0, v_1)) + \ln(Pr(v_1, v_2)) + \ldots \\ &\quad + \ln(Pr(v_{k-1}, v_k)) \end{aligned} \tag{2}$$

Equation (2) now resembles the summative form for calculating path length with weight values on edges. However, as a success probability falls within the range of $[0, 1]$, its natural logarithm could be a negative value. If we take its negated form and have $w(u, v) = -\ln(Pr(u, v))$ as the weight value assigned on an edge (u, v), the length of path p can be computed as

$$\begin{aligned} len(p) &= w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_{k-1}, v_k) \\ &= -\ln(Pr(v_0, v_1)) - \ln(Pr(v_1, v_2)) - \ldots \\ &\quad - \ln(Pr(v_{k-1}, v_k)) \\ &= -\ln(Pr(v_0, v_1) \cdot Pr(v_1, v_2) \cdot \ldots \cdot Pr(v_{k-1}, v_k)) \\ &= -\ln(Pr(p)) \end{aligned} \tag{3}$$

Based on eq. (3), the attacker's success probability given by a path of length l is calculated as

$$Pr(l) = e^{-l} \tag{4}$$

With eq. (4), the shorter a path is, the better the chance of successfully reaching DA is for the attacker.

[2] https://github.com/idnahacks/GoodHound/wiki/Exploit-Cost

3.2 Formulation with game theory

The problem of defending an AD network can be modelled as a Stackelberg game with the defender acting as the leader and the attacker following. The defender's strategy space includes all combinations of no more than b edges from E, the size of which should be $\sum_{k=1}^{b} \binom{|E|}{k}$. The attacker's strategy space consists of all paths from S to t. As such, both players have a strategy space that is exponentially large.

Let $\mathcal{P}(G)$ be the set of all attack paths from S to t that can be found in G, and the length of the shortest path in $\mathcal{P}(G)$ is denoted as $\delta(\mathcal{P}(G))$, which is calculated with edge weights following eq. (3). The defender's utility from removing a set of edges ε given the attacker's strategy space $\mathcal{P}(G)$ is determined by the level of reduction in the attacker's success probability from removing ε. Formally, it can be defined as

$$U_D(\varepsilon, P) = e^{-\delta(\mathcal{P}(G))} - e^{-\delta(\mathcal{P}(G \setminus \varepsilon))} \tag{5}$$

where $\varepsilon \subseteq E$ and $|\varepsilon| \leq b$.

The attacker is assumed to be a follower that makes a best response action by observing the defender's strategy. The attacker's utility from a proposed attack path p given the observed defender's strategy ε is then calculated as

$$U_A(p, \varepsilon) = \begin{cases} e^{-len(p)} & \text{if } p \text{ does not have any edge in } \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Therefore the attacker's problem of maximising U_A is essentially finding a shortest path that does not go through edges in ε.

The objective is to find an optimal defender's pure strategy for achieving a maximum reduction in the attacker's success probability given by an AD attack graph.

4 A BRUTE-FORCE ALGORITHM

Though the problem can be modelled as a Stackelberg game between the two players, it can also be viewed as a problem of removing b edges to eliminate shortest paths so as to maximise the length of the shortest path remaining in G. If a super-source node is added to connect to all source nodes, the problem is equivalent to the *Most Vital Arcs* problem defined in [2], which has been proven to be NP-hard.

There are two main tasks in the computational process: the enumeration of shortest paths and determining if removing at most b edges can eliminate all enumerated paths. When the budget b is no longer adequate, the length of the last path enumerated is the optimal result that the defender can achieve within budget.

In order to decide the adequacy of the defender's budget, we formulate an integer program as follows.

Let $P = (p_1, p_2, \cdots, p_k)$ be the set of attack paths to be eliminated. Path $p = (v_1, v_2, \cdots, v_n)$ is eliminated if it has at least one edge $e_i = (v_i, v_{i+1})$, $1 \leq i < n$ being removed. In the integer program, $\gamma(p)$ assigns a path $p \in P$ with either 0 or 1 and $\gamma(p) = 1$ if p is eliminated. Let E^* be the set of binary variables representing edges on paths in P to be potentially removed. Similar to $\gamma(p)$, $e \in E^*$ equals 1 if the corresponding edge is removed. To reduce the size of E^*, only removable edges that are shared by at least two paths in P are created as variables. If there are paths in P that do

not have such edges, a random removable edge on each of those paths can be chosen to add into E^*.

$$\begin{aligned}
\text{maximise} \quad & \sum_{p \in P} \gamma(p) \\
\text{subject to} \quad & \gamma(p) \leq \sum_{e \in p} e, && \forall p \in P \\
& \gamma(p) \cdot |\{e \mid e \in p\}| \geq \sum_{e \in p} e, && \forall p \in P \\
& \sum_{e \in E^*} e \leq b, \\
& e \in \{0,1\}, && \forall e \in E^* \\
& \gamma(p) \in \{0,1\}, && \forall p \in P
\end{aligned} \tag{7}$$

The integer program we propose maximises the number of paths eliminated by removing edges the quantity of which does not exceed b. The last two constraints ensure that all variables are binary. $\sum_e e \leq b$ imposes a limit on the number of removed edges. The first two constraints formulate an"OR" function defined as $\gamma(p) = e_1 \wedge e_2 \wedge \ldots \wedge e_x$ for all edges that are created as variables on path p. $\sum_{e \in p} e \geq 1$ when at least one $e \in p$ equals one, $\gamma(p)$ could be 0 or 1 in $\gamma(p) \leq \sum_{e \in p} e$ as a result. To ensure that $\gamma(p) = 1$ in this case, we have the second constraint $\gamma(p) \cdot |\{e \mid e \in p\}| \geq \sum_{e \in p} e$ so that $\gamma(p)$ has to be 1 to ensure its validity. If none of the edges on p is removed, $\sum_{e \in p} e$ would be zero, and $\gamma(p)$ can only be zero to meet both of the constraints.

The enumeration of shortest paths can be accomplished with Yen's algorithm [25] after adding a super-source node connecting to nodes in S. Combining with the integer program, the brute-force algorithm formulated is shown as algorithm 1.

Algorithm 1 *incremental_opt*(G, b)

```
1:  P ← []
2:  while True do
3:    if has_path(G, supersource, t) then
4:      path ← find_next_path(G)
5:      P.append(path)
6:      if not full_elimination(P, b) then
7:        return len(path)
8:      end if
9:    else
10:     return ∞
11:   end if
12: end while
```

In the algorithm, P denotes the set of shortest paths enumerated for elimination. $find\_next\_path$ uses Yen's algorithm to find the next shortest path from the super-source node to t in addition to paths already inside P. $full\_elimination(P, b)$ runs the integer program (7) and returns whether the maximum number of paths eliminated equals $|P|$.

The computational complexity of this algorithm is mainly determined by the number of paths enumerated until the algorithm converges, which can grow exponentially large in the worst case.

5 A DOUBLE ORACLE BASED ALGORITHM

The double oracle approach finds an optimal solution by dividing the gameplay into restricted games. Two players sequentially generate a best response strategy given a restricted game, which is then used for updating the restricted game considered by the opponent. The algorithm terminates when certain conditions are met that signify convergence.

As we model the problem as a Stackelberg game, the defender's restricted strategy space considered in the attacker's oracle only has the latest set of edges that the defender committed to. For the defender's oracle, the attacker's restricted strategy space P includes a set of the attacker's previously proposed paths. P also defines a limited space from which the defender's best response action is generated, as eliminating all paths in P would give a maximum U_D given the attacker's restricted strategy space P considered, and the optimal set of edges can only be on paths in P.

At the start of the algorithm, the attacker's restricted strategy space P is initialised with one attack path—a path of the shortest length in the original graph. The defender's restricted strategy space should be empty. In the subsequent iterations, the defender commits to a new set of edges that maximises their utility given the restricted game, which is then observed by the attacker for proposing a new best response strategy.

In our algorithm, it is important to ensure that paths in P are ordered in a temporal sequence: the path proposed in iteration j should be placed after the path proposed in iteration i, $i < j$. The algorithm terminates if the defender cannot eliminate all paths in P under budget and paths in P are in increasing order of path length. Next, we show why it is crucial that attack paths in P are in increasing order of path length at termination.

Proposition 1. *let $P = (p_1, p_2, \ldots, p_k)$ be the attacker's restricted strategy space in iteration $k+1$ where the algorithm terminates. p_i is the shortest path found after $p_1, \ldots, p_{i-1}$ have been successfully eliminated under budget, and p_k is the shortest path proposed by the attacker in iteration k that causes the budget become inadequate in iteration $k+1$ and brings the algorithm to termination. If $len(p_1) \leq len(p_2) \leq \ldots \leq len(p_k)$, then $len(p_k)$ is the maximum shortest path length achievable by removing at most b edges, namely the minimum attacker's success probability that an optimal defender's pure strategy could accomplish.*

Proof. Suppose there is a path p_x in P, $1 \leq x < k$, that has $len(p_k) < len(p_x)$, then $len(p_k)$ returned by the algorithm is not optimal as the set of edges removed in iteration $x-1$ that brings $len(p_x)$ is giving a better optimisation result. In fact, suppose $len(p_{i-1}) > len(p_i)$ for $1 < i < k$, even if $len(p_1) \leq len(p_2) \leq \ldots \leq len(p_{i-1})$ and $len(p_i) \leq len(p_{i+1}) \leq \ldots \leq len(p_k)$, there is no guarantee that $len(p_k) \geq len(p_{i-1})$. Therefore, it is essential that proposition 1 holds for the optimality of the converged result. □

Taking proposition 1 into account, the algorithm we propose is shown as algorithm 2. *get_mincut* runs eq. (7) for deciding the adequacy of the budget b for a full elimination of P as well as returning edges selected for removal. Line 4-6 ensures the optimality of the result based on proposition 1 by comparing the length of the newly found path with the length of the last path added into

P and deleting paths longer than the new path in the attacker's restricted strategy space. A new defender's pure strategy should only be generated with P that has monotonically increasing path lengths. The algorithm returns the maximised shortest path length, which can be converted to the attacker's success probability with eq. (4).

Algorithm 2 $double\_oracle(G, b)$:

```
P ← []
path ← dijkstra(G, supersource, t)
while True do
    if len(path) ≤ len(P[−1]) then
        Delete all paths in P that are longer than
        len(path)
    end if
    P.append(path)
    ε ← get_mincut(P)
    if |ε| > b then
        return len(path)
    else
        remove ε from G
        if has_path(G, supersource, t) then
            path ← dijkstra(G, supersource, t)
            recover ε in G
        else
            return ∞
        end if
    end if
end while
```

The computational efficiency of this double oracle algorithm is difficult to formalise. In the next section, we run the algorithm on a large array of AD attack graphs with randomised settings to have an in-depth assessment of the algorithm's run-time performance.

6 EVALUATION

6.1 Run time

In this section, we run a series of experiments to empirically show that the double oracle algorithm has excellent computational efficiency under various graph settings.

In order to randomise the network environment in our experiments to a large extent, we generate AD networks using two of the popular AD simulation tools: adsimulator[3] and dbcreator[4]. To further ensure practicality, we also evaluate the algorithms using a real-world AD network - obtained from a University network. Furthermore, we arbitrarily initialise each AD attack graph with various edge removability distributions, not only as a way of taking into account possibilities in which certain edges cannot be tampered with, but also for further diversification of the optimal edges the algorithm converges to, such that the computational process of both algorithms is forced to change. In addition, we randomly initialise multiple sets of source nodes for each size under consideration to bring variations in attack paths potentially being eliminated.

[3]https://github.com/nicolas-carolo/adsimulator
[4]https://github.com/BloodHoundAD/BloodHound-Tools/tree/master/DBCreator

Table 1: Sizes of AD networks and attack graphs (adsimulator)

G	Active Directory		Graph	
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$
$G1$	4841	68550	32376	96085
$G2$	10191	154517	68204	212530
$G3$	50191	993137	336204	1279150
$G4$	95191	1892566	636270	2433645

The arbitrary initialisation of edge removability is accomplished with random sampling. Source nodes are randomly chosen among user nodes that are furthest away from DA in terms of the number of hops, so that the scope of the graph covered by the attack paths would be wider and the scale of the graph can exert more influence on the algorithms' performance. DA is the group node that has "DOMAIN ADMINS" in its properties and is marked "highvalue".

For experiments presented that do not involve linearly increasing graph sizes, we use four AD networks generated with adsimulator of different scales. In order to consider cases where there are multiple edges between two nodes forming multiple attack paths, we expand the Active Directory networks from those multi-edges by adding temporary nodes and edges so that they become separate paths. Data on the sizes of AD attack graphs before and after expansion are given in table 1.

Suppose removing $0, 1, 2, \ldots, b_1 - 1$ edges all leave an attacker's success probability of $Pr_0(G)$. When the number of removed edges is raised to b_1, the attacker's success probability becomes $Pr_1(G)$, $Pr_1(G) < Pr_0(G)$. As such, removing b_1 edges is a threshold for reducing the attacker's success probability by one level.

To give an overview of the algorithms' efficiency in reducing the attacker's success probability, we measure how the run time rises through the course of reducing the attacker's success probability by five levels. The result is presented as fig. 2. For this experiment, the size of source nodes is set to five, the impact of which on the randomness of graph settings is trivial, which we shall discuss later in this section. AD attack graphs of each size have network environment randomised in terms of edge removability distributions and choices of source nodes as clarified previously, the variance in the run time caused by this randomisation is included as each box in the plot. As revealed in the chart, the double oracle algorithm is capable of maintaining a fast speed in the process.

In the following experiment, we first vary the size of source nodes when the budget is set to five. We then evaluate the algorithms' performance under AD attack graphs of linearly increasing sizes when both the budget and the size of source nodes are five. For the latter, we have 10 AD attack graphs with the number of nodes ranging from 3000 to 30000 before expansion. Each of the ten AD networks is again randomised. As shown in fig. 3 and fig. 4, both the size of source nodes and the scale of the graph do not show an accountable impact on the run time of both algorithms. The double oracle algorithm's run time remains steady at a low level, while the incremental algorithm, which is defined as algorithm 1, also does not give a run time correlated with both parameters.

The budget, however, exhibits a traceable influence on the speed of both algorithms. As shown in fig. 5, when running on graphs

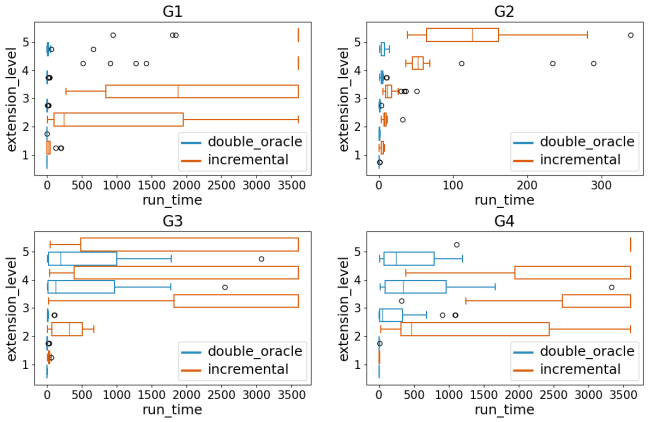

Figure 2: Run time comparison between the incremental and double oracle algorithm: reduction in the attacker's success probability by five levels ($|S| = 5$)

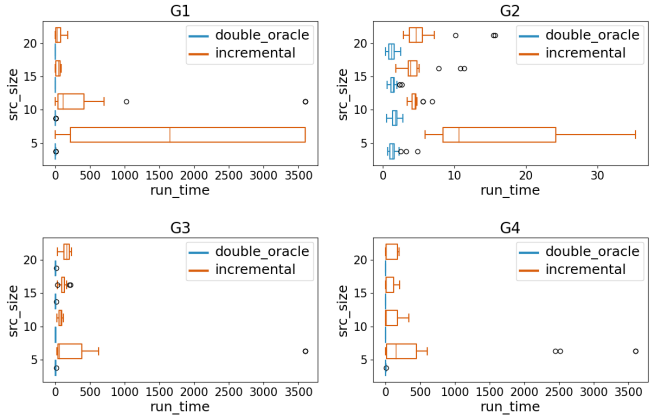

Figure 3: The size of S's impact on run time ($b = 5$)

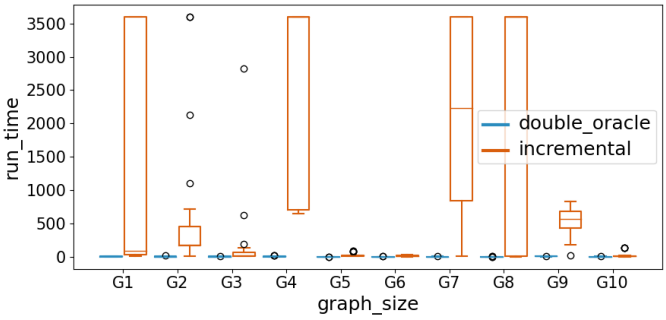

Figure 4: Graph size's impact on run time and enumerated paths ($b = 5$, $|S| = 5$)

shown in table 1, increasing budget drives up the run time of both algorithms, causing even the double oracle algorithm hit the run-time

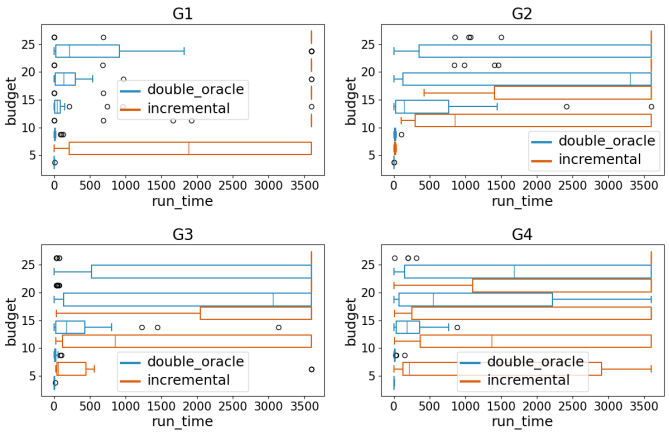

Figure 5: Budget's impact on run time ($|S| = 5$)

Table 2: Sizes of AD networks and attack graphs (dbcreator)

G	Active Directory		Graph	
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$
$G1$	5452	34911	9071	38530
$G2$	9053	64229	15070	70246
$G3$	49553	428436	82574	461457
$G4$	99953	965462	166570	1032079

limit in certain graphs. Nevertheless, the double oracle algorithm overall has better computational efficiency.

To further demonstrate that the double oracle algorithm's computational efficiency is applicable in broad application scenarios, we continue our testing of the algorithms on graphs generated with adsimulator under different network parameter settings, as well as graphs generated with dbcreator and a real-world AD network, each of which also has the edge removability distributions and source nodes randomised. Data on sizes of AD networks generated with dbcreator is shown in table 2. The real-world AD network we are testing on is from a university AD, which has 55949 nodes and 625835 edges. The network data we collected from the university does not reflect a full system environment, therefore has a relatively high security level where removing a small number of edges is sufficient for disconnecting DA from source nodes. Because of it, we are testing the algorithms' performance only when the budget is set to 5. The results are shown as fig. 6, fig. 7 and fig. 8, all of which indicate that the double oracle algorithm has a fast computational speed.

6.2 Comparisons with similar solutions

The integer programs proposed in [11] are currently the fastest solutions for network interdiction problem that provides optimal solutions, and is directly applicable to our model. We conducted a comparison between the two integer programs and the double oracle algorithm, the results are given in fig. 9, which shows that even though the efficiency of all algorithms is at an equal position

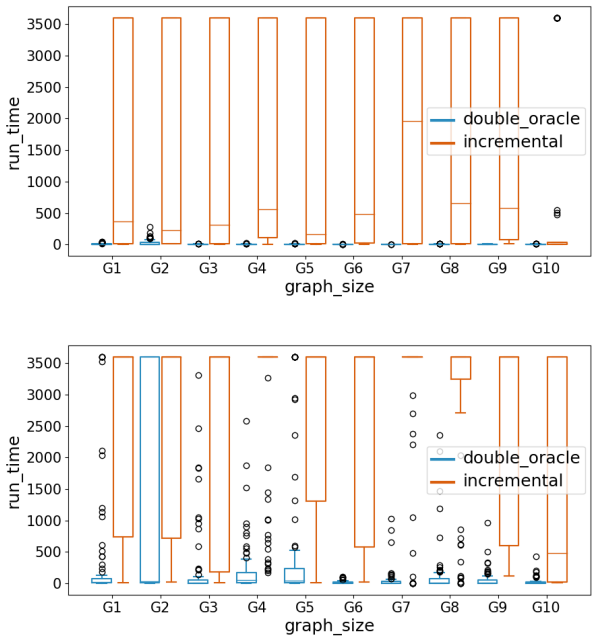

Figure 6: adsimulator with various AD parameters ($b = 5, 10$)

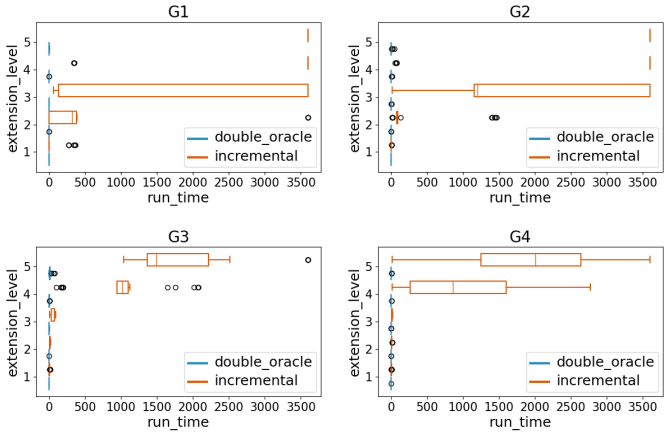

Figure 7: dbcreator: five levels of reduction in the attacker's success probability

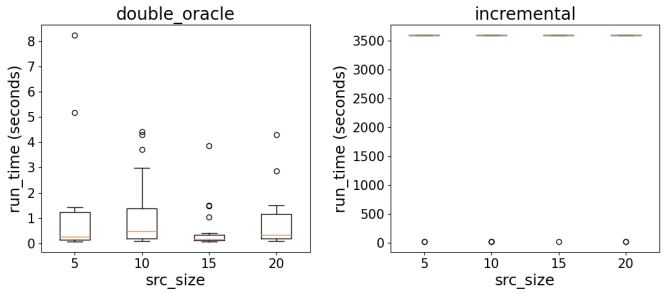

Figure 8: Application in a real-word AD system ($b = 5$)

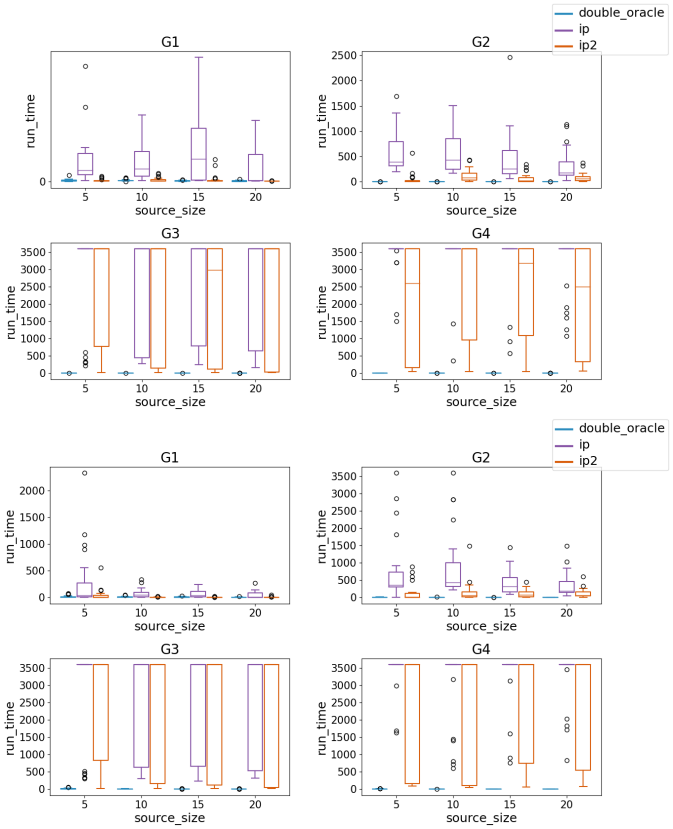

Figure 9: Run time comparison with LP: $b = 5, 10$

on smaller graphs, the advantage of the double oracle algorithm becomes more prominent as the scale of the graph expands.

As an open-source AD security monitoring tool, GoodHound provides a solution similar to ours with the identification of weakest links. In the following experiments, we explore whether removing the weakest links can effectively solve the problem of eliminating optimal attack paths.

We use the same AD attack graphs as in the previous section. For finding GoodHound weakest links in our AD attack graphs, we are temporarily setting all edge weights to one to be in line with GoodHound's method of finding shortest paths in terms of the number of hops. Unblockable edges on enumerated paths are left out of the ranking list of weakest links.

For simplicity, we only target the elimination of paths of the shortest length, namely paths giving the attacker the greatest chance of success.

When removing b^* edges is all it requires according to the double oracle algorithm, the percentages of projected paths eliminated by removing top-b^* GoodHound weakest links in each AD attack graph are shown in fig. 10. Except for rare cases given in $G3$ where there are merely 1 to 3 projected paths and the weakest link identified by GoodHound happens to cover them all, in most cases, removing top-b^* weakest links hardly gives a full elimination of projected paths, sometimes even giving zero elimination. This shows that

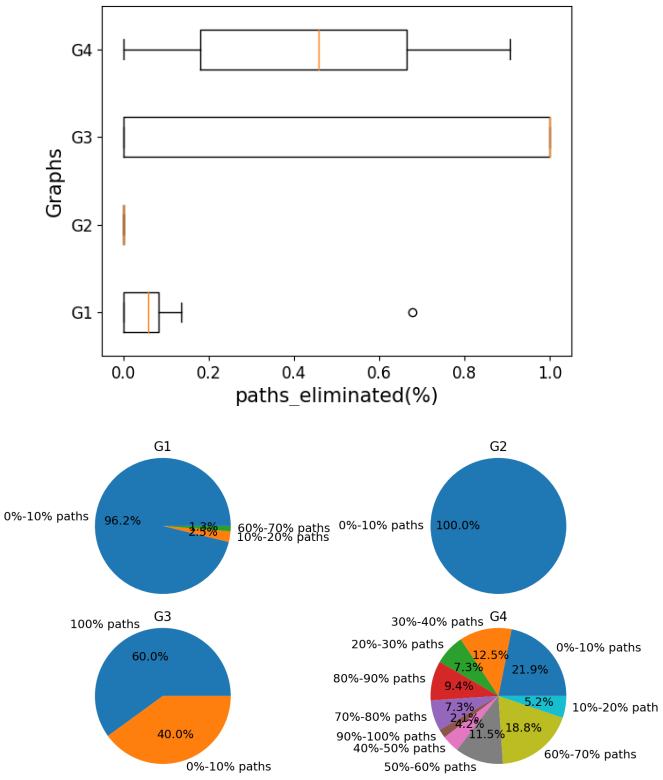

Figure 10: Percentages of projected paths eliminated by removing GoodHound weakest links

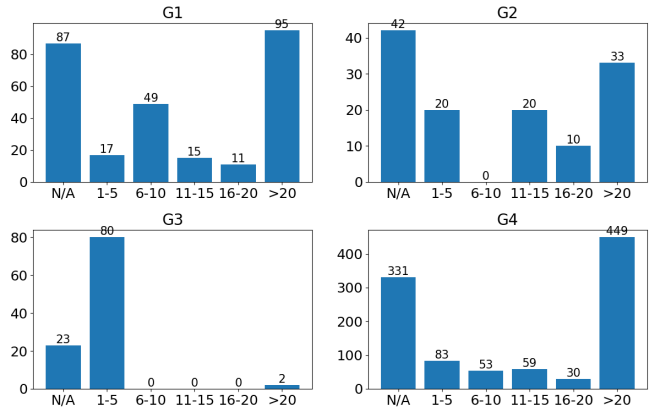

Figure 11: Rankings of double oracle edges among weakest links

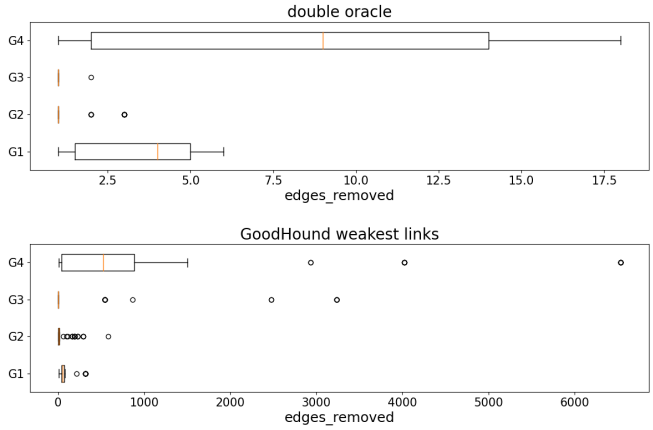

Figure 12: Comparison of the number of removed edges

weakest links cannot prioritise the elimination of optimal attack paths for an attacker.

Figure 11 shows the rankings of edges found by double oracle algorithm in the list of GoodHound weakest links, in which it indicates that edges in an optimal set could be of any ranking as a weakest link, and numerous edges are omitted from identification. Peculiarities of $G3$ provided some exceptional cases. Although weakest links indeed constitute a part of an optimal set, the single most optimal edge for the elimination of projected paths is not guaranteed to be the weakest, and removing a set of edges is inevitable in most cases, which further diversifies the rankings of edges in an optimal set.

Figure 12 shows the number of weakest links removed to eliminate projected paths compared with the minimum level of edge removal required. The contrast is self-evident: the number of weakest links removed to eliminate projected paths could range to thousands when removing less than 20 edges is sufficient. Consequently, following a ranking list of weakest links might cause an excessive number of edges being removed, potentially triggering greater disruptions in the network than necessary.

We observed that GoodHound only enumerates one shortest path from each relevant node to a high-value asset, while in most cases there should be numerous paths. Inadequate enumeration of attack paths being a reason for inaccurate evaluation of weakest links is worth considering. In addition, as there is hardly one most critical edge that enables all pathways to high-value assets, removing a set of edges for the effective elimination of connections is inevitable in most cases. If the set is not optimal, not only may there be a waste of defence resources and unnecessary disruption in network services, paths being omitted from removal by removing a suboptimal set of edges may even be those giving good chances for the attacker.

One interesting phenomenon we noticed is that shortest paths in terms of the attacker's success probability do not necessarily have the smallest number of hops. We believe this is a better reflection of the attacker's behaviour in the real world, as different link types in an Active Directory network should have different levels of exploitation difficulty. With some edges (u, v) that require zero exploitation effort and give 100% chance of success for the attacker, compromising u would mean an automatic expansion to v through the edge. As a result, if a path is abundant in such edges, even if it has a large number of hops, the attacker would favour this path over those having the smallest number of hops but consisting only of edges requiring onerous hacking jobs.

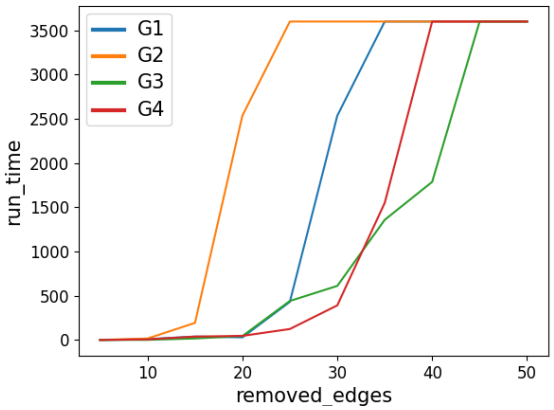

Figure 13: Double oracle: the relationship between the number of removed edges and run time ($|S| = 10$)

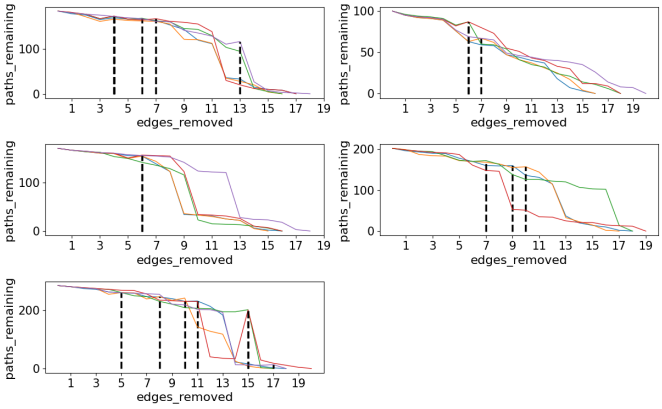

Figure 14: Double oracle: remaining projected paths to be eliminated as the number of removed edges increases ($G4$, $|S| = 20$)

6.3 Limitations of our solutions

As mentioned previously, the budget has an observable impact on the run time of the double oracle algorithm and may even cause it to reach a one-hour run time limit. Figure 13 provides a line chart of run time on some example graphs, which shows that the run time of the double oracle algorithm may grow exponentially as the budget increases.

In addition, as with all other optimal algorithms, double oracle is particularly vulnerable to unblockable paths—paths that consist only of unblockable edges. These paths represent the optimal attacker's success probability that the defender cannot further reduce and prevent the algorithm from continuing the elimination of paths. As a result, it cannot obtain an optimal number of eliminated paths.

In fact, as more edges are removed, the number of projected paths remaining with double oracle algorithm does not decrease strictly. To demonstrate this, we zoom into $G4$, the graph that requires the greatest amount of edges removed in general, for inspecting double oracle's path elimination process. Line charts in fig. 14 show the reduction process in the number of remaining paths as more edges are set for removal. Each chart provides results from one distinct set of source nodes of size 20. Points in the elimination process where the number of paths remaining rebounds are marked. This happens due to the fact that the double oracle algorithm maximises the resulted shortest path length, not the number of eliminated paths. In each iteration of the double oracle algorithm, there are multiple sets of edges eliminating which the same maximised shortest path length can be achieved, but there could potentially be some sets that are relatively more optimal in terms of maximising the number of projected paths eliminated, which are not selected by the algorithm. When more paths are sampled out and the number of which approaches the capacity of the budget in terms of the number of projected paths that can be eliminated, the optimality of edges chosen by the algorithm should improve. However, reducing the number of paths enumerated for computing the optimal result is one of the strengths of the double oracle algorithm that cannot be erased. Because of these segments of suboptimality, it cannot be guaranteed that the number of projected paths eliminated is optimal when there are unblockable paths.

Despite this limitation, we expect unblockable paths to be rare occurrences in real-world situations, as they signal deficiencies in the security of an AD system that a network administrator should be alert to. When the presence of unblockable paths is not a barrier, the double oracle algorithm could provide good guidance for hardening the security of an AD system.

7 CONCLUSION

In this paper, we studied the security hardening problem of AD systems by way of cutting links. We formulated the problem as a Stackelberg game between the defender and the attacker on an AD attack graph, and the objective is to find an optimal defender's pure strategy. We modelled the attack graph by extending from the model designed by BloodHound, with the attacker's probability of successful exploitation on each edge assigned as edge weights, which is similar to the model in [11], but with all AD edge types considered. We adopted the double oracle-based approach with the attacker's oracle being polynomial-time solvable, and the defender's oracle generating the best response from a space that is also restricted in line with the attacker's restricted strategy space considered, therefore having improved computational efficiency and scalability. We tested the algorithm on a series of randomised AD attack graphs with a comparison against a brute-force algorithm we also proposed in this paper, and showed that the double oracle algorithm is capable of maintaining a fast computational speed under various network environments. By comparing with currently the fastest solutions proposed in [11] that are applicable to our problem, we showed that the double oracle-based algorithm also has good scalability. With an evaluation of GoodHound weakest links, we argued that attack paths giving the attacker a good chance of success are not necessarily those having the smallest number of hops. We believe the double oracle algorithm we proposed in this paper can provide better guidance that targets the elimination of optimal attack paths. However, an increase in the defender's budget can cause the run time of the double oracle algorithm to increase

exponentially, and the algorithm's optimality in eliminating paths is vulnerable to unblockable paths, which is a limitation shared by all optimal solutions. Nevertheless, the double oracle can provide effective security-hardening measures at a fast speed in general cases.

ACKNOWLEDGMENTS

This work was supported with supercomputing resources provided by the Phoenix HPC service at the University of Adelaide. Hung Nguyen is partially supported by the Australian Research Council NISDRG grant NI210100139.

REFERENCES

[1] Mustafa Abdallah, Parinaz Naghizadeh, Ashish R Hota, Timothy Cason, Saurabh Bagchi, and Shreyas Sundaram. 2020. Behavioral and game-theoretic security investments in interdependent systems modeled by attack graphs. *IEEE Transactions on Control of Network Systems* 7, 4 (2020), 1585–1596.

[2] Amotz Bar-Noy, Samir Khuller, and Baruch Schieber. 1998. *The complexity of finding most vital arcs and nodes.* Technical Report.

[3] Daniel Borbor, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. 2019. Optimizing the network diversity to improve the resilience of networks against unknown attacks. *Computer Communications* 145 (2019), 96–112. https://doi.org/10.1016/j.comcom.2019.06.004

[4] Lucas Bouillot and Emmanuel Gras. 2014. Chemins de contrôle en environnement Active Directory. (2014). https://www.sstic.org/2014/presentation/chemins_de_controle_active_directory/

[5] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. 2007. Optimal Security Hardening Using Multi-Objective Optimization on Attack Tree Models of Networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '07)*. Association for Computing Machinery, New York, NY, USA, 204–213. https://doi.org/10.1145/1315245.1315272

[6] John Dunagan, Alice X Zheng, and Daniel R Simon. 2009. 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*. 305–320.

[7] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, Christopher Kiekintveld, and Michal Pěchouček. 2019. Hardening networks against strategic attackers using attack graph games. *Computers & Security* 87 (2019), 101578.

[8] Diksha Goel, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. 2023. Evolving Reinforcement Learning Environment to Minimize Learner's Achievable Reward: An Application on Hardening Active Directory Systems. *arXiv preprint arXiv:2304.03998* (2023).

[9] Diksha Goel, Max Hector Ward-Graham, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. 2022. Defending Active Directory by Combining Neural Network Based Dynamic Program and Evolutionary Diversity Optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 1191–1199. https://doi.org/10.1145/3512290.3528729

[10] Mingyu Guo, Jialiang Li, Aneta Neumann, Frank Neumann, and Hung Nguyen. 2022. Practical fixed-parameter algorithms for defending active directory style attack graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9360–9367.

[11] Mingyu Guo, Max Ward, Aneta Neumann, Frank Neumann, and Hung Nguyen. 2022. Scalable Edge Blocking Algorithms for Defending Active Directory Style Attack Graphs. *arXiv preprint arXiv:2212.04326* (2022).

[12] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. 2011. A double oracle algorithm for zero-sum security games on graphs. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. 327–334.

[13] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. 2020. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review* 35 (2020), 100219.

[14] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. 2003. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 536–543.

[15] Andi Morris. [n. d.]. GoodHound. https://github.com/idnahacks/GoodHound. Accessed: 2022-12-14.

[16] Thanh H Nguyen, Mason Wright, Michael P Wellman, and Satinder Baveja. 2017. Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. In *Proceedings of the 2017 Workshop on Moving Target Defense*. 87–97.

[17] Steven Noel and Sushil Jajodia. 2014. Metrics suite for network attack graph analytics. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. 5–8.

[18] Josh Payne, Karan Budhraja, and Ashish Kundu. 2019. How secure is your iot network?. In *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, 181–188.

[19] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. 2012. Dynamic Security Risk Management Using Bayesian Attack Graphs. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (2012), 61–74. https://doi.org/10.1109/TDSC.2011.34

[20] Andy Robbins. [n. d.]. The Attack Path Management Manifesto. https://posts.specterops.io/the-attack-path-management-manifesto-3a3b117f5e5. Accessed: 2022-12-14.

[21] Andy Robbins. 2021. BloodHound versus Ransomware: A Defenders Guide. https://posts.specterops.io/bloodhound-versus-ransomware-a-defenders-guide-28147dedb73b

[22] Andy Robbins, Rohan Vazarkar, and Will Schroeder. [n. d.]. BloodHound: Six Degrees of Domain Admin. https://bloodhound.readthedocs.io/en/latest/index.html. Accessed: 2022-12-14.

[23] Reginald E. Sawilla and Xinming Ou. 2008. Identifying Critical Attack Assets in Dependency Attack Graphs. In *Computer Security - ESORICS 2008*, Sushil Jajodia and Javier Lopez (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 18–34.

[24] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. 2002. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 273–284.

[25] Jin Y Yen. 1971. Finding the k shortest loopless paths in a network. *management Science* 17, 11 (1971), 712–716.

[26] Kengo Zenitani. 2022. A multi-objective cost–benefit optimization algorithm for network hardening. *International Journal of Information Security* (2022), 1–20.