Near Optimal Strategies for Honeypots Placement in Dynamic and Large Active Directory Networks

Extended Abstract

Huy Q. Ngo
The University of Adelaide
Adelaide, Australia
quanghuy.ngo@adelaide.edu.au

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

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

ABSTRACT

Active Directory (AD) is the default security management system for Windows domain networks and is the target of many recent cyber attacks. We study a Stackelberg game between an attacker and a defender on large Active Directory (AD) attack graphs, where the defender employs a set of honeypots to stop the attacker from reaching high value targets. Contrary to existing works that focus on small and static attack graphs, AD graphs typically contain hundreds of thousands of nodes/edges and constantly change over time. We show that the optimal honeypot placement problem is NP-hard even for static graphs and develop a tree decomposition method to derive an optimal deployment strategy and a mixed-integer programming (MIP) formulation to scale to large graphs. We observed that the optimal blocking plan for static graphs performs poorly for dynamic graphs. To handle dynamic graphs, we re-design the mixed-integer programming formulation by combining m MIP (dyMIP(m)) instances. We prove a performance lower-bound on the optimal blocking strategy for dynamic graphs and show that our dyMIP(m) algorithm produces near optimal results.

KEYWORDS

Active Directory, Attack Graph, Network Security, Honeypot Placement, Stackelberg Game

ACM Reference Format:
Huy Q. Ngo, Mingyu Guo, and Hung Nguyen. 2023. Near Optimal Strategies for Honeypots Placement in Dynamic and Large Active Directory Networks: Extended Abstract. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION

Microsoft Active Directories (AD) are popular directory services for identity and access management and are deployed at most enterprises. Due to their popularity, AD systems have been been a major targets for attackers over the last decade. In 2021, Microsoft reported more than 25.6 billion brute force attacks on their AD accounts [11]. In these attacks, the attacker first builds an attack graph of the targeted AD system where nodes are user accounts, computers, security groups, etc. Each edge in the AD attack graph represents an existing access/exploit that the attacker can use to move from node to node. The attacker then uses the attack graph to escalate themselves from low privilege nodes to higher privilege nodes (ex. Account A $\xrightarrow{\text{AdminTo}}$ Computer B $\xrightarrow{\text{HasSession}}$ Account C) [1]. Tools for generating the AD attack graphs are widely available, such as BloodHound [10]. Defending AD systems is very challenging as AD systems are large, complex, and continuously evolve over time.

Active defense with honeypots is not new. Plenty of work have investigate the honeypot allocation problem [2–4, 8, 9]. However, the problem of placing honeypots in an AD attack graph represents two unique challenges that have not been studied thus far. The first challenge is the scale of the graphs. An AD attack graph typically consists of thousands of nodes and hundreds of thousands of edges with millions attack paths, even for a small/medium organization. The second challenge is to develop a decoy solution that remains effective even when the graph randomly changes. Defending the AD attack graph has been studied in the previous literature [1, 5–7], but none of them consider the fact that AD attack graphs are very dynamic. One of the major sources of changes in the AD graphs are users' activities. In an AD attack graph, these dynamics are represented by a special type of edges called HasSession edges [10]. HasSession edges are added to the graph when user signs on to a computer and has their credential stored in the computer memory. HasSession edge stay online until being removed from the graph when the user signs off from the computer after a period of time.

In this study, we contribute a new method for defending AD by using active defense with honeypots. We show that our honeypot placement problem in AD graphs is NP-hard even for static graphs. Then, we provide a dynamic program based on tree decomposition to optimally solve the problem and a mixed integer program (staticMIP) formulation to scale the solution to large graphs. Furthermore, we extend our study to include the honeypot placement problem in dynamic graphs, which has not been previously studied in the literature.

2 MODEL DESCRIPTION

We consider a Stackelberg game between an attacker and a defender on a directed AD attack graph $G = (V, E)$. There is a set $S \subseteq V$ of entry nodes, and the attacker can enter the graph via one of $s = |S|$ entry nodes. The attacker tries to reach a destination node called Domain Admin (DA) via shortest paths only. From an entry node, when there are multiple shortest paths, we assume the attacker will randomly draw one of the shortest paths. There is a fixed set of *blockable* nodes $N_b \subseteq V$, the defender's task is to pick b nodes in N_b to allocate honeypots in order to intercept as many of the attacker's *shortest attack paths* as possible. The attacker cannot differentiate a normal node from a honeypot. Furthermore, if the attacker stumbles into a honeypot, the attack campaign fails.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

Static	b = 10			b = 20		
	Greedy	**staticMIP**	**DP**	**Greedy**	**staticMIP**	**DP**
*R*4000	0.4009 (0.014s)	0.4009 (0.140s)	0.4009 (0.189s)	0.2605 (0.020s)	0.2605 (0.102s)	0.2605 (0.713s)
*ADS*025	0.5889 (0.128s)	**0.5877** (0.257s)	-	0.3437 (0.195s)	**0.3418** (0.190s)	-
*ADS*10	0.5731 (0.220s)	0.5731 (0.226s)	-	0.3316 (0.380s)	**0.3315** (0.276s)	-

Dynamic	**staticMIP**	**dyMIP(1)**	**dyMIP(10)**	**dyMIP(50)**	**dyMIP(100)**
*DYADS*025	0.7196 (0.190s)	0.6994 (649s)	0.6865 (726s)	0.6862 (1271s)	0.6862 (2214s)

Table 1: Experimental result of Static graph and Dynamic graph scenario. *DYADS*025 is the dynamic version of *ADS*025. For Dynamic graph, we set $b = 20$

Initially, the AD graph is filled with "open paths" which are paths that can be used by the attacker to access the DA without any intervention. The defender's task is to reduce the number of "open paths". Let's denote by y_i the total number of open paths from entry node i to DA and y_i^B as the remaining open paths after the defender applies a blocking plan B. The attacker's success probability is then $\frac{y_i^B}{y_i}$. The defender's task is to strategically place the honeypots so that the attacker's success probability (measured by the fraction of shortest paths that are not covered by at least one honeypot) is minimized. As there are s entry nodes, the expected success probability can be obtained by averaging over all entry nodes. Given a static graph G, our optimization problem is formally defined as:

$$\min_{B \subset N_b, |B| \leq b} \sum_{i=1}^{s} \frac{y_i^B}{y_i \cdot s} \tag{1}$$

In real AD networks, the graph G changes constantly due to users' activities. We consider dynamic graphs with on/off HasSession edges (nodes remain static). We model the dynamic graph process as follows. There is a subset of HasSession edges $H \subseteq E$ where each edge is turned on and off randomly. We denote the set of all graph instances as $G_s = \{g_1, g_2, g_3, ..., g_m\}$, where $m = |G_s|$. A snapshot/realisation of the dynamic graph $g_t = (V, E_t)$ can be obtained by simulating whether each edge in H is on or off. We assume that each HasSession edge is on/off independently of others with a fixed probability. In the dynamic setting, giving a defensive budget of b, the defender's problem is to allocate honeypots to limit the attacker's clean paths on every possible snapshots/realisations of the attack graphs. Let's denote by $y_{i,g}$ the total number of open paths from entry node i to DA in snapshot g and $y_{i,g}^B$ as the remaining open paths after the defender applies a blocking plan B to snapshot g. The problem in dynamic graph is defined as:

$$\min_{B \subset N_b, |B| \leq b} \sum_{g \in G_s} \sum_{i=1}^{s_g} \frac{y_{i,g}^B}{y_{i,g} \cdot s_g} \tag{2}$$

Theorem 2.1. *Let L be the maximum shortest path length from any entry node to DA. The static version of the optimal honeypot placement problem (i.e., Expression (1)) is NP-hard when $L \geq 7$.*

3 MAIN RESULT

Static Graph: Our first approach is Dynamic Programming based Tree Decomposition (DP). Tree Decomposition refers to techniques that convert a general graph to a tree. The overall idea of using the tree decomposition technique for our problem is to convert our AD graph to a tree on which we could apply our Dynamic Program. We used the security level-based vertex elimination algorithm to generate tree decomposition as shown in Guo et al. [6]. In our dynamic programming implementation, the information held by each *context vertex* is the number of paths from the node itself to DA. We consider each tree node as a sub-problem where the defender decides whether to allocate a honeypot at the current node or not, given that there is a remaining budget b'. The DP algorithm guarantees optimal solutions and is efficient when the graph is close to a tree. In the second approach, we solve the problem via Mixed-Integer Programming (staticMIP). The staticMIP formulation is based on the observation that the number of paths from an arbitrary node to the target (DA) on the **all-shortest path** graph can be obtained by summing the numbers of paths to DA from all of its successors. Let $n(i)$ be the set of successors of node i. This can be represented as: $y_i = \sum_{j \in n(i)} y_j$. If we want to allocate a honeypot on node i, then y_i is reset to 0. Then, we have our blocking constraint as follows: $y_i = (1 - B_i) \sum_{j \in n(i)} y_j$, where B_i is the budget spent on node i and B_i is binary. We also have the budget constraint: $\sum_{i \in N_b} B_i \leq b$, where b is the allowed budget. The blocking constraint is nonlinear. Our complete formulation includes the linearization step for these constraints. We conducted experiments on synthetic AD graphs generated by DBCreator [1]($R4000$), and Adsimulator [2]($ADS025$ and $ADS10$). While DP can guarantee an optimal solution, it does not scale well on a large graph. On the other hand, staticMIP can scale very well on a large graph.

Dynamic Graph: We can repurpose our earlier MIP for static graphs to handle the task of jointly optimizing for $|m|$ sample graphs (dyMIP(m)). All we need to do is merge the constraints on individual graph instances by putting $|m|$ sets of constraints into the model and replace the objective by the sum over individual objectives. The blocking constraint in dynamic graph becomes $y_{i,g} = (1 - B_i) \sum_{j \in n(i,g)} y_{j,g}$, where $n(i, g)$ is the successor of node i in snapshot g. The linear program dyMIP(m) requires $O(m \cdot n)$ variables, where n is the number of nodes in a graph $G \in X$. However, one issue with dyMIP(m) is that it is computationally difficult to solve when m gets large. In a real AD graph, there could be a large number of possible snapshots. To deal with this, we split G_s into j equally-sized batches, each batch has t graphs (i.e., $t \cdot j = |G_s|$). We then run dyMIP(t) on every partition and produce a blocking plan for each partition. We use "majority voting" to come up with a blocking plan (finding the b most voted nodes to place honeypots). The lower bound is calculated to be 0.6708 ± 0.0059 for graph DYADS025. The result shows that staticMIP performs poorly in dynamic graphs, while dyMIP can produce a close-to-optimal result.

Future Work: For dynamic graphs, we jointly optimize for m sample graph snapshots/realisations to derive defence. For future work, instead of randomly sampling, we aim to identify m "representative" snapshots (i.e., via K-means clustering).

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

ACKNOWLEDGMENTS

This work was supported with supercomputing resources provided by the Phoenix HPC service at the University of Adelaide. Hung Nguyen is partly supported by ARC NISDRG Grant NI210100139 and NGTF-Cyber Grant ID10614.

REFERENCES

[1] 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.

[2] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. 2015. Approximate solutions for attack graph games with imperfect information. In *International Conference on Decision and Game Theory for Security*. Springer, 228–249.

[3] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. 2015. Optimal network security hardening using attack graph games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[4] 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.

[5] 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*. 1191–1199.

[6] 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.

[7] Mingyu Guo, Max Ward, Aneta Neumann, Frank Neumann, and Hung Nguyen. 2023. Scalable Edge Blocking Algorithms for Defending Active Directory Style Attack Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence* (2023).

[8] Ondrej Lukas and Sebastian Garcia. 2021. Deep Generative Models to Extend Active Directory Graphs with Honeypot Users. *arXiv preprint arXiv:2109.06180* (2021).

[9] Stephanie Milani, Weiran Shen, Kevin S Chan, Sridhar Venkatesan, Nandi O Leslie, Charles Kamhoua, and Fei Fang. 2020. Harnessing the power of deception in attack graph-based security games. In *International Conference on Decision and Game Theory for Security*. Springer, 147–167.

[10] Andy Robbins. 2023. "Bloodhound: Six Degrees of domain admin. https://github.com/BloodHoundAD/BloodHound. Accessed: 2022-08-02.

[11] David Weston. 2022. New security features for Windows 11 will help protect hybrid work. https://www.microsoft.com/en-us/security/blog/2022/04/05/new-security-features-for-windows-11-will-help-protect-hybrid-work/.