Graph Coarsening for Path Finding in Cybersecurity Graphs

Emilie Hogan, John R. Johnson, Mahantesh Halappanavar
Pacific Northwest National Laboratory
902 Battelle Blvd
Richland, WA 99352
{Emilie.Hogan, John.Johnson, Mahantesh.Halappanavar}@pnnl.gov

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms, Network problems, Path and circuit problems*

ABSTRACT

A network hacking attack in which hackers repeatedly steal password hashes and move through a computer network with the goal of reaching a computer with high level administrative privileges is known as a *pass-the-hash* attack. In this paper we apply *graph coarsening* on graphs obtained from computer network data for the purpose of (a) detecting hackers using this attack and (b) assessing the risk level of the network's current state. We repeatedly contract edges (obtaining a *graph minor*), which preserves the existence of paths in the graph, and take powers of the adjacency matrix to count the paths. This allows us to detect the existence of paths as well as find paths that have high risk of being exploited by adversaries.

1 Introduction

Pass-the-hash is a common advanced persistent threat technique that takes advantage of a feature in Windows security that is designed to allow single sign-on. When a user logs into a Microsoft Windows system, their password is passed through a Hash function (it is *hashed*) and this hashed value is authorized against the hash value that was stored when you created the password. If a hacker has access to the hashed password, they can bypass the conversion of the cleartext password to the hashed value and just use the hashed value to gain access to other systems. Once they have a privileged account, e.g. Administrator, they can obtain access to additional hashed passwords and move through the network continually escalating privileges.

A cybersecurity network can be modeled formally as a graph, where the nodes are machines on the network and the edges are communications between machines. In principle, in a physically connected network, any machine can communicate with any other. However, execution of a pass-the-hash attack poses additional constraints; namely to move laterally through the network, the credential in the credential store of the machine initiating the login must match a local administrator credential on the machine being logged into. We define the *reachability graph* as the subgraph of the enterprise network topology created by starting with all nodes that have high value credentials (e.g. domain controller) and recursively adding edges to machines that meet these criteria. Figure 1 provides an example where machines are represented by rectangles. For simplicity and without loss of generality, it assumed that there is one local administrator (LA) and one value in the credential store (CS) which are logically distinct identities (in actuality there are multiple credentials, including local administrators). The figure shows all paths along the reachability graph to a "high value" credential represented by the number 9.

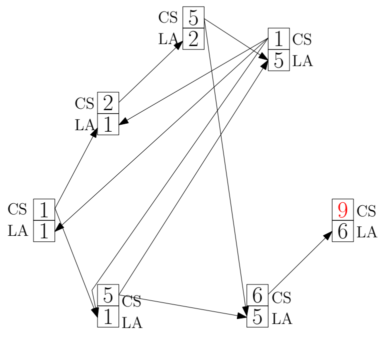

Figure 1: Example reachability graph

In real-world networks, this is a highly dynamic model as every time someone logs into a machine a new credential is stored in the credential store and they expire after a certain time-to-live.

Our eventual goal is to use graph theory and matrix algorithms to answer two questions:

- (*detection*) has somebody tried to use pass-the-hash in some past time window, and
- (*risk level*) how vulnerable is the network to a pass-the-hash attack at the moment?

In order to answer these questions we must be able to detect paths in a graph representation of the computer network. In this paper we describe the coarsening method via graph minors that we have developed to count walks in a cybersecurity graph. We also mention a way to use the symbolic adjacency matrix of a graph to find paths. We note that others have used techniques from graph theory to tackle problems in cybersecurity [5, 6]. However, in these papers the authors use different graph models and are asking different questions.

In Section 2 we give the two graph representations, one for each of the questions above, and discuss current graph and matrix sparsification techniques. We mention the limitations that these sparsification techniques have in being applied to solve this problem. Then, in Section 3 we describe our methods for compressing (coarsening) the graph in order to count the number of paths, as well as a technique to keep track of the actual paths themselves. Next, we discuss the performance of this technique when applied to various classes of graphs in Section 4. Finally we conclude in Section 5 with some future directions for this work.

2 Background

2.1 Data and Network Model

We obtained real-world enterprise network traffic data from a seven month period on the Pacific Northwest National Laboratory network in the form of Windows event logs. Each client on the network logs its events, which are then collected on an enterprise server. The event logs contain information about processes, behaviors, and inter workings of the workstations and servers, and are stored in a somewhat unstructured and non-uniform text block. Members of our team were able to normalize the data and find the common components of each event including source IP, host IP, timestamp, event ID, event log type, hostname, user, domain, and logon type. We take these new normalized events and create a graph to represent the network traffic during a given period of time.

Our first model is a graph, $G_D = (V_D, E_D)$, in which the vertices are the IPs in the network (all IPs that show up in the event logs as either source or host IP), and there is an edge $\langle u, v\rangle \in E$ if there is an event log in which Source IP $= u$ and Host IP $= v$. We will refer to this graph as the *detection graph* since this model will aid in solving the detection problem. We aim to detect paths which begin outside the network and terminate at an IP containing a high level credential. These paths can be labeled as "at risk", and flagged for further exploration by a network administrator.

In order to solve the risk level problem we must have a slightly different graph, $G_R = (V_R, E_R)$, the aforementioned *reachibility graph*. Again, the vertices are all IPs in the network, so $V_R = V_D = V$. However, we must include many more edges to determine a level of risk. Given the credential currently held by the adversary, they may make a jump to any other computer that has this credential (with local administrator privileges) stored on it. Therefore, we must infer a credential store from the event logs. This will be a function, C, from vertices (IP addresses) to subsets of users. If there is an event with Source IP $= u$, Host IP $= v$, and User $= X$ then we may infer that $X \in C(u)$ and $X \in C(v)$. Once we have the credential store map, we create an edge $\langle u, v\rangle \in E_R$ if $C(u) \cap C(v) \neq \emptyset$. As we mentioned in Section 1, credentials have a time-to-live, which allows them to expire on a particular IP. So the credential store, and therefore the edges, will be dynamically changing.

The detection graph for one day of our real-world data contained 5,899 IP addresses and 34,234,966 $\langle \text{source}, \text{host}\rangle$ entries, but many edges are repeated (same $\langle \text{source}, \text{host}\rangle$ with different timestep, user, etc.). After collapsing parallel edges we were left with 23,377. Additionally, we must include IP addresses which contact the network from the outside. This can add on the order of millions more IP addresses. Because of this it may not be feasible to perform our path search on the full graph. We were therefore motivated to consider graph sparsification and approximate matrix operation techniques.

2.2 Sparsification Techniques

In graph sparsification, researchers typically attempt to approximate a dense graph with a sparse graph which approximates, for example, every cut [1] or every eigenvalue [8] to some small multiplicative constant. A typical method for graph sparsification is random sampling of edges given some probability distribution. The distribution would depend on the properties that one wants to approximate. However, because this method removes edges it is highly likely that it will destroy many paths. It's true that if the graph starts out connected and remains connected after sparsification we still have at least one path between all pairs of vertices. However, we may not have a good approximation to the shortest path or the number of paths that exist between two vertices, for example.

One may also consider matrix sparsification. Instead of sampling edges of the graph we sample entries of the adjacency matrix (recall that an adjacency matrix has rows and columns indexed by vertices, $a_{ij} = 1$ if $\langle i, j\rangle \in E$, and 0 otherwise). In [4] Drineas, Kannan, and Mahoney describe two algorithms for matrix sparsification that allow one to approximate the product of two matrices, i.e., given matrices A and B, sample from them to get C and R such that CR is a good approximation for AB. Getting a good approximation for the product of matrices is a good first step to counting paths since one can count walks of length k by raising the adjacency matrix of a graph to the k^{th} power. In the first method, C is created by sampling columns from A, and we sample rows from B to create R. This provides a good approximation for AB, however it does not allow us to approximate odd powers of an adjacency matrix. The second method creates both C and R by changing some entries of A and B to zero based on a probability distribution. This leads to the same problem as in graph sparsification, it destroys paths.

3 Our Methods

3.1 Preserving Paths Using Graph Minors

As mentioned in Section 2.2, current graph and matrix sparsification techniques focus on removing edges (or matrix entries) based on some probability distribution. This process can easily destroy the existence of some paths in the graph. Though it may speed up certain graph algorithms, it removes the one property of the graph that we are interested in. Therefore, we sought to make the graph smaller in some sense, but not destroy the existence of paths. We do this by coarsening, through the use of graph minors. The idea of graph coarsening is not new. It has been extensively used in multilevel solvers for example (e.g., see [2, 7]). However, we believe that a coarsening method like this has not been used to aid path detection in addressing Advanced Persistent Threat attacks such as the pass-the-hash attack.

Graph minors are a topic covered in many introductory graph theory textbooks (e.g., [3]) so we give only the basic definition here. Given an undirected graph, $G = (V, E)$, and a pair of vertices $u, v \in V$ so that $e = \langle u, v\rangle \in E$, we create a new graph $G/e = (V', E')$. Remove both u and v from V and add in a new vertex, uv, so $V' = (V \smallsetminus \{u, v\}) \cup \{uv\}$.

Then remove all edges that involve u or v, and connect uv to all of the former neighbors of u and v. The new edge set, E', is given by

$$E' = (E \smallsetminus \{\langle x,u\rangle, \langle x,v\rangle : x \in V\}) \cup \{\langle x,uv\rangle : \langle x,u\rangle \in E \text{ or } \langle x,v\rangle \in E, \text{ and } x \neq u,v\}.$$

The definition for directed graphs is nearly identical, but we must respect the direction of the edges involving u and v. See Figure 2 for an example of a single graph minor step (i.e., a single edge contraction).

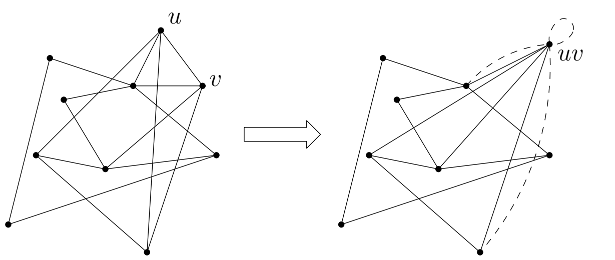

Figure 2: Example graph minor. If loops and multi-edges are allowed then we include the dashed edges and self loop.

It's clear that, at least in undirected graphs, performing an edge contraction preserves paths in the graph. Consider an $a \rightsquigarrow b$ path of length $k+1$, $P_{ab} = [a, v_1, v_2, \ldots, v_k, b]$, and contract an edge along that path, $e_i = \langle v_i, v_{i+1}\rangle$. Then there is a nearly identical $a \rightsquigarrow b$ path of length k in G/e_i,

$$P'_{ab} = [a, v_1, v_2, \ldots, v_i v_{i+1}, \ldots, v_k, b].$$

This is exactly P_{ab} where $[v_i, v_{i+1}]$ has been replaced by the new vertex $v_i v_{i+1}$. In addition, if there is no $a \rightsquigarrow b$ path in G then there can be no $a \rightsquigarrow b$ path in G/e, for any $e \in E(G)$. Note, however, that the *number* of paths will almost surely change.

In many graph coarsening and partitioning methods the goal is to minimize some cut metric in the graph [2]. Vertices are partitioned into disjoint groups and each group is contracted into a single vertex in the same way that we described a single edge being contracted. The way in which the vertices are partitioned depends on the end goal of the partitioning. We take a slightly different approach by contracting a single edge at a time based on the degrees of the endpoints. The hope is to affect as few paths as possible. Currently we choose an edge which minimizes the sum of the degrees of the endpoints, but we realize there may be other ways of choosing the edge, e.g., number of common neighbors of the endpoints or betweenness centrality of the edge.

3.2 Counting and Finding Walks

It's well known that if A is the adjacency matrix of a (directed or undirected) graph then the (i,j) entry of A^k is the number of walks (paths in which vertices and edges may be repeated) from i to j in G. So if we take the sum of the powers of A,

$$W_k(G) = \sum_{\ell=1}^{k} A^\ell, \tag{1}$$

we get the number of walks from i to j of length $\leq k$. We will be using this matrix to assess the performance of the minors/coarsening method for approximate the number of walks in a graph.

We can also use the adjacency matrix to keep track of what the walks are, but we must use a symbolic adjacency matrix. Let S be a matrix in which the entries are zero or formal variables

$$s_{ij} = \begin{cases} x_{ij} & \langle i,j\rangle \in E \\ 0 & \langle i,j\rangle \notin E. \end{cases}$$

Then, entries in S^k will aid in identifying the walks and paths. For example, if we have

$$x_{1,2}^2 x_{2,1} + x_{1,2}x_{2,3}x_{3,2} + x_{1,2}x_{2,4}x_{4,2} + x_{1,3}x_{3,4}x_{4,2}$$

as the $(1,2)$ entry in S^3 then we know that there are three walks, $[1,2,1,2]$, $[1,2,3,2]$, and $[1,2,4,2]$, and one path, $[1,3,4,2]$, of length 3 from vertex 1 to vertex 2.

This method is, of course, quite memory intensive since these polynomials get large quickly. However, for finding short paths this can be feasible.

4 Early Results

Given an undirected graph, $G = (V,E)$, we perform edge contractions successively. For various values of m, the number of edges contracted, we calculate $W_\ell(G_m)$, as defined in (1), where G_m is the graph after contracting m edges. We wish to compare this to the actual number of walks, $W_\ell(G)$. However, the matrix of walks for G_m is of smaller dimension than the matrix for G so we cannot immediately do a comparison like taking the norm of their difference. We note that there are some vertices in G_m which do not have a counterpart in G, those vertices that were created as a result of contracting an edge. There are also vertices in G that are not in G_m, those that were removed when an edge was contracted. But there is a set, V_m, of vertices which occur in both G_m and G. We keep track of the association for each vertex (in original graph, result of contraction, removed from graph) as we go along, so as to keep track of V_m throughout the process.

We first calculate $W_\ell(G_m)$ and $W_\ell(G)$ to get the counts for all vertex pairs, then we restrict the rows and columns of both matrices to just those vertices in V_m, giving us $W_\ell(G_m)|_{V_m}$ and $W_\ell(G)|_{V_m}$. Since there are fewer vertices in G_m than there are in G we expect that there will be fewer walks as well, so comparing the actual number of walks may be misleading. Instead we will compare the number of walks between each pair of vertices as a percentage of the total number of walks in the graph. In other words, we compare

$$\left.\frac{W_\ell(G)}{||W_\ell(G)||_1}\right|_{V_m} \text{ and } \left.\frac{W_\ell(G_m)}{||W_\ell(G_m)||_1}\right|_{V_m},$$

where $||\cdot||_1$ is the L_1 norm of the matrix (sum of the absolute value of the entries). Let

$$D_{\ell,m} := \left.\frac{W_\ell(G)}{||W_\ell(G)||_1}\right|_{V_m} - \left.\frac{W_\ell(G_m)}{||W_\ell(G_m)||_1}\right|_{V_m}.$$

The (i,j) entry of $D_{\ell,m}$ is the number of walks of length $\leq \ell$ from i to j in G as a percentage of the total number of walks of length $\leq \ell$ minus the same quantity for G_m. Our hope is that the average of the absolute values of the entries in $D_{\ell,m}$ is small meaning that these percentages are typically quite similar.

We have done this for both random graphs (Erdős-Rényi and scale-free) and subgraphs of our network graphs. The results from random graphs with $\ell = 20$ and $|V| = 1000$ are shown in Figure 3. Note that our measure of error appears to grow like a polynomial for both the Erdős-Rényi and scale-free random graphs, but the error is much lower for Erdős-Rényi. In Figure 4 we show the data for subgraphs of the

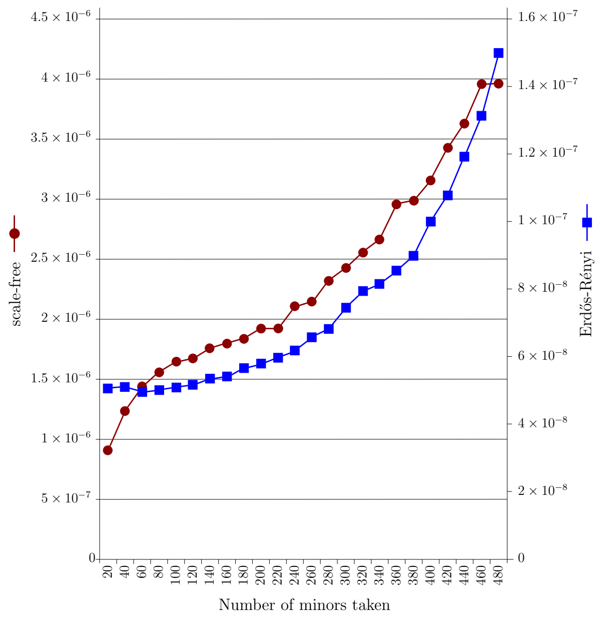

Figure 3: Average of the absolute values of the entries of $D_{20,m}$, with $m = 20, 40, \ldots, 480$, for an Erdős-Rényi ($p = 0.5$) and scale-free random graph with $|V| = 1000$.

real network data for $\ell = 10$ and various values of $|V|$. We created the detection graph, G_D, and randomly chose vertices from V_D (choose each vertex with some probability p) to be the vertex set for an induced subgraph of G_D. We did this random process multiple times with various values of p to get vertex sets of different sizes. Notice that the shape of the plots in Figure 4 most resemble that of the scale-free plot in Figure 3.

5 Conclusion

We believe that graph coarsening via minors gives us a good approximation of the number of paths in a given graph. In the future we would like to extend this research by testing on larger graphs and directed graphs. As mentioned in Section 3.1 we will also experiment with different schemes for choosing which edges to contract. In addition, we have not used any information about what vertices are contained in the metanodes created by edge contraction. We hope to look into precomputing paths on the metanodes prior to contracting and using that information to get more accurate path counts on the coarser graph.

6 Acknowledgements

We acknowledge the members of the GRADIENT team for their help on this project: Satish Chikkagoudar, Sutanay Choudhury, Peter Hui, Bill Nickless, Bryan Olsen, and Elena Peterson. We also wish to acknowledge the anonymous reviewers for their helpful comments to improve our paper.

The research described in this paper is part of the Signatures Discovery Initiative at Pacific Northwest National Laboratory. It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy.

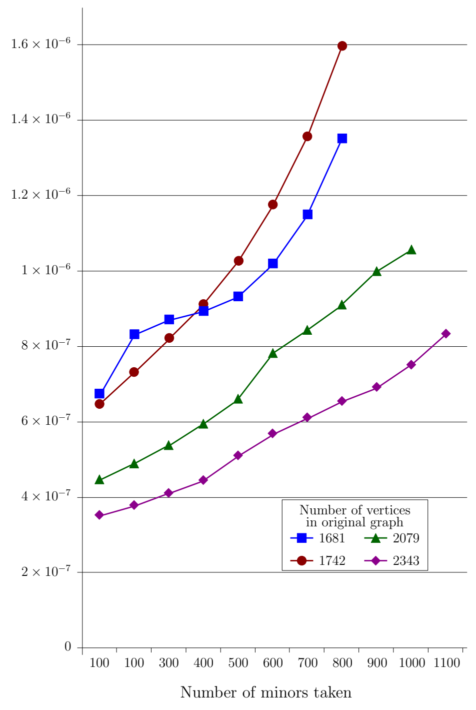

Figure 4: Average of the absolute values of the entries of $D_{10,m}$, with $m = 100, 200, \ldots, |V|/2$, for randomly chosen induced subgraphs of our cybersecurity graphs with $|V|$ values as indicated.

7 References

[1] A. A. Benczúr and D. R. Karger. Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Proc. 28th Annual ACM Symposium on Theory of Computing*, 1996.

[2] C. Chevallier and I. Safro. Comparison of coarsening schemes for multilevel graph partitioning. In T. Stützle, editor, *Learning and Intelligent Optimization*, volume 5851 of *Lecture Notes in Computer Science*, pages 191–205. Springer-Verlag, 2009.

[3] R. Diestel. *Graph Theory*. Springer-Verlag, Berlin Heidelberg, third edition, 2006.

[4] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[5] W. Eberle and L. Holder. Applying graph-based anomaly detection approaches to the discovery of insider threats. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2009.

[6] S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challenges*, pages 248–266. Kluwer Academic Publisher, 2005.

[7] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[8] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proc. 40th Annual ACM Symposium on Theory of Computing*, 2008.