A Graph Analytic Metric for Mitigating Advanced Persistent Threat

John R. Johnson and Emilie A. Hogan

Abstract— **This paper introduces a novel graph analytic metric that can be used to measure the potential vulnerability of a cyber network to specific types of attacks that use lateral movement and privilege escalation such as the well-known *Pass The Hash, (PTH)*. The metric is computed from an oriented subgraph of the underlying cyber network induced by selecting only those edges for which a given property holds between the two vertices of the edge. The metric with respect to a select node on the subgraph is defined as the likelihood that the select node is reachable from another arbitrary node in the graph. This metric can be calculated dynamically from the authorization and auditing layers during the network security authorization phase and will potentially enable predictive deterrence against attacks such as *PTH*.**

Index Terms—**cybersecurity, discrete mathematics, graph theory**

I. Introduction

There are few real-time metrics of enterprise network vulnerability, and for those that do exist it is difficult to use them meaningfully to prevent cyber attacks. Additionally, given the diversity and complexity of interactions among computers on a network, it is difficult to construct or detect signatures of network activity that distinguish between legitimate and illegitimate activity. This paper, however, describes a novel real-time metric, γ_v, (GAMMA: Graph Analytic Metric for Mitigating [advanced persistent threat] APT), that detects when a system level event (e.g., authorization) will change the mechanics of the network in a way that makes the network more vulnerable to a particular type of attack. By dynamically calculating γ_v whenever an authorization request is made, network administrators can automatically and predictively control the level of exposure to certain well documented *Advanced Persistent Threat, APT*, attacks such as *Pass-The-Hash, (PTH)* or prevent them altogether.

PTH is one of the most common and effective techniques used for illicitly obtaining privileged control over a computer network, [13][14]. The PTH vulnerability is credited to Paul Ashton in reference to his posts to the Open Source Vulnerability Database, in 1997, [2][16]. The PTH attack is launched against the authentication phase of network security. For Microsoft Windows systems, which are the most common target of PTH attacks, there are four mechanisms of authentication: LM, NTLM, NTLMv2, and Microsoft Kerberos v5, [10][17]. All of these are susceptible to PTH attack, and for all but Microsoft Kerberos, there are readily available tools available to assist in the attack, [10][14]. In each of these mechanisms, hashes of passwords are stored in memory of a machine by the Local Security Authority Subsystem Service (lsass.exe), (and in some cases they may also be stored in the Security Accounts Manager file). Local administrators on a machine have the ability to dump the hashed credential store in memory and obtain the hashed credentials. These credentials are used to log into other machines, and if one of these credentials is the local administrator on another machine, then the process can be repeated and the adversary can move laterally through the network escalating privileges. The ultimate goal of this type of attack is to reach a Domain Controller hashed password that can be used to dump the ntds.dit file on the Domain Controller, which provides access to all the passwords to the network.

A simple analysis of the process will show that there are only two conditions necessary for a PTH attack to proceed. First, the *initial* condition is that the adversary obtains administrator privilege over at least one machine on the network; this enables them to access the local hashed credential store. Secondly, there must be a *recursive* condition that allows the adversary to laterally move through the network. PTH is a remarkably simple process in that the *only* recursive condition required is that a credential in the local credential store is an administrator credential on another machine in the network. This is an example of *homogenous* process in the sense that at each step, the same condition is applied to move through the network. This process is formalized in the next section.

An enterprise cyber network can be modeled using the language of graph theory by a graph G = (V, E) where the vertices, V, are the systems on the cyber network and the set of edges, E, are connections or potential connections between systems on the network. In this context, a connection is defined as a logon event or potential logon event.

It is important to distinguish between three different aspects

May 3, 2013.

This work was funded in part by the Signature Discovery Initiative at the U.S. Department of Energy's Pacific Northwest National Laboratory (PNNL). PNNL is operated by Battelle Memorial Institute under Contract DE-ACO6-76RL01830.

J. R. Johnson is with the Pacific Northwest National Laboratory, Richland, WA 99354 USA (john.johnson@pnnl.gov).

E. A. Hogan is with the Pacific Northwest National Laboratory, Richland, WA 99354 USA (emilie.hogan@pnnl.gov).

of the underlying network that have bearing on analysis: the *structure* of the network; the *mechanics* of the network; and the *dynamics* of the network. The structure determines the physical or logical layout of the network, i.e., how things are connected together. The mechanics determine how the scientific domain—in this case cybersecurity—operates on the network, i.e., what are the rules that determine what can happen, what can't happen, and what can happen but should not. The dynamics look at how both the structure and the mechanics evolve over time. As a concrete example, the structure of an enterprise network may be the physical connection or logical layout of the machines. The mechanics could be the rules that dominate user movement through the network, i.e., to enter the network a user must first authenticate, and then to move from machine to machine within the network, the user must be authorized to access the new machine, [9][17]. The dynamics could reflect both intrinsic and extrinsic events on the node of a network, [1]. In the context of PTH, an extrinsic event could be the deposition of a credential into the local store of a machine when a user logs in; this involves an influencing action from a neighbor machine. An intrinsic event could be the expiration of a credential in the local credential store once it is time to live has expired; this change is isolated to the node and is not dependent upon a neighbor's action at the particular time step.

There has been much work on analyzing network structure using standard graph statistics such as diameter, degree distribution, max/min cut, etc., [3][11][12][15]. There has also been work in dynamic graphs to study how these structures evolve over time, [1][5][6]. The unique approach presented here is to incorporate these techniques, but also enhance the graph-based approach by introducing new graph-based mathematical formalisms to capture the *mechanics* of a network—i.e., what the network can, cannot, or should not do.

For practical purposes one can assume, without loss of generality, that in a given enterprise any system on the network is potentially accessible by any other system on the network. Thus the underlying structural network, when modeled using the language of graph theory, can be assumed to be a complete graph $G = (V, E)$ where V is the set of all systems on the network and E = V × V are all potential connections between systems. Note this may not reflect the physical topology of the network, but it can be a useful assumption for the logical topology of the network. Since communication can go in either direction, we can assume the graph is undirected (or alternatively bi-directional). For simplicity of exposition, we assume there are only two types of credentials: normal and privileged, and we will additionally assume there is only one privileged credential. These assumptions can be relaxed without impacting the fundamental results presented here.

With these preliminaries, one can define the *reachability graph* as an oriented subgraph of the fully connected graph based upon the mechanics of the authorization aspects of network security and the salient characteristics of a particular vulnerability.

Fig. 1 provides a simplified example of the reachability graph based upon PTH. The CS/LA boxes represent a single

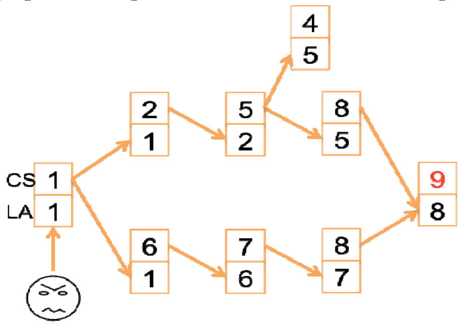

Fig. 1. Example reachability graph.

machine on an enterprise network. The "CS" value represents the value in the local Credential Store. The "LA" value represents the value of local administrator on the machine. An edge exists between two nodes if the CS value on the source node equals the LA value on the destination node. The red 9 represents the privileged credential that the adversary is after.

Once this reachability graph is constructed, a simple metric, γ_v, can be computed as the total number of vertices in the reachability graph that are on a path to the select node of interest. This provides a quantitative measure as to how vulnerable a network is to an attack such as PTH.

Using this simple metric, network administrators can automatically monitor the state of the network by recalculating, with each authorization request, whether the potential γ_v associated with granting the new authorization request would increase or decrease the vulnerability of the network. If the vulnerability exceeds a given threshold, then the request can be denied or some other action may be taken (e.g., flushing credential stores on selected machines). If the threshold is set to 0, then attacks such as PTH could effectively be prevented.

We now continue with the technical details of this method in Section II. Then, in Section III we discuss the results of our implementation and its performance. Finally we conclude and state some future work in Section IV.

II. Technical Detail

A. Formal Description

Definition: The *reachability graph*, *H,* of a graph *G* is an oriented subgraph of *G* such that the edges of *H* are chosen by a function $f: E_G \rightarrow \{1,-1, 0\}$ where e_{ij} is an edge in *H* iff $f(e_{ij}) \in \{1, -1\}$. If $f = 1$ then the edge in *H* is directed from *i* to *j*. If $f = -1$ then the edge is directed from *j* to *i*. The vertices of *H* are the set of vertices consisting of the endpoints of E_H. *f* is called the *reachability function.*

This definition is consistent with similar reachability notions such as the *reachability query*, [8].

Definition: The γ_v-*metric* or simply γ_v of a vertex, *v*, in a graph, *H*, is defined to be the number of vertices in *H* on a path ending in *v*:

$$\gamma_v = \sum_{u \in H} p(u,v) \tag{1}$$

This γ_v-*metric* is similar to the notion of *inward and outward node accessibility*, [4], but γ_v takes the specific path into consideration and is deterministic rather than probabilistic.

The γ_v-*metric* is not related to the *gamma-index* of a graph, which measures the number of observed links in a graph in proportion to the total possible edges.

Pseudocode for generating the reachability graph and calculating γ_v can be found below.

```
def generateReachability(g):
h = Graph()
for edge in g.edges:
   if nodes[edge.dst].creds[LA] in  \
   nodes[edge.src].creds
    h.edges.append(edge)
    h.nodes.append(src)
    h.nodes.append(dst)
   if nodes[edge.src].creds[LA] in  \
   nodes[edge.dst].creds
    newedge = Edge(dst, src)
    h.edges.append(newedge)
    h.nodes.append(src)
    h.nodes.append(dst)
   return h
```

```
def calculateGAMMA(h, v):
  gamma = 0
  for edge in h.edges
    if edge.dst = v
    gamma++
    nodeq.push(edge.src)
  while nodeq not empty
    node = nodeq.pop()
    for neighbor in node.in-neighbors
      if not neighbor.visited
        gamma++
        neighbor.visited = TRUE
        nodeq.push(neighbor)
  return gamma
```

B. *Practical Implementation*

The γ_v-*metric* can be used to reduce the exposure of a network to a PTH attack. There are three main components of modern network security: authentication, authorization, and auditing, [9][17]. Kerberos is a popular single sign-on security system that performs all three of these functions, [17].

In the Kerberos authentication phase, a client logs into the local machine, and he or she enters a username and a password. His/her password is passed through a one-way hash function to generate the secret key of the client. The user identification (ID) is sent in clear text to an Authentication Server (AS), which uses the client's secret key to return an encrypted Ticket Granting Ticket (TGT), which contains information about the client, a time-to-live for the ticket, and an encrypted session key.

In the authorization phase, the client sends the TGT and the ID of the requested resource to the AS. It also sends the AS the client ID and an encrypted timestamp. The AS verifies the authorization of the client, and then after a few more steps, the client is able to connect with the requested resource.

From this high level description of the process, it can be seen that the AS has all the information required to generate and maintain the reachability graph: the history of successful logins, the credentials associated with a client (which would include whether the user has administrator privileges on client, source, or machine), and it has the time-to-live for the certificates. So, if the reachability graph is retained in memory of the AS, then the AS can determine whether granting the authorization request will increase the vulnerability of the network, i.e., the γ_v value, and take action accordingly.

III. Performance Results

From a practical perspective, a metric such as the γ_v-*metric* must be able to be integrated into an operational authorization service, such as Kerberos, without significantly impacting the overall performance of the system. This section provides theoretical performance bounds for computing the *reachabilty graph* and the γ_v-*metric*, along with empirical studies to show that for most enterprises, the calculation of this metric can meet operational performance requirements.

The empirical study uses two synthetically generated graphs. With the assumption that the underlying graph, G, is fully connected, properties are applied to the nodes of G so that the generation of the reachability graphs result in two distinct, well structured, synthetic reachability graphs that are useful for benchmarking: the first is a *star graph* where the center of the star, to which all other vertices point, is the vertex, *v*, for which we will compute χ_v; the second is a single *path graph* where the vertex used for computing χ_v is the endpoint of the directed path. These graphs were chosen to highlight two extreme cases in which all the nodes of the original graph are present in the reachability graph: in the first, there are E – 1 paths all of length one; in the second, there is only one path that traverses every node. In general, the reachability graph will likely be significantly smaller than the underlying completely connected graph, and the path patterns will be between these two extremes.

A. Theoretical Performance

The computation of the *reachability graph* requires a single pass over all the edges of the graph, which requires *O(E)* time and is thus linearly scalable in the number of edges.

In the implementation given above, calculation of γ_v requires a first pass over all the edges of the reachability graph, which requires $O(E_H)$ time, (where E_H are the edges of the reachability subgraph). It then proceeds through each node expanding the horizon of nodes for which it is a destination adding only those nodes that haven't been visited before. At a course level, the worst case would be bound by $O((E_H)^2)$ since for each edge, the maximum number of potential source nodes for the edge would be $E_H - 1$ (since loops are not counted) each of which would be added to the queue. As each of these source nodes are expanded, their destination nodes are either already accounted for (as source nodes from the previous iteration) or are part of the remaining pool that is only one node pointing to the remaining $E_H - 1$ nodes, thus the overall computation performance would be $(E_H) \times (E_H - 1) = O((E_H)^2)$. This is the case of the star graph.

In the case of the single path graph, there is still the first pass over all the edges of the reachability graph to fill the initial queue (which at the end of the pass will contain one entry), this takes time $O(E_H)$. At each iteration, a single new node will be entered into the queue for a total of N_H iterations, where N_H is the number of nodes in the reachability graph. But in the single path graph, $E_H = N_H - 1$, so the performance is bound by $(E_H) \times (E_H + 1) = O((E_H)^2)$.

B. Empirical Performance

Tables 1 and 2 provide results from 1,000 independent runs computing the performance of the construction of the reachability graph and the computation of γ_v. The *order* of the graph is the number of distinct nodes in the reachability graph. An order of 5,000 was chosen to be representative of a medium sized enterprise. The statistics presented include the average, maximum, and minimum performance in units of seconds. The benchmarking was performed on a 2 GHz Intel Core i7 with 8 GB 1600 MHz DDR3 RAM running OS X Lion 10.7.1. The code that generated these results is an un-optimized Python script (not compiled) run under Python version 2.7.1.

TABLE I
PERFORMANCE OF STAR REACHABILITY GRAPH

Star	Order	Average	Max	Min
Reachability	5,000	0.26s	0.29s	0.26s
γ_v	5,000	0.0578s	0.0579s	0.0576s

TABLE II
PERFORMANCE OF SINGLE PATH REACHABILITY GRAPH

Single Path	Order	Average	Max	Min
Reachability	5,000	0.48s	0.50s	0.47s
γ_v	5,000	0.061s	0.076s	0.060s

From these results, it can be seen that for the two extreme graphs (star and single path), the time to compute the reachability graph is nearly an order of magnitude greater than the time to compute the γ_v metric. This is likely due to the overhead in constructing and storing a new graph data structure for the reachability graph. A more efficient implementation that doesn't make a separate copy of the nodes and edges would reduce the performance of this phase. Additionally, in an operational venue, the full reachability graph would rarely be computed and rather at each authorization request, the reachability graph would just be updated. It is anticipated that this would be computationally much less complex. The computation of the γ_v metric for both scenarios is in the 60-millisecond range, which would be well under the network transit time for a user to request and receive authorization to a resource, so it is reasonable to conjecture that this metric could be implemented without significant performance impact. This conjecture needs to be validated against a real network under average and peak authentication server load.

IV. CONCLUSIONS AND FUTURE WORK

This paper has shown how a simple metric, γ_v, can be used to quantify the risk of exposure to potential cybersecurity threats such as Pass-The-Hash. It has further shown that in principle it is possible to compute this metric in real time during the authorization phase of network security thus giving network administrators the ability to configure a network to minimize, or potentially eliminate exposure to these types of attacks.

This work can be extended in several ways. In a real world network, there isn't a single credential for an adversary to try to obtain; there may in fact be several and they may not have the same value to the adversary. So the metric should be extended to handle the partial order of a variety of credentials across the enterprise. This would require extending the γ_v metric to the entire graph and defining a global γ metric that aggregates across all the vertices that would, in turn, likely involve some notion of normalization; if two networks are quite different in size, there needs to be a mechanism for asserting their relative vulnerability.

At the implementation level, there is additional empirical work to study the viability of constructing the reachability graph at the AS, as the AS is not aware whether the final negotiation between the Service Server and the Client results in an accepted connection. It is not anticipated that this will significantly change the structure of the reachability graph, but it is important to understand how often this happens. It is unknown whether the behavior of enterprise networks is such that if the threshold is set too low, the network becomes unusable. Future work will include a deeper study of properties of the reachability graph and an empirical study of integrating the γ_v metric calculation with an open source version of Kerberos to benchmark how the system would perform under a more realistic load, and a deeper study of properties of the reachability graph.

Acknowledgment

J. R. Johnson and E. A. Hogan thank Bill Nickless for introducing us to the idea of developing a metric to assess the vulnerability of a network to APT attacks that use lateral movement through a network to escalate privileges. We also thank Mahantesh Halappanavar for stimulating discussions on this topic.

References

[1] A. Barratt, M. Barthelemy, and A. Vespignani, *Dynamic Processes on Complex Networks*, Cambridge University Press, Cambridge, 2008.

[2] P. Ashton (1997, April 8). NT "pass the hash" with modified smb client vulnerability. [Online] Available: http://www.securityfocus.com/bid/233/info

[3] F. Chung, and L. Lu, "Complex Graphs and Networks," Regional Conference Series in Mathematics, No. 107, Conference Board of the Mathematical Sciences, American Mathematical Society, 2006.

[4] L. d. F. Costa, "Inward and Outward Node Accessibility in Complex Networks as Revealed by Non-Linear Dynamics," [Online] Available: physics.soc-ph (arXiv:0801.1982v1).

[5] D. Eppstein, Z. Galil, and G. Italiano, "Dynamic Graph Algorithms," *Algorithms and Theoretical Computing Handbook*, M. J. Atallah, ed., CRC Press, 1999.

[6] D. Eppstein, Z. Galil, G. Italiano, and A. Nissenzweig, "Sparsification--A technique for speeding up dynamic graph algorithms." *Journal of the ACM* V. 44 No. 5, 1997.

[7] B. Ewaida, "Pass-the-hash attacks: Tools and Mitigation," Information Security Reading Room, SANS Institute, 2009.

[8] W. Fan, J. Li, X. Wang, and Y. Wu, "Query Preserving Graph Compression," SIGMOD'12, Scottsdale, AZ, 2012.

[9] J. Garman, (2003, August) *Kerberos: The Definitive Guide*, O'Reilly Media.

[10] C. Hummel, "Why Crack When You Can Pass The Hash," Information Security Reading Room, SANS Institute, 2009.

[11] M. O. Jackson, *Social and Economic Networks*, Princeton University Press, Princeton, 2008.

[12] E. D. Kolaczyk, *Statistical Analysis of Network Data: Methods and Models*, Springer Series in Statistics, Springer, 2009.

[13] R. Kraus, B. Barber, M. Borkin, and N. Alpern, *Seven Deadliest Microsoft Attacks (Seven Deadliest Attacks)*, Syngress, 2010.

[14] Mandiant "APT1 Exposing One of China's Cyber Espionage Units, Mandiant Report, Alexandria, VA 22314, 2013.

[15] M. E. J. Newman, *Networks: An Introduction*, Oxford University Press, Oxford, 2010.

[16] The Open Source Vulnerability Database. [Online] Available: http://osvdb.org/show/osvdb/83797

[17] D. Todorov, *Mechanics of User Identification and Authentication*, Auerbach Publications, Boca Raton, 2003.

[18] "Detecting the Enemy Inside the Network, How Tough is it to Deal with APTs," Special TrendLabs Primer on APTs, TrendMicro, Inc., 2012. [Online] Available: http://www.trendmicro.com/cloud-content/us/pdfs/business/white-papers/wp_apt-primer.pdf

[19] Wikipedia: The Free Encyclopedia http://en.wikipedia.org/wiki/Kerberos_(protocol)