A Graph-Based Impact Metric for Mitigating Lateral Movement Cyber Attacks

Emilie Purvine
Pacific Northwest National Laboratory
Seattle, WA
emilie.purvine@pnnl.gov

John R. Johnson
Pacific Northwest National Laboratory
Richland, WA
john.johnson@pnnl.gov

Chaomei Lo
Pacific Northwest National Laboratory
Richland, WA
c.lo@pnnl.gov

ABSTRACT

Most cyber network attacks begin with an adversary gaining a foothold within the network and proceed with lateral movement until a desired goal is achieved. The mechanism by which lateral movement occurs varies but the basic signature of hopping between hosts by exploiting vulnerabilities is the same. Because of the nature of the vulnerabilities typically exploited, lateral movement is very difficult to detect and defend against. In this paper we define a dynamic reachability graph model of the network to discover possible paths that an adversary could take using different vulnerabilities, and how those paths evolve over time. We use this reachability graph to develop dynamic machine-level and network-level impact scores. Lateral movement mitigation strategies which make use of our impact scores are also discussed, and we detail an example using a freely available data set.

Keywords

Cyber security; Graph; Impact metric

1. INTRODUCTION

Advanced Persistent Threat (APT) is a phrase used to characterize a class of sophisticated cyber attacks by highly skilled, organized adversaries operating with specific long term goals. These attacks typically have a distinctive signature that consists of entry across the network perimeter, compromise of a system internal to the network, exploitation of computer vulnerabilities resulting in lateral movement through the internal network escalating credentials and privileges, and finally exfiltration of data or information out through the network perimeter [13]. Lateral movement refers to an attackers path through the network, hopping between computers to find an intended goal. It is this phase of the APT which we study in this paper.

There are many types of vulnerabilities which can be exploited for the purpose of lateral movement. One of the most prevalent is through the network's authentication mechanism. Most enterprise networks are intrinsically vulnerable to attacks that employ lateral movement. This is a consequence of networks being configured to allow single sign on in which users log in only once to access resources on a network. Common implementations of single sign on (e.g. Kerberos [5], or Microsoft Active Directory [3]) involve a trusted third party to support authentication [12]. Beyond exploiting single sign on (which we discuss in more detail in Sections 3.2 and 3.3 as an example) there are other ways to employ lateral movement. Brute force password stealing through ARP spoofing or other tools such as `pwdump` allow attackers to discover passwords which can be used to sign in to new network systems [14]. Software vulnerabilities can also be exploited by attackers to deploy malicious code allowing them to tunnel between infected machines. The Adversarial Tactics, Techniques & Common Knowledge (ATT&CK) program at MITRE has broken down the phases of an APT, including lateral movement, with descriptions of many more possible tactics used by attackers [9].

The rest of the paper is organized as follows. In Section 2 we detail other work in the area of metrics and mitigation for lateral movement attacks. Our network model and impact metric are given in Section 3.1, including the specific example of the Pass the Hash vulnerability in Section 3.2. We discuss proposed mitigation strategies which utilize our impact metric in Section 3.3, and our experimental results are shown in Section 4.

2. RELATED WORK

Because of the fact that lateral movement is difficult to both detect and defend against, there are many studies offering strategies to mitigate the risk. One such work studies the authentication lateral movement vulnerability and form a dynamic bipartite *authentication graph* [6]. The authors compute the largest connected component of this graph as a quantitative measure of the network's vulnerability to such attacks. Mitigation strategies are proposed which lower this largest connected component size. Our work is similar in the creation of a dynamic graph, and our reachability graph, defined in Section 3.1, is related to their authentication graph. However, the dynamic metric we introduce can be more specialized to the network's mission and the individual computer vulnerabilities.

Heat-ray [4] is a tool introduced by researchers at Microsoft that can inform network administrators of potential configuration changes that mitigate a network's susceptibility to lateral movement attacks, which they refer to as *identity snowball attacks*. They construct an *attack graph* based upon observation of the network over a week long period.

SafeConfig'16, October 24, 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4566-8/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2994475.2994476

The attack graph is constructed by adding edges between nodes (representing computers, accounts, and groups) where edges represent that ability for the source-node to "control" the destination-node. Once the attack graph is constructed, combinatorial optimization and machine learning methods are used to determine possible configuration changes (i.e. edge removals) that decrease the number of nodes reachable from certain starting nodes. We also use the notion of reachability in our network impact metric, however our model is more dynamic. We are able to calculate risk in near real-time allowing for more responsive mitigation strategies.

In [10, 11] the authors also define an attack graph where here nodes represent various attack states of the network – often some combination of a machine and some of its properties – and edges represent state changes in the system caused by an action by the attacker. Edges in this attack graph are weighted either by a probability that the associated action took place, cost to the attacker, or time for the action to take place. An analyst may then inquire about what short paths – representing low cost, or high probability attacks – are available in the network thus discovering the most vulnerable areas of the network. Another usage is to understand the effect of an altered network topology on the attack graph, and thus on the vulnerability of the network. Our model differs from this traditional attack graph model in two fundamental ways. First, our vertex set will simply be the set of computers on the network. Each computer will have a set of properties, but each of these properties is not separated into its own vertex. Second, our model takes into account vertex weights allowing for non-uniform computer desirability. The compromise of a rarely used workstation should have less impact on the network than that of a large server. Rather than measure the probability or cost of different attack paths we take the approach of quantifying the impact of compromise on the network based on the value and reachability of each computer.

3. NETWORK MODEL, IMPACT METRIC, AND MITIGATION

3.1 Problem Formulation

There are a multitude of ways an enterprise network can be modeled with widely ranging complexity or detail. For the purposes here, a minimal model that captures only the essential features of a generic lateral movement vulnerability is presented. We first present a model agnostic to the particular set of vulnerabilities, and in Section 3.2 we will describe our model in the context of the example Pass the Hash vulnerability.

Consider the set of computers in the network, $C = \{c_i\}_{i=1}^{n}$. We start by assuming that each computer has a *vulnerability state* associated with it that consists of properties related to lateral movement, e.g., users with credentials stored on the machine, a list of local administrators, a set of vulnerable ports. These properties will change over time as users log on and log off, or as vulnerabilities are patched and new ones emerge, and may be gathered from various sources such as Nessus scans or authentication logs. For a computer c_i and time t the vulnerability state will be denoted by $s_{i,t}$. Let S denote the set of all possible vulnerability states.

Given the vulnerability state of two computers at time t, $s_{i,t}$ and $s_{j,t}$, we can evaluate whether or not it is possible

$s_{i,t}$	$s_{j,t}$	r
1	1	F
1	2	T
1	3	F
2	1	F
2	2	T
2	3	F
3	1	F
3	2	T
3	3	F

Table 1: An example reachability function on $S = \{1, 2, 3\}$.

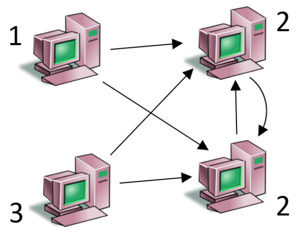


Figure 1: An reachability graph where each computer is labeled with its current vulnerability state and the reachability function is as in Table 1.

that an adversary could exploit the vulnerabilities on both computers and move from c_i to c_j. We introduce a boolean-valued *reachability function*, $r : S \times S \to \{T, F\}$, which records this ability to move through the network.

From the set of computers, vulnerability states, and the reachability function we now define the *reachability graph* as a dynamic graph, $G_t = (C, E_t)$, where the set of computers in our network serve as the vertices, and the edges change over time as vulnerability states evolve. There is an edge $\langle c_i, c_j \rangle \in E_t$ if and only if $r(s_{i,t}, s_{j,t}) = T$, i.e., if the vulnerability states of c_i and c_j at time t allow for movement from c_i to c_j. As an example, consider a system with four computers and vulnerability states $s_{1,t} = 1$, $s_{2,t} = s_{4,t} = 2$, and $s_{3,t} = 3$. Using the reachability function defined in Table 1 we obtain the reachability graph given in Figure 1.

In prior work [7] we defined a reachability metric to measure the ability of an adversary to reach each computer in a network. This local measurement, which we denoted by γ_i for a computer $c_i \in C$, is a count of how many computers have a directed path which leads to c through the reachability graph.

Definition 1. Let $G_t = (C, E_t)$ be the reachability graph on computers C at time t. For computer $c_i \in C$ its *reachability score* at time t is given by

$$\gamma_{i,t} = \sum_{\substack{c_j \in C \\ c_j \neq c_i}} p_t(c_j, c_i)$$

where $p_t(c_j, c_i) = 1$ if there is a directed path from c_j to c_i in G_t, and 0 otherwise.

Notice that the reachability graph may not be acyclic, that is, there may be a computer which can reach itself via a nontrivial directed path. But this does not affect the reachability score since we count each computer which can reach c_i only once and do not include c_i itself. The reachability score for $c \in C$ quantifies how many other computers, if compromised, could potentially lead to the compromise of c, but it does not track the overall impact of a compromise on the network, nor does it count how far away the computers are. Our aim in this paper is to use the reachability score to compute a level of impact that each computer poses toward the network in the event of a compromise anywhere, along with the total impact or risk of the network as a whole.

The first observation towards measuring the impact is that the compromise of different machines can have very different effects on the network. This leads us to consider a weight function on the set of computers which measures the overall value of the computer. This weight function should depend on the mission of the network, and on the vulnerability states. For example, a network for a financial institution will likely place highest weight on machines which store the day-to-day financial data, but a social media company like Twitter or Facebook may place their highest weight on servers containing user data. However, in both cases, machines with many vulnerabilities should also have high weight due to their ability to be compromised easily. We denote the weight on c_i at time t by $w_{i,t}$. For a given time t we can calculate the reachability and weight vectors

$$\gamma_t = \langle \gamma_{i,t} \rangle_{i=1}^n , \qquad w_t = \langle w_{i,t} \rangle_{i=1}^n ,$$

and use these to define a global network impact metric, Γ_t.

Definition 2. Let $G_t = (C, E_t)$ be the reachability graph on computers C at time t, and let γ_t and w_t be the reachability and weight vectors, respectively. We define the *impact metric*, Γ_t, at time t to be

$$\Gamma_t = \gamma_t \cdot w_t = \sum_{i=1}^n \gamma_{i,t} \cdot w_{i,t}.$$

The intuition behind this definition for a global metric is the following. Each computer can be considered to have high or low reachability, and high or low weight, in the context of the rest of the network. Any computer with both high reachability and high weight is clearly undesirable. We do not want a valuable machine to be easily accessible within the network. High reachability and low weight is also not preferred, but is less of a problem, as is low reachability and high weight. Machines with low scores in both reachability and weight have the lowest impact within the network. If another machine were to be compromised it is unlikely that a low reachability machine would be compromised as a result. But if it is, its low weight will minimize the impact to the network.

Clearly this model relies on knowing vulnerabilities ahead of time, and one might argue that the vulnerabilities we should care about are those that are as yet unknown. However, we believe that assessing the risk to known vulnerabilities is still an important problem. As security patches are released for known vulnerabilities an enterprise may not be able to deploy them all simultaneously. Doing so might bog down the network or require system reboots at inconvenient times. We propose that modeling the impact that vulnerabilities could have will aid in prioritization of security patches.

We are able to prove a simple upper bound, $\overline{\Gamma}$, for Γ_t based on the maximum weight and network size. This upper bound will be useful when we describe mitigation in Section 3.3.

Proposition 1. Let $W = \max\{w_{i,t}\}$ be the maximum weight allowed in the system. Then

$$\Gamma_t \leq (|C| - 1)|C|W =: \overline{\Gamma}.$$

PROOF. This proof is very simple based on bounds $w_{i,t} \leq W$ and $\gamma_{i,t} \leq |C| - 1$.

$$\begin{aligned} \Gamma_t &= \gamma_t \cdot w_t \\ &\leq \langle |C| - 1 \rangle_{i=1}^n \cdot \langle W \rangle_{i=1}^n \\ &= \sum_{i=1}^n (|C| - 1)W \\ &= (|C| - 1)|C|W \end{aligned}$$

□

3.2 Example: Pass the Hash

As mentioned in the introduction, one of the most pervasive vulnerabilities is the use of single sign on authentication in Windows and Kerberos. The technique that most adversaries use to take advantage of this vulnerability is Pass the Hash (PTH). The PTH vulnerability is credited to Paul Ashton in reference to a post he made in 1997 [1]. When any user logs into a computer, a hashed version of their password is stored locally in the machine's credential store. This hash will then be used to log into other network resources automatically, without prompting the user for another password, until some pre-determined *time to live* (typically on the order of days) at which point the user must re-authenticate by entering their password again. PTH takes advantage of the fact that local administrators on a machine have the ability to dump the credential store in memory and obtain the hashed credentials. These credentials can be used to log into other machines instead of using the plain text password, and if one of these credentials is local administrator on another machine, then the process can be repeated thus allowing one to move laterally and discover new credentials with higher privileges. Typically, 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.

In this section we will describe model proposed in Section 3.1 in the context of the PTH vulnerability. Since PTH vulnerability stems from the availability of cached credentials we let $s_{i,t}$ be the pair $\langle LA_{i,t}, CS_{i,t} \rangle$ where $LA_{i,t}$ is the set of local administrators on computer c_i at time t, and $CS_{i,t}$ is the set of hashed credentials in the credential store of c_i at time t. Then, given $s_{i,t}$ and $s_{j,t}$ the reachability function is defined as:

$$r(s_{i,t}, s_{j,t}) = \begin{cases} 1 & \exists\, a \in CS_{i,t} \ s.t.\ a \in LA_{j,t} \\ 0 & else. \end{cases} \tag{1}$$

In other words, if there is a credential stored on computer c_i at time t that is a local administrator on computer c_j at time t then an adversary is able to jump from c_i to c_j at time t as local administrator on the target machine. This allows for either continuation of lateral movement through the network or access to other administrator tools on the machine (e.g., making registry edits, software installation).

Given dynamic data for the credential stores and local administrators in a network and this PTH reachability function we can construct G_t and calculate the weight vector, γ_t, as the network vulnerability profiles evolve. However, in order to calculate our network impact metric, Γ_t, we need to also define a vertex weight, $w_{i,t}$. In this paper we give one example of a weight based on the contents of the credential store, but realize that there may be other factors that influence a vertex weight function specific to each network.

The mechanics of this user-based weight function can be formally modeled by first identifying each user with a credential, its security identifier, or *sid*. Let D be the set of all *sid*s on the network, and $\mathcal{L} = \{\ell_i\}_{i=1}^d$ be a totally ordered set of privileges where $\ell_i \leq \ell_j$ if and only if $i \geq j$. We reverse the levels in this way so that ℓ_1 is the highest permission, and therefore the 1^{st} choice for an adversary to steal, whereas ℓ_d is the lowest permission and the last choice for an adversary. Then the permission function, $P : D \to \mathcal{L}$, assigns a level of control to each *sid*. Multiple *sid*s can have the same network access permissions, the point is that each user has a unique ID (its *sid*) which has a particular permission level that is allowed to coincide with other users' permission levels. Typically as the adversary moves laterally through the network they will desire new credentials for which the permission, $P(sid)$, is higher in the total order $\mathcal{L}$ than the previous credential they used. Therefore, we will assign a weight to each *sid* denoting its value to attackers based on the total order $\mathcal{L}$.

Definition 3. A *sid value function* is any natural-number-valued function on the set of all *sid*s, $u : D \to \mathbb{N}$, such that if $P(sid_i) \leq P(sid_j)$ in $\mathcal{L}$ then $u(sid_i) \leq u(sid_j)$.

We will now define a specific *sid* value function, $u^\star : D \to \mathbb{N}$, with a particularly desirable property. First, let $s^i = \{sid \in D : P(sid) = \ell_i\}$ be the set of *sid*s which have permission ℓ_i. Since all $sid \in s^i$ have the same permission, we want them to have the same value under $u^\star$. With an abuse of notation let $u^\star(s^i)$ represent the value of any $sid \in s^i$ under *sid* value function $u^\star$. We then define $u^\star(s^d) = 1$ and

$$u^\star(s^i) = \prod_{j=i+1}^{d} \Big(|s^j| + 1\Big) \tag{2}$$

for $i < d$. It is not difficult to show that the value of $u^\star(s^i)$ is greater than $\sum_{j=i+1}^{d} |s^j| u^\star(s^j)$. In other words, the value of any *sid* at level ℓ_i is greater than the sum of all *sid* values for lower permission *sid*s. If we then define

$$w_{i,t} = \sum_{a \in CS_{i,t}} u^\star(P(a)) \tag{3}$$

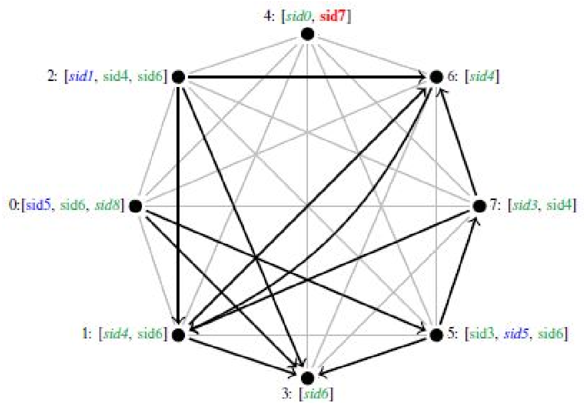
this property allows us to determine the maximum permission credential in $CS_{i,t}$ simply by knowing $w_{i,t}$. If the weight is larger than $u^\star(s^2)$, for example, then we know there is a *sid* from s^2 in the credential store of that vertex. Table 2 shows an example of $u^\star$ calculated on a partial order with $d = 3$ and 9 *sid*s. In this example, if we have $w_{i,t} = 8 \geq u^\star(s^2)$ for some computer c_i then we know there is a *sid* from s^2 present in the credential store of c_i.

Now with the definition of computer vulnerability states, S, as the set of local administrators and credential store contents, the reachability function (1), and the specific vertex

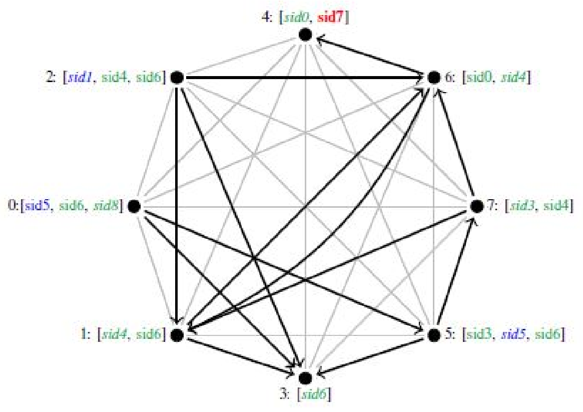
Permission	*sids*	$u^\star$
ℓ_1 (e.g., domain controller)	sid_2, sid_7	20
ℓ_2	sid_1, sid_5, sid_8	5
ℓ_3 (e.g., guest)	sid_0, sid_3, sid_4, sid_6	1

Table 2: Example set of credentials with permission levels and $u^\star$ function.

weight (3) based on the *sid* value function (2), we would be able to calculate the network impact metric Γ_t for any given dynamic credential store and local administrator data. In Section 4 we detail the results of an example dynamic Γ_t calculation for a specific real-world data set. Here we provide a small example calculation at two time points for graphs shown in Figure 2. The w_t and γ_t vectors are shown in Table 3 allowing us to calculate $\Gamma_1 = 32$ and $\Gamma_2 = 163$. The difference between $t = 1$ and $t = 2$ is the addition of the single credential sid_0 at vertex 6 which allows for the edge $\langle 6, 4 \rangle$ in the graph at $t = 2$.



(a) Reachability graph at time $t = 1$.



(b) Reachability graph at time $t = 2$.

Figure 2: Construction of reachability graph from local administrators and credential stores. Local administrators are shown in *ialics*.

3.3 Mitigation

We propose to use this global impact metric Γ_t to recommend mitigation strategies to cyber analysts or even perform automated mitigation. A threshold value, θ, can be set for

i	w_1	γ_1	w_2	γ_2
0	11	0	11	0
1	2	5	2	5
2	7	0	7	0
3	1	6	1	6
4	21	0	21	6
5	7	1	7	1
6	1	5	2	5
7	2	2	2	2

Table 3: Weight and reachability vectors for reachability graphs shown in Figure 2 using vertex weight function based on $u^\star$ given in Table 2.

a network by a system administrator. If Γ_t is ever calculated to be above this threshold then action should be taken to lower Γ_{t+1} below the threshold thus lowering the impact of an infiltration on the network. The action to be taken necessarily must depend on the vulnerability that is being modeled. What works for an authentication vulnerability like Pass the Hash will likely not make sense for a software vulnerability.

In the case of Pass the Hash our recommended mitigation action is to either (a) wipe credential stores for high value nodes, thus lowering some $w_{i,t+1}$ weights and removing edges in G_{t+1}; (b) change the reachability function (by updating some network protocols) to naturally disallow connections between certain computers; or (c) deny authorization events until credentials naturally expire, thus gradually lowering Γ_t. We recommend against using option (c) since it will be very disruptive to the network's mission as users will not be able to authenticate for a period of time. Option (a) is the most passive and straightforward to implement. Additionally, the only disruption to a user is the possible need to re-enter their password if their credential is removed from a local credential store. Finally, option (b) may be useful to consider if Γ_t routinely breaches the threshold. In this case there may be a fundamental reason that the network is constantly in a high impact state.

In the case of software or similar vulnerabilities we may use a threshold to decide when to automatically deploy patches rather than letting the user deploy the patch at a more convenient time. For example, if Γ_t gets too high it could become necessary to deploy patches on high weight, high reachability machines to lessen the impact those machines would have on the network if compromised. Of course this will impact the user, but this impact should be outweighed by the decrease in Γ_t coming from patching the vulnerable systems.

Setting a reasonable value for θ is a question that a system administrator will have to tackle based on the size of the network – number of computers, different vulnerabilities, and weights – and risk they are willing to take on. This θ will also likely need to be based on the particular network topology and policies. Though we cannot provide a heuristic for setting θ, we propose the simple use of the bound proved in Section 3.1 as a starting point. A threshold of the form $\theta = \alpha\overline{\Gamma}$, for $0 < \alpha < 1$ may provide some initial idea of how the number of alerts scales with the bound.

4. CASE STUDY: APPLICATION TO PASS-THE-HASH VULNERABILITY

4.1 Data

The test data that we are using is from an openly released data set out of Los Alamos National Laboratory (LANL) [8]. It contains user-computer authorization data for 9 months of activity at LANL. The data has been anonymized so that users are listed as a unique integer prefaced with the letter "U", and computers are listed as a unique integer prefaced with the letter "C". The files themselves consist of many individual authorization events, one on each line of the file which looks like "`time, user, computer`" indicating that at the given `time` we had an authorization request of `user` onto `computer`. If user a logs into a computer c_i at time t we add them to $CS_{i,t}$. The local administrator is assumed to be the first user to log onto a particular machine. This assumption may not be realistic, but a heuristic is necessary since local administrator information is not included in the data set. In total, the data contains 11,362 users and 22,284 computers. For this report we took a subset of the data with only 50 computers and 150 users over a single day. Additionally, we randomly assign user permission levels uniformly between 1 and 3. Since time to live values of credentials tend to be on the order of days, and these experiments are for only a single day, we will assume the time to live to be greater than a single day. In other words, we do no time-out of credentials on a computer in the results that follow.

4.2 Implementation details

We implemented the reachability graph creation and Γ calculations in C++ (gcc 5.2.0) specifically for the Pass the Hash vulnerability. Currently we read authentication log data from a text file, formatted as described in the previous section. However, we read and process the text file line by line synchronized to the computer clock using ActiveMQ-CPP 3.8.4 and Apache-ActiveMQ-5.11.1 so the infrastructure is available to connect to a stream of authentication data. Our Γ tool also provides polymorphic features allowing the user to adapt to a different set of vulnerabilities. This is done through the use of function pointers, function objects, and virtual functions in an inheritance hierarchy. Using this template, users can define their own classes or functions for the following: input data processing, reachability function, vertex value function, vulnerability states (e.g., a list of users or installed software), and permission level and weight.

In order to efficiently calculate Γ for large systems we made a couple of optimizations. First, we observe that if at time t, c_i can reach c_j, and c_j can reach c_i, then $\gamma_{i,t} = \gamma_{j,t}$. When this reciprocal relationship between c_i and c_j exists then they must be in the same *strongly connected component* of G_t (see most standard graph theory texts, e.g., [2] p. 50, for a discussion of strongly connected components). Therefore, we can speed up the calculation of γ_t by first calculating the strongly connected components of G_t which has complexity $O(|C| + |E_t|)$. Once the strongly connected components are computed we are able calculate γ_t in complexity $O(N)$, where N is the number of strongly connected components, because the set of strongly connected components forms a directed acyclic graph. Since $N \leq |C|$ we get total complexity of $O(|C| + |E_t|)$. This is compared to calculating a BFS from each vertex in the graph which would require $O(|C|(|C| + |E_t|))$. Our optimization offers a significant speedup.

Secondly, we allow the user to set a frequency for calculating Γ. As the number of users and computers grows so does the reachability graph, and at a certain point we are not able to keep up with calculating Γ after every authentication event. Therefore, we allow the user to select a frequency (e.g., every 10 seconds, 1 minute, 10 minutes) with which to calculate the network impact metric. The code is run using MPI parallelism where the credential stores and G_t are tracked and updated continuously (after every authentication event) on the main compute node, and new compute nodes are tasked to calculate Γ at the frequency requested by the user. In our experiments on subsets of the real data we were able to cope with roughly 20 authorizations per second and still calculate Γ in real time. However, in the full data set there are on average roughly 28 authorization events per second with a maximum of 1025. Therefore, this Γ calculation frequency is a necessary optimization.

4.3 Results

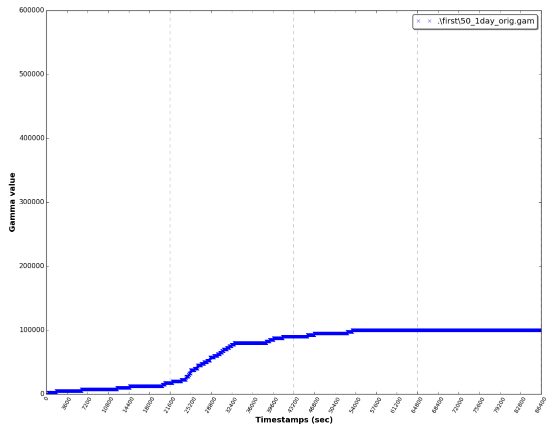
Here we show three separate evolutions of Γ_t. In the first case we run one day of authorization events from the 50 computer system. The Γ_t evolution is shown in Figure 3a. Notice that while $\max\{\Gamma_t\}$ is roughly 100,000 we let the y axis go up to 600,000 in order to easily make comparisons to the following two plots. Here we use no threshold or mitigation strategy to get a system baseline.

In the second case we artificially inject vulnerabilities into the system at $t = 32,400$ by adding three authorization events into the first data set based on inspection of the system. Just as before we do not employ any mitigation and allow Γ_t to grow unbounded. The results are shown in Figure 3b. Here we see that at $t = 32,400$ the impact metric makes a sudden jump up to roughly 450,000. This is much higher than the maximum value in the unmanipulated system.

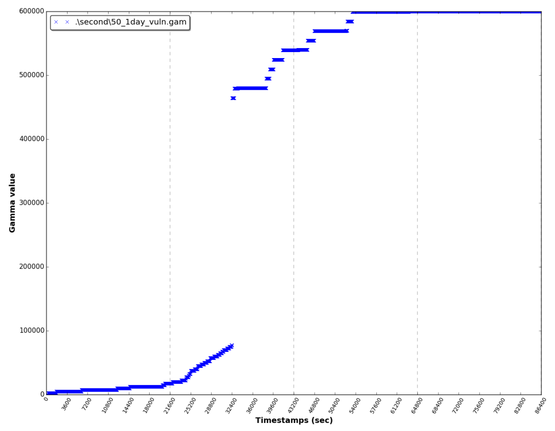
Finally, in our third evolution of Γ_t we run the second data set, with the vulnerabilities, but this time with a threshold value of $\theta = 400,000$. We mitigate using option (a) described in Section 3.3: wiping credential stores of the highest weight vertex whenever $\Gamma_t > \theta$. In Figure 3c we see again at $t = 32,400$ we have a jump in Γ_t above 400,000. However, because of the mitigation of wiping the highest weight credential store the next value for Γ_t is significantly lower. The value continues to grow as new users sign into the computers until the threshold is achieved again, driving down Γ_t for a second time.

In this three-fold example we have shown that Γ_t is able to identify a system vulnerability immediately. Additionally we see that the mitigation action of wiping credential stores can drive down the system impact score when it evolves over a set threshold.

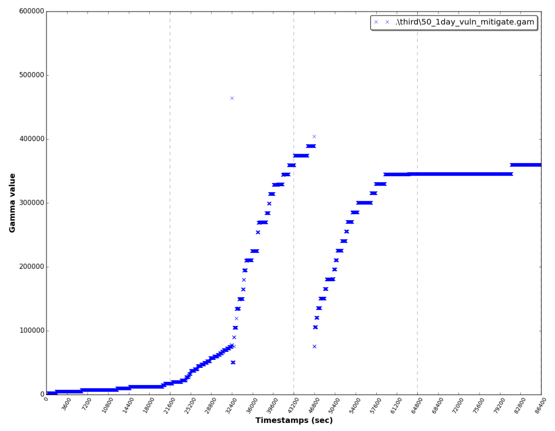
We can also show scalability on a larger subsystem with 10,000 computers. In Figure 4 we show the evolution of Γ_t over one week in this subsystem for Γ_t calculation frequency of 1 minute (blue and green) or 10 minutes (red and yellow), and for credential time to live either 1 day (blue and red) or 3 days (green and yellow). In the top figure we allow Γ_t to evolve without bound, i.e., without setting a threshold θ. Notice that the green and yellow (3 days time to live) diverge from the red and blue (1 day time to live) around the start of the first day. This is when the initial credentials



(a) Γ_t evolution for original data set.



(b) Γ_t evolution for data set with vulnerabilities and no mitigation strategy.

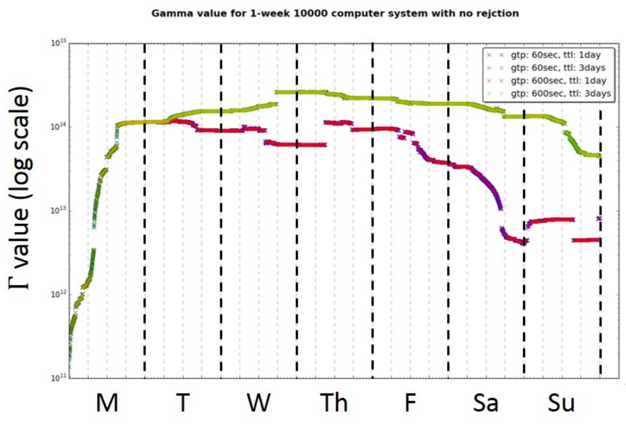


(c) Γ_t evolution for data set with vulnerabilities and using threshold $\theta = 400,000$ and credential wiping mitigation.

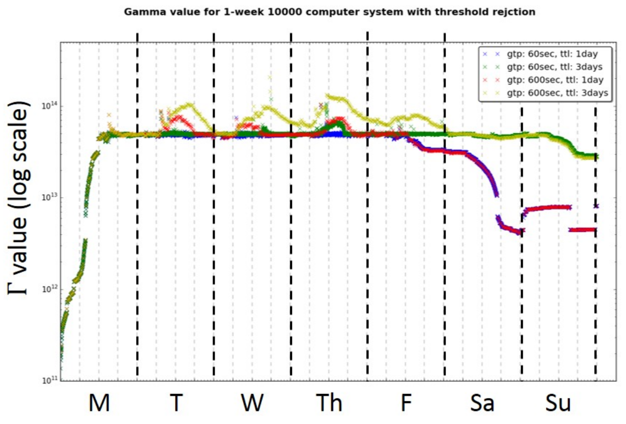
Figure 3: In all cases the x axis is t in seconds and the y axis is Γ_t.

begin to expire from machines in the red and blue cases, but stay around for two more days in the green and yellow case.

In Figure 4b we use the threshold $\theta = 5 \times 10^{13}$ and mitigation strategy of wiping the credential store of the computer with largest $w_{i,t} \cdot \gamma_{i,t}$, but leaving the local administrators. Notice that in the yellow and red cases, when we calculate Γ_t only every 10 minutes, we do have Γ_t rise above the threshold. This is because Γ_t must have become too large during the 10 minutes between calculation frequency. Since we only expire credentials off of one system (in a real implementation we would likely expire from more systems) we can only slowly make up the Γ_t overage. We do sometimes go over the threshold in the 1 minute frequency cases (blue and green) but less often.



(a) No threshold



(b) Threshold $\theta = 5 \times 10^{13}$.

Figure 4: Evolution of Γ_t over one week in a 10,000 computer system. Each color indicates a different Γ_t calculation frequency (f) and credential time to live (ttl). Blue: $f = 1$ minute, $ttl = 1$ day. Green: $f = 1$ minute, $ttl = 3$ days. Red: $f = 10$ minutes, $ttl = 1$ day. Yellow: $f = 10$ minutes, $ttl = 3$ days.

We ran all four of these examples not in real time, but instead reading authentication events as fast as possible. Each took significantly less than the full week to run, indicating that we may be able to calculate Γ_t more often than 1 or 10 minutes and still keep up with this volume of data. See Table 4 for the compute times for each experiment.

Γ_t frequency	Time to live	Compute time
1 minute	1 day	42 hr, 46 min
1 minute	3 days	40 hr, 44 min
10 minutes	1 day	7 hr, 38 min
10 minutes	3 days	6 hr, 35 min

Table 4: Compute time for four runs of 1 week, 10,000 computer system at varying Γ_t frequency and time to live. Times much less than 1 week indicate real-time processing should be achievable on a system of this size.

5. CONCLUSION

In this paper we have described a general network model and impact metric for studying cyber attacks which have a lateral movement component. The impact metric is calculated dynamically as the network evolves over time and can be used to mitigate the lateral movement vulnerability through thresholding. Though we initially describe the model in a way that is agnostic to the type of vulnerability, we additionally given the example of the authentication vulnerability Pass the Hash. Using real network data from the LANL computer network we show example calculations of our metric in the case of the original as-is data, the same system with a small vulnerability added, and the vulnerable system with mitigation strategy employed. We believe that this example shows the capability of our model to quickly identify and mitigate high impact events within the network.

6. ACKNOWLEDGEMENTS

This work has been performed under the High Performance Data Analytics program at Pacific Northwest National Laboratory.

7. REFERENCES

[1] D. J. Bayliss. *Error Correcting Codes: A Mathematical Introduction.* Chapman Hall/CRC Mathematics Series, 1997.

[2] G. Chartrand, L. Lesniak, and P. Zhang. *Graphs & Digraphs, Fifth Edition.* Chapman & Hall/CRC, 5th edition, 2010.

[3] B. Desmond, J. Richards, R. Allen, and A. Lowe-Norris. *Active Directory: Designing, Deploying, and Running Active Directory.* O'Reilly Media, Inc., 4th edition, 2008.

[4] J. Dunagan, A. X. Zheng, and D. R. Simon. Heat-ray: combating identity snowball attacks using machinelearning, combinatorial optimization and attack graphs. In J. N. Matthews and T. E. Anderson, editors, *SOSP*, pages 305–320. ACM, 2009.

[5] J. Garman. *Kerberos, The Definitive Guide.* O'Reilly, 2003.

[6] A. Hagberg, A. Kent, N. Lemons, and J. Neil. Connected components and credential hopping in authentication graphs. In *2014 International Conference on Signal-Image Technology Internet-Based Systems (SITIS).* IEEE Computer Society, Nov. 2014.

[7] J. R. Johnson and E. Hogan. A graph analytic metric for mitigating advanced persistent threat. In *IEEE*

International Conference on Intelligence and Security Informatics. IEEE Computer Society, 2013.

[8] A. D. Kent. User-computer authentication associations in time. Los Alamos National Laboratory, http://csr.lanl.gov/data/auth/, 2014.

[9] MITRE. Adversarial Tactics, Techniques & Common Knowledge (ATT&CK). https://attack.mitre.org/wiki/Main_Page, 2016.

[10] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*, NSPW '98, pages 71–79, New York, NY, USA, 1998. ACM.

[11] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 2, pages 307–321 vol.2, 2001.

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

[13] TrendMicro, Inc. Detecting the enemy inside the network, how tough is it to deal with APTs. http://www.trendmicro.com/cloud-content/us/pdfs/business/white-papers/wp_apt-primer.pdf, 2012.

[14] TrendMicro, Inc. Lateral movement: How do threat actors move deeper into your network? http://about-threats.trendmicro.com/cloud-content/us/ent-primers/pdf/tlp_lateral_movement.pdf, 2013.