Practical Attacks Against Graph-based Clustering

Yizheng Chen
School of Computer Science, College of Computing
Georgia Institute of Technology
yzchen@gatech.edu

Yacin Nadji
School of Electrical and Computer Engineering
Georgia Institute of Technology
yacin@gatech.edu

Athanasios Kountouras
School of Computer Science, College of Computing
Georgia Institute of Technology
kountouras@gatech.edu

Fabian Monrose
Department of Computer Science
University of North Carolina at Chapel Hill
fabian@cs.unc.edu

Roberto Perdisci
Department of Computer Science
University of Georgia
perdisci@cs.uga.edu

Manos Antonakakis
School of Electrical and Computer Engineering
Georgia Institute of Technology
manos@gatech.edu

Nikolaos Vasiloglou
Symantec CAML Group
nikolaos_vasiloglou@symantec.com

ABSTRACT

Graph modeling allows numerous security problems to be tackled in a general way, however, little work has been done to understand their ability to withstand adversarial attacks. We design and evaluate two novel graph attacks against a state-of-the-art network-level, graph-based detection system. Our work highlights areas in adversarial machine learning that have not yet been addressed, specifically: graph-based clustering techniques, and a *global* feature space where realistic attackers without perfect knowledge must be accounted for (by the defenders) in order to be practical. Even though less informed attackers can evade graph clustering with low cost, we show that some practical defenses are possible.

KEYWORDS

Adversarial Machine Learning; Unsupervised Learning; DGA; Network Security

1 INTRODUCTION

Network level detection systems are used widely by the community as the first line of defense against Internet threats [9, 20, 27, 29, 33, 41, 63]. These systems often represent the underlying network traffic as a graph for various reasons, but most importantly for the computational efficiency and scalability that graph techniques enable. These computational advantages, for example, enable categorical objects (like domain names and IP addresses) to be transformed into feature vectors in a multi-dimensional euclidean space. This allows supervised and unsupervised learning to take place with greater efficiency.

The wealth of new capabilities that statistical learning systems brought to the security community make them a prime target for adversaries. Several studies have shown how security systems that employ machine learning techniques can be attacked [36, 49, 52, 60, 62], decreasing their overall detection accuracy. This reduction in accuracy makes it possible for adversaries to evade detection, rendering defense systems obsolete.

While graph based network detection systems are not immune to adversarial attack, the community knows little about *practical attacks* that can be mounted against them. As these network detectors face a range of adversaries (e.g., from script kiddies to nation states), it is important to understand the adversary's capabilities, resources, and knowledge, as well as the cost they incur when evading the systems.

In this paper we present the first practical attempt to attack graph based modeling techniques in the context of network security. Our goal is to devise generic attacks on graphs and demonstrate their effectiveness against a real-world system, called Pleiades [9]. Pleiades is a network detection system that groups and models unsuccessful DNS resolutions from malware that employ domain name generation algorithms (DGAs) for their command and control (C&C) communications. The system is split into two phases. First, an unsupervised process detects new DGA families by clustering a graph of hosts and the domains they query. Second, each newly detected cluster is classified based on the properties of the generated domains.

To evade graph clustering approaches like Pleiades, we devise two novel attacks—targeted noise injection and small community—against three commonly used graph clustering or embedding techniques: i) community discovery, ii) singular value decomposition (SVD), and iii) node2vec. Using three different real world datasets (a US telecommunication dataset, a US university dataset and a threat feed) and after considering three classes of adversaries (adversaries with minimal, moderate and perfect knowledge) we mount these two new attacks against the graph modeling component of Pleiades.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '17, October 30-November 3, 2017, Dallas, TX, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134083

We show that even an adversary with minimal knowledge, i.e., knowing only what is available in open source intelligence feeds and on their infected hosts, can evade detection.

Beyond devising practical attacks, we demonstrate that the attacks are inexpensive for adversaries. Fortunately, defenders are not without recourse, and detection systems' parameters can be tuned to be more resistant to evasion. Based on these discoveries, we make recommendations to improve Pleaides' resilience.

Our paper makes the following contributions:

Two Novel Attacks. The *targeted noise injection attack* improves on prior work that randomly injects noise; by targeting the injected vertices and edges to copy the graph structure of the original signal, we force noise into the resulting clusters. Our *small community attack* abuses the known property of small communities in graphs to subdivide and separate clusters into one or more unrelated clusters.

Practical Attacks and Defenses. While more knowledgeable attackers typically fare better, we demonstrate that even minimal knowledge attackers can be effective: attackers with no knowledge beyond their infections can render 84% of clusters too noisy to be useful, and evade clustering at a rate of 75%. The above attacks can be performed at low cost to the adversary by not appearing to be anomalous, nor losing much connectivity. Simple defenses raise the attacker's costs and force only 0.2% of clusters to be too noisy, and drop the success rate to 25%. State of the art embeddings, such as node2vec, offer more adversarial resistance than SVD, which is used in Pleiades.

2 BACKGROUND

2.1 Graph-based Clustering

Graph clustering is commonly used in security. Community discovery identifies criminal networks [39], connected components track malvertising campaigns [21], spectral clustering on graphs discovers botnet infrastructure [9, 20], hierarchical clustering identifies similar malware samples [11, 45], binary download graphs group potential malware download events [29, 40, 41], and newly devised graph embeddings, like node2vec [26], could further improve upon the state of the art. Beyond clustering, other graph-based techniques are used, such as belief propagation [18, 46]. Unfortunately, it is unknown how resistant these techniques are to adversarial evasion.

2.1.1 Community Detection. There are many ways to detect communities in a graph. Several techniques in this space rely on a modularity metric to evaluate the quality of partitions, which measures the density of links inside and outside communities. This allows an algorithm to optimize modularity to quickly find communities. The Louvain algorithm [14] scales to large networks with hundreds of millions of vertices. Communities are usually hierarchical [42, 47, 50]; however, finding sub-communities within communities is a known hard problem [15]. This allows attackers to hide sub-communities in a "noisy" community by adding edges.

2.1.2 Spectral Methods. In [57], Braverman et al. discuss several popular spectral clustering strategies. First, a similarity matrix is used to represent the graph. Each row and each column represent a vertex to be clustered, and the weight is a similarity score between the corresponding vertices. After proper normalization, the matrix M is used as input to singular value decomposition (SVD) of rank k, $SVD_k(M) = U\Sigma V^*$. When the resulting eigenvectors (e.g., vectors in U) are further normalized, they can be used as an embedding in a euclidean space for learning tasks. In spectral methods, the hyperparameter k is usually chosen by first evaluating the scree plot of eigenvalues to identify the "elbow" where higher ranks have diminishing returns of representing the input matrix. When the scree plot starts to plateau at the ith eigenvalue, we set $k = i$ [17, 59].

Spectral clustering with SVD is known to have limitations when clusters are imbalanced; this is due to either graphs being scale-free (power law distribution) [31], or when small communities exist [30]. Unfortunately, both commonly occur in real-world data. In practice, these small communities are merged into what is colloquially called the "death star" cluster: a large, noisy cluster that contains many small communities.

2.1.3 node2vec. Contrary to the strong homophily assumption of community detection and spectral clustering, node2vec [26] has the advantage of balancing homophily and structural equivalence in its embeddings. For example, vertices that are sink nodes will have similar embeddings. node2vec generates embeddings of vertices by optimizing the sum of the log likelihood of seeing the network neighborhood given a vertex v, for all vertices on the graph:

$$\max_f \sum \log P(N_S(v)|f(v)) \tag{1}$$

Where $f(v)$ is the embedding of vertex v, $N_S(v)$ represents the network neighborhoods of v with a series of vertices obtained by the sampling strategy S. node2vec proposes a sampling strategy by random walks starting from every vertex on the graph with the following parameters: 1) number of walks from each vertex, 2) length of each walk, 3) probability to return to the same vertex (Breadth First Search), and 4) probability to explore out to further vertices (Depth First Search). Once the walk samples have been obtained, node2vec uses a tunable *neighborhood size* to get the neighborhood of vertices. For example, a walk with length 5 $\{v1, v2, v3, v4, v5\}$ generates the following neighborhoods with size 3: $N(v_1) = \{v2, v3, v4\}$, $N(v_2) = \{v3, v4, v5\}$.

In order to compute the embeddings given $f(v)$, Equation 1 is factorized as a product of the conditional probability of each vertex in the neighborhood based on the conditional independence assumption. Each underlying conditional probability is defined as a sigmoid function, and the embeddings are learned by stochastic gradient descent (SGD) with negative sampling optimization. Effectively, node2vec learns embeddings in a fashion similar to word2vec [37] but does not use skip-grams. Attackers can target the neighborhood size and sampling parameters to encourage their vertices to be under-sampled and thus split into multiple noisy clusters.

2.2 Related Work

Existing work in adversarial machine learning has focused on analyzing the resilience of classifiers. Huang et al. [28] categorize attack influence as either *causative* or *exploratory*, with the former polluting the training dataset and the latter evading the deployed system by crafting *adversarial samples*. Following the terminology of Huang et al., our work focuses on *exploratory* attacks that target

the graph clustering component of Pleiades. We assume that the clustering hyperparameters are selected with attack-free labels, and the subsequent classifier is not polluted when they are trained. Contrary to other *exploratory* attacks in literature, we face the challenge that the clustering features cannot be modified or computed directly, and that attackers often have an incomplete view of the defender's data.

In order to compute optimal graph partitions or vertex embeddings, one needs to have a *global* view of all objects on the graph. On the contrary, related work can compute classification features directly from crafting adversarial samples. For example, features are directly obtained from spam emails [36, 60], PDF files [49, 53, 62], phishing pages [49], images [16, 25, 44, 52], network attack packets [24], and exploits [55, 58]. These security applications classify an object based on features extracted from only that object and its behavior. This makes the features of system classifiers more *local*, and enables evasion techniques such as gradient descent directly in the feature space. We make the following definition: a *local* feature can be computed from only one object; whereas a *global* feature needs information from all objects being clustered or classified.

Since Pleiades uses *global* features, an adversary's knowledge can affect the success of attacks. For example, if the adversary has full access to the defender's datasets, she can reliably compute clustering features and is more equipped to evade than a less knowledgeable attacker. Many researchers [43, 56] have shown that, even without access to the training dataset, having knowledge about the features and an oracle to obtain some labels of objects is sufficient for an attacker to approximate the original classifier.

Biggio et al. [12, 13] are the first to study adversarial clustering. They propose a bridge attack, which works by injecting a small number of adversarial samples to merge clusters. The attackers have perfect knowledge in their assumption. We distinguish our work by i) considering attackers with different knowledge levels, ii) evaluating how adversarial graph-clustering in network security affects the whole system, and iii) quantifying the cost of attacks. With respect to attack cost analysis, Lowd et al. [35] propose a linear cost function as a weighted sum of feature value differences for crafting evasive adversarial samples. Since we do not work directly in the feature space, we propose different costs for the attacks we present in Section 3.

To summarize, our work is novel because we focus on adversarial clustering, which deals with global features that cannot be directly changed. We also evaluate capabilities of attackers with various knowledge levels, and quantify the costs associated with attacks.

3 THREAT MODEL & ATTACKS

In this section, we describe our threat model and explain our attacks as modifications to a graph G. In practice, the attacker changes the graph based on the underlying data that are being clustered. For example, if the vertices in a graph are infected hosts and the domains they query as in Pleiades, the graph representation can be altered by customized malware that changes its regular querying behavior.

3.1 Notation

An undirected graph G is defined by its sets of vertices (or nodes) V and edges E, where $G = (V, E)$ and $E = \{(v_i, v_j) : \text{if there exists an edge between } v_i \text{ and } v_j, v_i \in V, v_j \in V\}$. An *undirected bipartite graph* is a special case where V can be divided into two disjoint sets (U and V) such that every edge connects at a vertex in U and one in V, represented as $G = (U, V, E)$. While the attacks apply in the general case, oftentimes bipartite graphs appear in security contexts: hosts (U) query domains (V), clients connect to servers, malware make system calls, etc. Finally, a *complete* undirected bipartite graph is where every vertex in U has an edge to every vertex in V.

$\mathcal{G}$ is an undirected graph that represents the underlying data a defender clusters. The graph clustering subdivides $\mathcal{G}$ into clusters $C_0, \ldots, C_k$, where $V = C_0 \cup \ldots \cup C_k$. If the graph clustering method is based on graph partitions, then each cluster C_i is a subgraph G_i, and $\mathcal{G} = G_0 \cup \ldots \cup G_k$. Often when applied, a defender seeks to cluster vertices either in U or V of the bipartite graph, for example, cluster end hosts based on the domains they resolve, or malware based on the system calls they make. An attacker controls an *attacker graph*, $G \subset \mathcal{G}$. The adversary uses the targeted noise injection and the small community attacks described below to change G to G', by adding or removing nodes and edges from G.

These attacks violate the underlying basic assumptions of graph clustering techniques, which either renders the clustered subgraph G' to be useless to the defender or prevents G' from being extracted from $\mathcal{G}$ intact (See Section 3.3).

3.2 Threat Model

Before describing attacker knowledge levels, we discuss knowledge that is available to all attackers. We assume all attackers have at least one active infection, or $G \subset \mathcal{G}$. The attacker is capable of using any information that could be gathered from G to aid in their attacks. We also assume that an attacker can evaluate clusters like a defender can, e.g., manual verification. When done with a classifier, an attacker has black-box access to it or can construct a surrogate that approximates the accuracy and behavior of the real classifier based on public data. This may seem extreme, but the plethora of open source intelligence (OSINT) [4, 6, 22] data and MLaaS machine learning tools [1–3, 5, 7] make this realistic. Finally, an attacker has full knowledge of the features, machine learning algorithms, and hyperparameters used in both the unsupervised and supervised phases of the system under attack, as these are often published [8, 9, 20, 29, 41, 45, 48]. Since clustering requires some graph beyond G, we must consider attackers with various representations of the defender's $\mathcal{G}$. We evaluate three levels: minimal, moderate, and perfect knowledge. The minimal level attacker only knows what is in their attack graph G, but the perfect attacker possesses $\mathcal{G}$. For example, a perfect adversary would have access to the telecommunication network data used in Pleiades, which is only obtainable by the most sophisticated and well resourced of adversaries.

Minimal Knowledge. The minimal knowledge case represents the least sophisticated adversary. In this case, only the attacker graph G is known, as well as any open source intelligence (OSINT). For example, the attacker can use OSINT to select potential data to inject as noise, or can coordinate activities between their vertices in G. In the Pleiades example, an attacker with minimal knowledge can draw information from their infected hosts.

Moderate Knowledge. The moderate knowledge case represents an adversary with $\tilde{\mathcal{G}}$, an approximation of $\mathcal{G}$. If attacking Pleiades, $\tilde{\mathcal{G}}$ would be a host/domain graph from a large enterprise or university in order to approximate the view that the defender has. This allows the adversary to evaluate their attacks. The size of $\tilde{\mathcal{G}}$ affects the evaluation from the attacker's perspective, which we will explore by randomly sampling subgraphs of $\tilde{\mathcal{G}}$. An attacker with moderate knowledge is similar to a sophisticated adversary with access to large datasets through legitimate (i.e., commercial data offerings) or illegitimate (i.e., security compromises) means.

Perfect Knowledge. Finally, the perfect knowledge case is for an adversary who has obtained $\mathcal{G}$ from the defender. Given the full dataset and knowledge of the modeling process, an adversary can completely reconstruct the clustering results of the defender to evaluate the effectiveness of their attacks. Ideally, this data would be well guarded making this level of knowledge only realistic for the most sophisticated of attackers, e.g., nation-state sponsored threats. Nevertheless, considering the damage that could be done by a perfect knowledge attacker is important as a security evaluation, since it allows us to find potential weaknesses in the graph clustering techniques.

3.3 Attacks

We present two novel attacks against graph clustering. The first, *targeted noise injection*, improves on random injections [34, 54] by emulating the legitimate signal's graph structure. The second, *small community attack*, exploits the known phenomenon of *small communities* in graphs [30, 32]. Our attacks violate both the homophily and the structural equivalence assumptions used by graph clustering methods. That is, our attacks either change what nodes are close together to violate homophily, or they change observations of node neighborhoods so as to violate structural equivalence.

Identifying a successful attack depends on the system, which will be described in detail in Section 4. Since we use Pleiades, we evaluate attacks by the impact on a subsequent classification of the resulting adversarial clusters. However, this could be done purely at the unsupervised level by *manually* evaluating the accuracy of the output clusters, or leveraging prior work in adversarial malware clustering [12] to measure global cluster quality decrease. Next, we evaluate the cost incurred by the attacker. We analyze the costs by measuring changes to their graph's structure that would either flag them as anomalous or damage connectivity between the graph's vertices. In the descriptions below, an attacker's initial graph G is shown, and the alterations yield a modified graph, G', that represents a defender's view of the attacker's graph after the adversarial manipulation.

3.3.1 Targeted Noise Injection. Figure 1 illustrates two targeted noise injection attacks. Consider a bipartite attacker graph G, with vertex sets U (circles) and V (squares). To mount the attack, noise is injected into G to generate G'. We inject noisy edges from nodes controlled by the attacker for the purpose of mirroring real edges. This encourages newly connected nodes to be clustered together with the attacker's nodes.

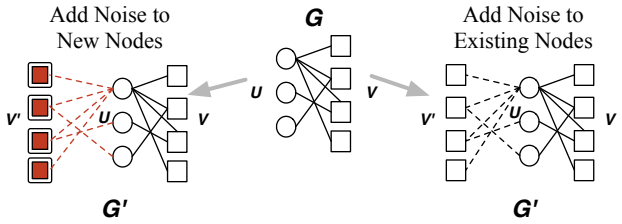


Figure 1: Example of targeted noise injection attacks on a graph.

Algorithm 1 Targeted Noise Injection Attack Algorithm for Attacker $\mathcal{A}$ controlling G

Input: $\mathcal{A}, m, G = (U, V, E)$
Output: G'

1: **for** $i = 1$ **to** m **do**
2: $V_i' \leftarrow$ according to knowledge of $\mathcal{A}$
3: **for** $v' \in V_i'$ **do**
4: Mirror the edges such that $f : (u, v) \in E \mapsto (u, v') \in E_i'$ is bijective.
5: **end for**
6: **end for**
7: Return $G' = (U, (\bigcup_{i=1}^{m} V_i') \cup V, (\bigcup_{i=1}^{m} E_i') \cup E)$

To inject noise, the attacker creates an additional vertex set V', represented by red squares. Entities in V' should differ substantially from those in V, which depend on the underlying system to be evaded. In Pleiades' case, this means the injected domains (V') must be different, in terms of character distribution, from the legitimate domains (V). Then, for every edge between U and V, the attacker creates a corresponding edge between U and V', as shown in Figure 1. That is, the attack function $f : (u, v) \in E \mapsto (u, v') \in E'$ is bijective. This creates $G' = (U, V \cup V', E \cup E')$, where E' are the corresponding edges from U to V', denoted by dotted red edges in the figure. The other way to inject noise is to create edges from U to existing nodes from $\mathcal{G}$, as shown in Figure 1. This does not add additional nodes, but identifies other vertices on the defender's graph $\mathcal{G}$ to use as V'. A new edge is created for all edges between U and V. Attacker information is used to identify additional nodes to use. Example nodes may include other non-malicious domains queried by infected hosts, or a machine's existing behavior observed by the malware. More commonly, it requires some knowledge of the graph being clustered, $\mathcal{G}$. This process can be repeated to increase $|V'|$ to be multiples of $|V|$.

Algorithm 1 formally describes noise injection for attacker $\mathcal{A}$ controlling the attacker graph G, with noise level m. Line 1 to Line 6 repeats the noise injection process m times. In line 2, $\mathcal{A}$ generates the set of *noisy nodes* V' according to her knowledge. From line 3 to 5, the attacker creates a one-to-one mapping from E to E_i'. Line 8 returns the manipulated attacker graph $G' = (U, (\bigcup_{i=1}^{m} V_i') \cup$

$V, (\bigcup_{i=1}^{m} E'_i) \cup E)$. We will evaluate two variants to determine how much noise is needed to mount a successful, but low cost attack. In the first variant $m = 1$, and in the second $m = 2$.

While additional edges and nodes could be injected arbitrarily at random, we choose to mirror real edges in order to make both nodes from V' and V have similar embeddings. We define V' to be the set of *noisy nodes*. The targeted noise injection attack exploits the homophily assumption [14, 57] of graph clustering methods. In community discovery and spectral methods, graph partitions cannot distinguish injected noisy nodes (V') from real nodes (V), which exhibit structurally identical connections to U. The co-occurrence increases the observation of noisy nodes appearing in neighborhoods of real nodes, and vice versa for node2vec. We expect nodes from V' to join existing clusters containing V.

The targeted noise injection attack has a cost for the attacker of raising the profile of nodes belonging to attacker graph G, potentially making them outliers. Specifically, hosts from U will increase in percentile with respect to their degree, i.e., a relatively high degree could indicate anomalous behavior, which we can measure by the increase in percentile ranking changes before and after an attack. We call this the *anomaly cost*.

For attacking Pleiades, consider a graph where U are infected hosts and V are the domain names that the hosts in U query. To generate G' an attacker instructs their malware to query an additional domain ($v \in V'$) for each domain used for its normal malicious operation ($v \in V$). This causes the domains from V and V' to conflict such that the clustering is not useful to the defender. However, the anomaly cost may make these trivial to detect. Nonetheless, we will show in Section 5.2.4 that the cost of attack is small enough to be practical.

3.3.2 Small Community. Figure 2 illustrates four potential small community attacks of increasing intensity. The small community attack removes edges and/or nodes such that the graph clustering separates a single attack graph into multiple clusters, while maintaining as much connectivity from the original graph as possible. Again, G is a bipartite attacker graph with identical vertex and edge sets as before. To mount the attack, an adversary first constructs a complete version of G, $\hat{G}$. In $\hat{G}$, every vertex in U has an edge to every vertex in V. To construct G', the adversary removes edges from $\hat{G}$. In Figure 2, the attacker has removed one and two edges per vertex in V in G'_1 and G'_2, respectively. Then in G'_3 and G'_4, the attacker has removed a vertex from V, and then removed one and two edges per remaining vertex. The attacker randomly chooses n_v (such that $0 \leq n_v \leq |V| - 1$) nodes to remove, and n_e (such that $0 \leq n_e \leq |U| - 1$) edges from each remaining node V in $\hat{G}$. In the extreme case, there is only one vertex remaining from V connecting to one in U, which often cannot be captured by graph embeddings. Each attack instance is configured with (n_v, n_e) pair, or, in other words, the $(|V| - n_v, |U| - n_e)$ pair to keep nodes and edges. We define the *attack success rate* as the number of successful attack configurations divided by $|U| * |V|$.

If the attacker has minimal knowledge, she can choose n_v and n_e randomly, and hope for the best. With perfect knowledge (knows $\mathcal{G}$), she can choose the smallest n_v and n_e that succeed. Attackers without some knowledge or approximation of $\mathcal{G}$ will be unable to verify if their attacks succeed. While G could be manipulated directly, removing nodes and edges lowers the utility for the attacker by losing connectivity in their attack graph. We aim to reduce the average edge number per node of V in G, while simultaneously maintain the lowest possible cost. Constructing and altering $\hat{G}$ both simplifies the experiments of quantifying the small community attack cost and makes the job of the attacker easier. We believe this does not negate the correctness of our experiments.

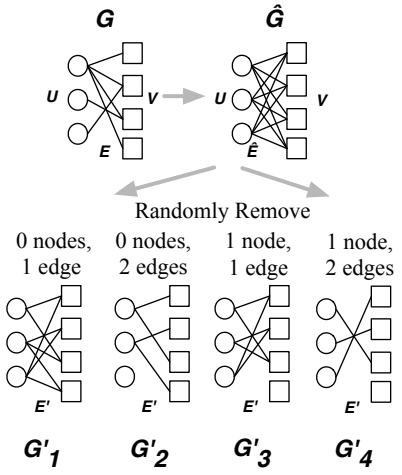


Figure 2: Example small community attacks on a graph.

Algorithm 2 Small Community Attack Algorithm for Attacker $\mathcal{A}$ controlling G

Input: $\mathcal{A}, G = (U, V, E)$
Output: G'

1: Construct $\hat{G} = (U, V, \hat{E})$ from G, where $|\hat{E}| = |U| * |V|$
2: $n_v, n_e \leftarrow$ according to knowledge of $\mathcal{A}$, where $n_v < |V|, n_e < |U|$
3: $V' \leftarrow$ Choose $|V| - n_v$ random nodes from V
4: **for** $v' \in V'$ **do**
5: Choose $|U| - n_e$ random edges that connect to v' to update U' and E'
6: **end for**
7: Return $G' = (U', V', E')$

Algorithm 2 formally shows the the small community attack $\mathcal{A}$ in control of graph G. Line 1 constructs the complete graph $\hat{G}$ from G. In line 2, $\mathcal{A}$ chooses (n_v, n_e) according to her knowledge. Then, the attacker chooses $|V| - n_v$ random nodes from V as V'. Each node in V' connects to all nodes in U. From line 4 to 6, the attacker chooses $|U| - n_e$ random edges to keep for each node in V', and thus forming U' and E'. Lastly, line 7 returns $G' = (U', V', E')$ as the manipulated attacker graph.

The small community attack exploits the information loss in graph embeddings. While community discovery works better at identifying islands and singletons, graph embeddings may miss such signal given the hyperparamters chosen at deployment. Existing methods for choosing hyperparameters do not account for

potential small community attack opportunities. The downside of the attack is the *agility cost*. By removing nodes and edges from G, the adversary has to give up control over nodes, redundancy, or even functionality. In addition to losing n_v, the agility cost can be measured by the change in *graph density* (Equation 2) from $\mathcal{D}(G)$ to the chosen $\mathcal{D}(G')$. We define the following $D(G)$ and $D(G')$:

$$\mathcal{D}(G) = \frac{|E|}{(|U| * |V|)} \tag{2}$$

$$\mathcal{D}(G') = \frac{|E'|}{(|U| * |V|)} \tag{3}$$

A graph's density ranges from [0, 1], which denote a graph with zero edges or all possible edges, respectively. For $D(G')$ we consider how many edges are in E' compared to the maximum possible number of edges between the original U and V. This normalizes the number of edges by the structure of the G. The *agility cost* is $D(G) - D(G')$ if $D(G) > D(G')$, or zero if $D(G) \leq D(G')$. A loss in density implies a potential loss of connectivity, but maintaining or increasing the density bodes well for the attacker. They can afford an even denser structure, yet still evade defenders. It is important to note that, while n_v is lost, this is reflected in the density score, as $|V|$ includes any removed vertices like n_v. A lower density, and therefore a higher cost, is incurred when edges and/or vertices are removed relative to the original structure seen in G.

Consider an attack on Pleiades. $\hat{G}$ is created by completing G. To mount the attack like G'_2, the adversary partitions the domain names that are used to control her malware by removing one of the control domains (n_v=1), and then excludes two distinct hosts that query each of the remaining domains (n_e=2). In other words, the adversary can also randomly choose one host ($|U| - n_e$) to query each one of remaining control domains. This reduces the density from $D(G) = 0.5$ to $D(G'_2) = 1/3$, and sacrifices one node (n_v).

If the adversary has knowledge that allows testing whether the attack is successful or not, the attacker can increasingly remove domains and queries from hosts until clustering $\mathcal{G}$ no longer results in G' being extracted as a single cluster. In practice, as described in Section 2.1, the subdivided G' often ends up either as portions of the "death star" cluster; or in multiple, noisy clusters. In both cases, the legitimate cluster is effectively hidden in a forest of noise. In order to verify an attack was successful, however, an attacker must have $\mathcal{G}$ or an approximation.

4 ATTACKS IN PRACTICE

We chose to attack Pleiades because it has been commercially deployed and relies on graph modeling. Our reimplementation has similar performance, as shown in Appendix C. We now describe portions of the reimplementation in detail.

4.1 Pleiades

An overview of Pleiades is shown in Figure 3. We focus our attacks on the clustering component and use the classification phase to demonstrate attack success. First, Pleiades clusters NXDOMAINs (V) queried by local hosts (U) using the host-domain bipartite graph ($\mathcal{G}$). It groups together NXDOMAINs queried by common hosts into clusters $C_0, \ldots, C_k$, based on the assumption that hosts infected

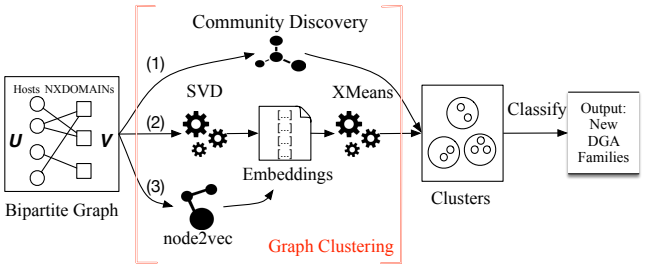


Figure 3: Overview of the DGA detection system.

with the same DGA malware query similar groups of DGA domains. The graph clustering can be achieved by either community discovery (1), spectral clustering (2) as in the original paper [9], or node2vec clustering (3). Then the classification module computes domain name character distributions of each cluster into a numerical feature vector, which is used to classify known DGA families. A new unknown DGA family with features statistically similar to a known one can be detected by this process. The system operates on daily NXDOMAIN traffic generated by all hosts in a network using the following data sources.

4.1.1 Datasets. We use anonymized recursive DNS traffic from a large telecommunication company from December 18, 2016 to December 29, 2016. The dataset contains NXDOMAINs queried by hosts and the query timestamps. On average, there are 262 thousand unique anonymized hosts, with 44.6 million queries to 1.8 million unique NXDOMAINs in a day. We use this dataset to construct Host-NXDOMAIN Graph as ground truth without attack. This is available to defenders and perfect knowledge attackers.

As a surrogate network dataset, we use NXDOMAIN traffic from a large US university network collected on December 25, 2016. It contains 8,782 hosts and 210 thousand unique NXDOMAINs. Among these NXDOMAINs, only 227 appeared in the ground truth network dataset. The surrogate dataset is available to attackers with moderate and perfect knowledge.

Last but not least, we use a reverse engineered DGA domains dataset to train the supervised component of the system. We run the reverse-engineered algorithms [10] to generate DGA domains for 14 malware families: Chinad, Corebot, Gozi, Locky, Murofet, Necurs, NewGOZ, PadCrypt ransomware, Pykspa, Qadars, Qakbot, Ranbyus, Sisron, and Symmi. The training dataset also includes live DGA domains observed in the ground truth network. We label 267 clusters belonging to four malware families present in the ground truth network dataset (Pykspa, Suppobox, Murofet, and Gimemo), and manually verify that these subgraphs are attack free. We train a Random Forest classifier with an average accuracy of 96.08%, and a false positive rate of 0.9%. The classifier trained from this dataset is available for attackers of all knowledge levels. Table 1 summarizes these datasets.

We discovered 12 new DGA malware families in only 12 days using the ground truth network traffic (see Appendix C for details). We discovered real but unsuccessful evasion attempts in the wild, and retrained our classifier with evasive instances. We believe we have faithfully reimplemented Pleiades because we use comparable datasets and we achieve similar clustering and modeling results.

Dataset	Number of Records	Minimal	Moderate	Perfect
Reverse Engineered DGA Domains	14 DGA Families; 395 thousand NXD	X	X	X
Host-NXDOMAIN Graph (Surrogate)	8782 hosts; 210 thousand NXD	-	X	X
Host-NXDOMAIN Graph (Ground Truth)	average 262 thousand hosts; 1.8 million NXD	-	-	X

Table 1: Summary of datasets and their availability to minimal, moderate, and perfect knowledge attackers.

4.2 Attacks

Using the notation described in Section 3, let $\mathcal{G}$ be a bipartite graph of the defender. U represents hosts, both infected and uninfected, and V represent NXDOMAINs queried by hosts in the underlying network. An edge exists from $v_i \in U$ and $v_j \in V$ iff the ith host queried the jth NXDOMAIN. For an attacker graph $G \subset \mathcal{G}$, the hosts in U are infected hosts under the control of the attacker. In the noise injection case, the attacker instructs their malware to query NXDOMAINs beyond what is needed for normal operation, as shown in Figure 1. In the small community case, the attacker coordinates the querying behavior of their malware such that they query fewer NXDOMAINs in common, as in Figure 2. We will evaluate the effectiveness of the attacks by the drop in predicted class probabilities and the predicted label of the classifier. In a Random Forest, the predicted class probabilities of a feature vector are calculated as the average predicted class probabilities of all trees in the forest. In practice, if the predicted class probability decreases substantially, the classifier will incorrectly label the instances, and the attack will be considered successful.

4.3 Attack Costs

To compute the anomaly cost for noise injection, we analyze percentile changes of edges related to hosts in U in the structure of $\mathcal{G}$ from before and after the attack. We quantify this change by computing the cumulative distribution functions (CDFs, example in Appendix A) of vertex degrees before and after a successful attack is mounted. Concretely, if an attacker can evade Pleiades but raises the profile of their infected hosts from the 50^{th} (in the CDF before attack) to the 99.9^{th} percentile of NXDOMAINs queried per host (in the CDF after attack), a defender will be able to detect such behavior with simple thresholding (i.e., monitoring hosts entering the 95^{th} percentile).

To quantify the adversarial cost behind the small community attack, we measure the change of attacker graph density $D(G')$ as defined in Section 5.3. If the attacker graph density decreases, this means the attacker no longer uses NXDOMAINs for their infection and/or the infected hosts query fewer NXDOMAINs in common, reducing their connectivity overall and increasing the botnet's management cost.

5 RESULTS

First, we show how to select hyperparameters for each of the three graph methods. Next, we present our results for both attacks against each graph based clustering technique, for the three knowledge levels. Finally, we explain the costs incurred by the attacker, and how these can be used to identify possible defenses.

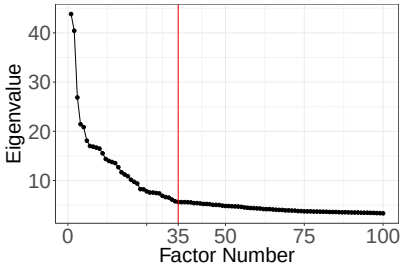


Figure 4: Scree plot of eigenvalues of SVD.

Summary of Results. Our attacks against the graph clustering component of Pleiades gravely reduce the predicted class probability of the subsequent classification phase. Even with minimal knowledge, an adversary can launch an effective targeted noise injection attack dropping the median predicted class probability to 0%. In the higher knowledge levels, the maximum predicted class probability can be as low as 10%. Using a set of labeled DGA malware families observed in spectral clustering, the attacks reduce the prediction accuracy from 99.6% to 0%.

In addition to being effective, the attacks do not substantially raise the anomaly profile of infected hosts: before and after the targeted noise injection attacks the hosts occupy a similar percentile for the number of NXDOMAINs queried. Small community attack results show that the traditional way of choosing hyperparameters for generating graph embeddings is insufficient when we analyze the system in an adversarial setting, because it creates a large area for possible small community attack instances. While following the accepted methodology for selecting the rank for SVD and hyperparameters for node2vec, all DGA clusters can be hidden in noisy clusters by subdividing the infected hosts into smaller groups to sacrifice some agility, even while using hundreds of DGA domains. Even in the minimal knowledge case where the small community attack cannot be tested, attackers can sometimes still hide.

5.1 Choosing Hyperparamters

First, we carefully choose hyperparameters for the graph clustering methods in Pleiades to ensure high quality clusters are generated. Figure 11 shows the results used to determine the appropriate hyperparameters.

5.1.1 Spectral Clustering. We use the scree plot shown in Figure 4 to choose the rank of SVD. We fix the rank of the SVD to be 35, where the scree plot plateaus. While different than the 15 used in the original Pleiades implementation [9], the underlying datasets are different so it is not unreasonable to find different ranks.

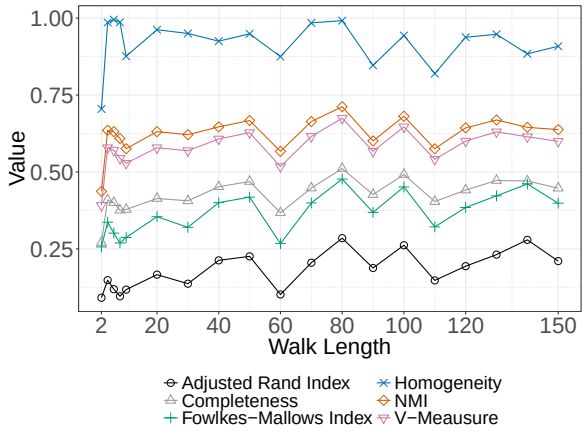


Figure 5: Using cluster validity metrics to choose walk length.

5.1.2 Community Discovery. We use the best partition method from the NetworkX community discovery library [61] which implements the Louvain algorithm [14]. The Louvain algorithm efficiently extracts good communities on the graph by optimizing the modularity metric, which measures the density of links within communities in comparison to outside them. It first sets each node to be its own community, and iteratively merges them to maximize modularity. The algorithm stops when a local maxima is reached. This community detection algorithm scales to large network with hundreds of millions of nodes.

5.1.3 node2vec. We use traditional cluster validity metrics to compare different hyperparameters of node2vec. Twelve DGA malware families, including both known and newly detected ones, were used as *reference clusters*. We use validity metrics including Adjusted Rand Index, Completeness, Fowlkes-Mallows index, Homogeneity, Normalized Mutual Information (NMI), purity, and V-Measure score. We first choose context size six, which has the first highest validity scores. Several larger context sizes generate equal validity scores, but they produce noisier clusters. This is because that larger context sizes in DNS graphs tend to include more noisy nodes, such as popular benign NXDOMAINs, or popular hosts that are likely proxies.

Then we choose the walk length according to Figure 5. Multiple walk length values produce high validity scores, but we choose walk length 20, which corresponds to the second highest peak. Because using walk length 20 generates cleaner Murofet clusters than a walk length smaller than 10, due to the fact that longer walk length provides more samples of neighborhoods and the model is learned better. The number of walks per node, dimensions, and SGD epoch does not show much difference. We decide on 15 walks, 60 dimensions, and one learning epoch after manual inspection. Lastly, we use a uniform random probability to choose the next node in the random walk process.

5.2 Targeted Noise Injection

We run our version of Pleiades to generate all attacker graphs. Four DGA families were identified: Pykspa, Suppobox, Murofet, and Gimemo. For each we extract the attacker graphs (G) and the ***target domains*** (V). These domains are labeled using the classifier from Section 4.1.1. Before and after the attack, there can be multiple clusters formed within G and G', depending on the graph clustering technique. We use the classifier model to test how likely it is that each cluster belongs to the true DGA malware family, both before and after the attack. We present the overall distribution of the *predicted class probabilities* to show the impact of the attacks.

We use different types of noisy domains at different knowledge levels. For a DGA, these nodes are new NXDOMAINs (V') that will be classified as benign, also queried by the infected hosts (U). In the minimal knowledge case, we create a DGA algorithm that is classified as benign. It is a dictionary DGA that uses the most popular English words from movie subtitles [23], popular web terms, and the top one million Alexa domains. We randomly add numbers and dashes, and randomly select a top-level domain from four choices. In addition, we generate some punycode domains that start with the character sequence "xn−", and some domains with a "www" subdomain. We generate 59,730 verified NXDOMAINs. In the perfect and moderate knowledge cases, the adversary uses existing, unpopular NXDOMAINs from $\mathcal{G}$ and the surrogate dataset, respectively.

5.2.1 Spectral Clustering. Figure 6a shows the classifier's predicted true class probabilities from before the attack is mounted, and after the minimal, moderate, and perfect knowledge targeted noise injection attacks are performed. For each knowledge level, we inject two different levels of noise as described in Section 5.2 and re-run the clustering and subsequent classification to assess the damage from the targeted noise injection. Recall that we try two attack variants, attack variant 1 and 2, where we inject one or two mirrored sets of vertices and edges, respectively. This is to both i) understand how much noise is needed to yield successful evasion, and ii) determine the cost incurred by adding noise.

Spectral clustering generates 267 DGA clusters from the four malware families across 12 days. Before the attack, only 0.4% clusters (1 out of 267) are predicted with the wrong labels. In comparison, after the attacks, all clusters are predicted with the wrong labels. Next, we will examine the predicted class probabilities change in the true class label.

Figure 6a uses the violin plots to show the distribution of predicted class probabilities for the *true* DGA families, before and after the attacks. The circle is the median value, the thick line within the violin indicates interquartile range, and the thin line depicts a 95% confidence interval. The shape of the violin shows the distribution of the probability values. For example, the first violin in Figure 6a has a median of 100% predicted class probabilities, and all data points in the interquartile range have 100% probability value. Specifically, before the attacks, 238 clusters are predicted with 100% class probability that they belong to the true class, and only 28 clusters have a probability between 60% and 100%. For example, the Pykspa cluster had a class probability of only 10% because it contained only two domain names that had very different feature distributions from the majority of Pykspa clusters. The two variants

of the attack introduced *at least* 50% and 66% noise to the DGA clusters.

Minimal Knowledge. After the attacks, we classify each new adversarial cluster containing target domains and plot the target class probability distributions in the Figure 6a. Attack variant 1 ("Minimal Benign DGA 1") generated new clusters with ≤ 80% predicted class probability, with a median of 0%. The predicted class probabilities of 84% of the new clusters drop to zero. Attack variant 2 further decreases the classifier prediction confidence, as shown by "Minimal Benign DGA 2" in Figure 6a. After injecting two benign DGA domains, the predicted class probabilities of 87% of the new clusters plummet to 0%. The overall distribution of prediction confidences also shifts downward compared to "Minimal Benign DGA 1".

Perfect Knowledge. The median of predicted class probabilities for DGA malware families drops to 10%. As depicted by "Perfect Long Tail 1" in Figure 6a, 86% of adversarial clusters were assigned the probabilities of belonging to the true DGA class that are at most 10% . The distribution of class probability values has a smaller variance compared to those in the "Minimal Benign DGA 1". "Perfect Long Tail 2" in Figure 6a shows that the maximum prediction confidence is 30%, slightly lower than the maximum 40% confidence from the targeted noise injection attack of "Minimal Benign DGA 2".

Moderate Knowledge. We see similar results for the two targeted noise injection attack variants in the moderate knowledge case as in the other cases: a strong drop in predicted class probabilities, with a smaller, more compact distribution of values for attack variant 2. After attack variant 1, 98.3% of new clusters were assigned less than 20% confidence; after attack variant 2, 98.8% of new clusters have less than 20% confidence.

Spectral clustering can be largely defeated at all knowledge levels using the targeted noise injection attacks.

Since previous experiments show that minimal knowledge attackers can carry out targeted noise injection as effectively as more powerful attackers, we will simply demonstrate that the same targeted noise injection attack variant 1 in minimal knowledge also works with community discovery and node2vec.

5.2.2 Community Discovery. We use the same set of DGA domains labeled in Spectral Clustering for evaluation. Before the attack, 80% clusters can be predicted with the correct label, which dropped to 2% after the attack. Figure 6b shows the predicted class probabilities for communities containing all *target domains* before and after the attack. Before the attack, the median of predicted probabilities is 90%, and the interquartile range is from 50% to 100%. Specifically, 71 communities contain target domains, among which ten communities only contain one target domain, and seven communities have between 40% to 70% target domains. These noisy communities formed the lower part of the distribution, with ≤ 50% predicted class probabilities in "Community Before Attack", as shown in Figure 6b. After the attack, the median class probability craters to 0%. Overall 98% of new communities were predicted with lower than 50% probability of belonging to the true class, and 86% of communities have lower than 10% class probabilities.

This demonstrates that the targeted noise injection attack is also effective against the community discovery algorithm.

Before Attack	**< 95th Percentile, 9.12% of hosts**	
Average Increase	From Percentile	To Percentile
Attack Variant 1	69.86%	88.73%
Attack Variant 2	69.86%	93.98%
Before Attack	**>= 95th Percentile, 90.88% of hosts**	
Average Increase	From Percentile	To Percentile
Attack Variant 1	99.74%	99.85%
Attack Variant 2	99.74%	99.88%

Table 2: Anomaly cost as percentile of the distinct number of NXDOMAINs queried by hosts, before and after the attack. Only 9.12% of infected hosts become more suspicious, while the rest remain the same.

5.2.3 node2vec. Using the same set of DGA domains labeled in Spectral Clustering, before the attack, 89% clusters can be predicted with the correct label, which dropped to 0.8% after the attack. Figure 6b shows that, before the attack on node2vec, the median of predicted probabilities is 100%, and the interquartile range is from 90% to 100%. A total of 85% of clusters were predicted with at least 70% class probability. After the attack, 92% clusters have at most 10% predicted class probabilities.

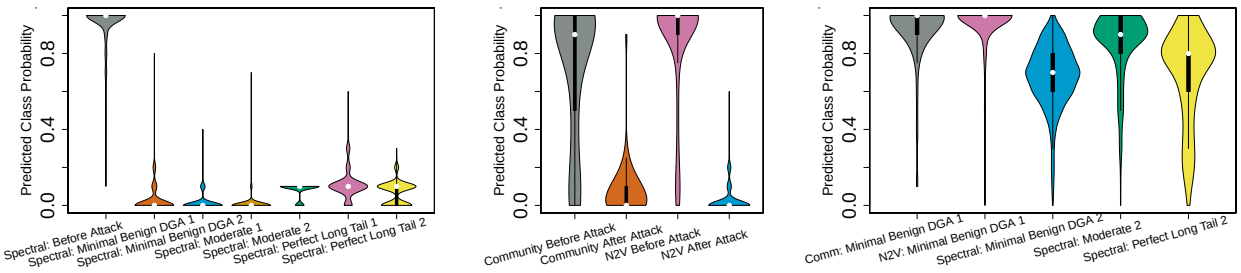
Targeted noise injection attack also evades node2vec embeddings.

5.2.4 Targeted Noise Injection Costs. It is simple for malware to query additional domains, however, infected hosts engaging in such queries may become more suspicious and easier to detect due to the extra network signal they produce. This may cause the anomaly cost of the targeted noise injection attack to be high enough to render it useless.

We analyze the anomaly cost by measuring the infected host percentile of the NXDOMAIN distribution both before and after the attacks for the two variants of the targeted noise injection attacks, summarized in Table 2. Before any attack, only 9.12% of infected hosts ranked lower than 95^{th} percentile, and the remaining 90.88% of them ranked higher than 95^{th} percentile. This means that, without any attack, infected hosts were already querying more unique NXDOMAINs than most hosts in the network. However, doing targeted noise injection attacks further increases the percentile ranks of the infected hosts, but not substantially.

We separated the results based on whether infected hosts were querying fewer domains than 95% of all hosts in the local network. Table 2 shows that among the 9.12% infected hosts ranked lower than 95^{th} percentile before the attack, they increased from an average percentile of 69.86% to 88.73% after the targeted noise injection attack variant 1. Furthermore, they increased to 93.98% after attack variant 2. However, 90.88% of infected hosts did not become more anomalous. They were ranked higher than the 95^{th} percentile before the attack. Their average percentile increased by 0.11% after attack variant 1, and by 0.14% after attack variant 2. Because they were querying more domains than other hosts before the attack, injecting noise does not change their percentile substantially.

The majority of hosts had little change in "suspiciousness", whereas a small percentage of hosts increased their suspiciousness after the targeted noise injection attacks.



(a) Spectral Clustering: Predicted class probabilities. **(b) Community Discovery and node2vec: Predicted class probabilities.** **(c) Retraining: Predicted class probabilities.**

Figure 6: Figure 6a: Predicted class probabilities before the targeted noise injection attack and after two variants of the targeted noise injection attack in minimal, moderate, and perfect knowledge. Figure 6b: Predicted class probabilities before and after the targeted noise injection attacks for community discovery and node2vec. Figure 6c: Predicted class probabilities under different attacks after retraining including the "Minimal Benign DGA 1" clusters.

5.3 Small Community

We choose a group of 618 domains and 10 infected hosts belonging to Suppobox as the basis for the small community attack. They form a community using the community discovery algorithm, and two clusters using spectral or node2vec embeddings. A small community attack is successful if and only if *all* DGA domains join either the "death star," or clusters where the subsequent classifier does not predict them as the true malware DGA class. Recall that the death star cluster contains tens of thousands of domains that cannot be properly classified. In all experiments, the small community plots denote the configurations where an attack succeeds based on the aforementioned criteria. This is represented by green regions (see Figure 7) when the "death star" is joined, or white cells when the noisy clusters cannot be predicted as the true class label (see Figure 9) when using node2vec.

5.3.1 Spectral Clustering. As described earlier, the small community attack can only be verified in the perfect and moderate knowledge cases. In the minimal knowledge case, however, an attacker can still mount the attack by randomly removing edges and nodes, as described in Section 2, while hoping for the best.

Minimal Knowledge. The upper-leftmost plot in Figure 7 demonstrates the possible successful configurations for mounting the small community attack by randomly removing nodes and edges. The plot shows the remaining number of NXDOMAINs on the Y-axis ($|V| - n_v$) and the remaining number of connections from infected hosts for each NXDOMAIN on the X-axis ($|U| - n_e$). The shaded region shows approximately a 75.16% success rate for an attacker with no knowledge of the defender's graph $\mathcal{G}$. While a minimal knowledge attacker cannot guarantee their attack will succeed, they nonetheless have a high chance of success.

Perfect Knowledge. The upper-left plot in Figure 7 depicts a successful small community attack area when the computed SVD rank is 35. The figure shows that only a small set of configurations with 380 to 618 DGA domains, each queried by between 3 to 10 random hosts, were unable to successfully launch a small community attack.

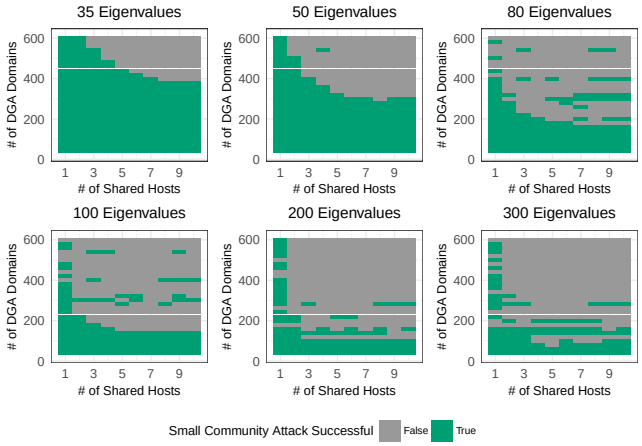


Figure 7: Different number of eigenvalues.

The cost of the small community attack is very low against the system, which is configured with rank 35 and runs in this network. For example, an adversary controlling the DGA does not need to give up any infection, but only needs to reduce the number of infected hosts that query a common set of DGA domains from 618 to 380 in order to hide the domains. In this case, by removing $n_v = 238$ NXDOMAINs, the attacker does not lose any additional host querying activities $min(n_e) = 0$. But if the attacker needs extra redundancy provided by 460 distinct NXDOMAINs, each domain can only be queried by a subset of 5 hosts. Then $n_v = 158$, and accordingly, $min(n_e) = 5$. In this case, the attacker does not need to lose control of any infected hosts, but she does need to coordinate each five infected hosts to query a subset of distinct NXDOMAINs that do not overlap with each other.

Moderate Knowledge. After reducing the number of DGA domains and the number of infected hosts per domain to the successful attack area shown in Figure 8, the DGA domains join the

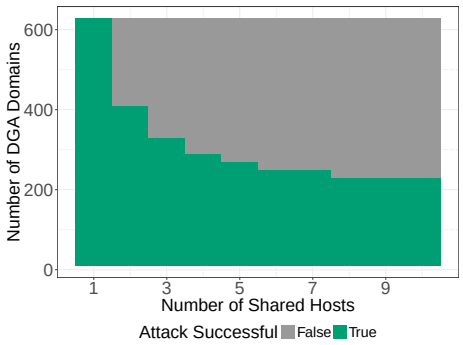


Figure 8: Success area for joining the death star of the surrogate dataset in the moderate knowledge case. All the successful attack configurations worked in the ground truth network.

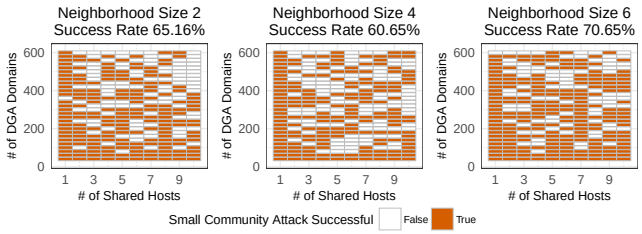


Figure 9: Success area of small community attacks with different context size.

surrogate death star. We test that these values also work to join the original death star. Because the original network size is larger than the surrogate network size, the real successful area (the top-left plot in Figure 7) is much bigger than the one shown Figure 8. Thus, the small community attack works with moderate knowledge when the surrogate dataset is a smaller network than the original network. If the surrogate dataset is a larger network, the adversary may miscalculate the cost of joining the death star, which may not work in the original network. By using such a surrogate dataset, the adversary will likely choose fewer DGA domains and their shared hosts to simulate a successful attack, compared to the ideal case in perfect knowledge. In other words, the practical cost of launching a small community attack with moderate knowledge is more than the minimal cost of such an attack in the original network. We explore the effect of network size in Section 5.3.4.

Spectral clustering can be evaded using the small community attack, even when the attack cannot be verified by the attacker with a success rate of 75%+. More sophisticated attackers can always evade.

5.3.2 Community Discovery. Unlike graph embedding techniques that lose information about smaller components of the graph, community discovery algorithms do not lose information and can properly handle portions of $\mathcal{G}$ with exactly one edge. Rather than clustering poorly with other small components, they are considered to be separate communities. So the cost of the small community attack is much higher than with graph embeddings because attackers must generate singletons that are small enough to evade classification, forcing the attacker's graph to be disconnected. Therefore, they can evade clustering with the cost of losing their ability to efficiently manage their bots. For example, to evade community discovery in the example presented in Figure 2, an attacker would have to use the modified attack graph G_4' and the drop from $D(G) = 0.5$ to $D(G_4') = 0.25$ is enough to consider the attack cost too high. In the DGA case, this would mean each infection would need its own distinct domain-name generation algorithm, which would be an exceedingly high cost for an attacker. As such, we do not compute results for small community attacks on community discovery.

Community discovery is resistant to the small community attack due to the high costs it would cause the attacker, however, spectral methods and node2vec are more likely to be used by defenders as they result in cleaner clusters and better classification results.

5.3.3 node2vec. The third plot in Figure 9 shows that the small community attack is still possible with node2vec, using aforementioned hyperparameters (Section 5.1). The attack is possible when the number of shared hosts is 1 (the first column except the top cell), and when the number of DGA domains is ≤ 40 (the bottom two rows). Elsewhere, the attack succeeds randomly due to the random walk. In summary, the small community attack is definitely possible with very small component sizes. Compared to SVD, the cost is higher here. For example, the attacker needs to give up $n_v = 578$ unique NXDOMAINs in a day, along with $n_e = 0$, for the small community attack to be successful. But if the attacker is not willing to give up such cost, the small community attack is not guaranteed to succeed given the randomness of neighborhood sampling. However, if a minimal knowledge attacker randomly chooses any n_v and n_e for a small community attack, she will have a 70.65% attack success rate shown by the third plot in Figure 9.

node2vec is susceptible to the small community attack, but with fewer guarantees and higher costs than in the spectral case, due to its inherent randomness. node2vec being used in Pleiades would render the system more resilient against small community attacks.

5.3.4 Small Community Costs. The cost of the small community attack is affected by both the size of network and change in density when the attack is performed.

Size of Network. The network size is related to the number of nodes (hosts and domains) and the number of edges (the query relationship). As a straightforward way to model the network size, we randomly sample the hosts in the ground truth network dataset along with all domains queried. We also keep the same attacker subgraph G, containing the Suppobox DGA community with 10 infected hosts and 618 DGA domains, along with other domains queried by these hosts for the experiment.

Figure 10 shows the small community attack results by sampling 10% to 90% of all hosts. When only 10% of hosts were sampled, the small community attack failed in most areas of the plot. The attack success area increases as the network size gets larger. This means that the cost for small community attack is lower in a larger network than in a smaller network, given the same hyperparameters. A larger network is harder to accurately represent in an embedding, which provides more areas for attackers to hide and evade.

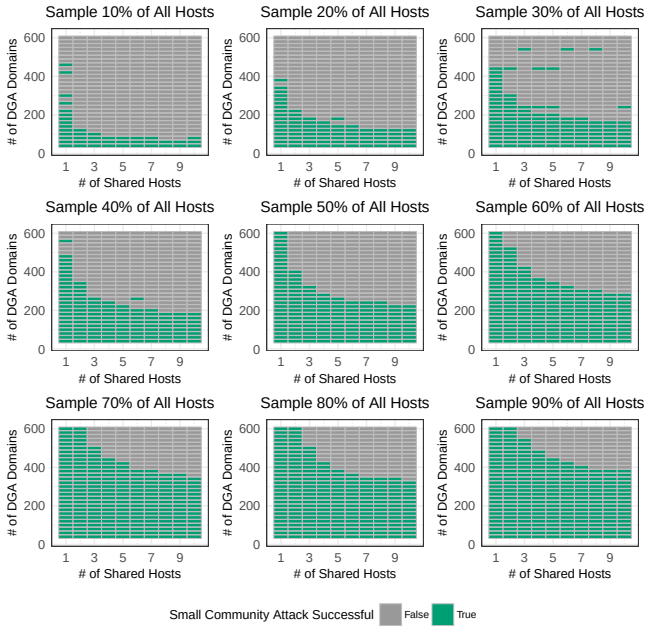


Figure 10: Different sizes of the network dataset.

	Spectral Clustering		
	Density		
Join Death Star	Median	Maximum	Minimum Cost
SVD rank 35	0.078	0.61	0
SVD rank 50	0.11	0.45	0.03
SVD rank 80	0.065	0.26	0.22
SVD rank 100	0.052	0.19	0.29
SVD rank 200	0.0032	0.10	0.38
SVD rank 300	0.026	0.26	0.22
	node2vec		
	Density		
	Median	Maximum	Minimum Cost
Neighborhood Size 6	0.026	0.065	0.415

Table 3: Agility cost of small community attacks under different hyperparameter configurations.

A moderate knowledge level attacker should attempt to acquire a surrogate network dataset smaller than the ground truth network dataset for a safe estimate of their small community attack cost.

Agility Cost. By removing nodes and edges, the attacker loses redundancy. For example, hosts need to query fewer DGA domains, or malware can be allowed fewer malicious actions. We measure the agility cost by the change in density of the attack graph. Density captures the number of edges present in the attack graph over the maximal number of possible edges. In Section 5.3, Equation 2 defines the attack graph density before the small community attack; and Equation 3 defines the density after the attack. Before the attack, $D(G) = 0.48$ for the Suppobox community. For each SVD rank parameter, we record attack configurations that were successful small community attacks as green ares in Figure 7. There are some outliers outside the continuous area. Although these attacks do not make NXDOMAINs join the death star, they move NXDOMAINs to clusters that cannot be predicted with the correct label. To measure the minimum agility cost, we exclude the outliers by only calculating attacker graph density that resulted in joining the death star. Table 3 summarizes the median and maximum attacker graph density in these small community attacks, with the minimum cost represented by the difference between $D(G)$ and $max(D(G'))$. When the SVD rank is 35, the $max(D(G'))$ to join the death star is slightly bigger than $D(G)$, which means there is no cost in launching the small community attack. In this case, the attacker can evade while having *more* connectivity. As the SVD rank increases, the attacker graph density is reduced, which means a successful attack is more costly to the adversary. Also, the minimum cost increases as the SVD rank increases. For example, when the SVD rank is 80, the minimum cost is 0.22, reducing the attack graph density from 0.48 to 0.26. The attacker needs to reduce the number of distinct DGA domains from 618 to 160 to evade, but each domain can be queried by all infected hosts. In comparison, when the SVD rank is 200, the minimum cost is 0.38. The attacker needs to further reduce the number of distinct DGA domains to 60 to evade, with each domain queried by all infected hosts. The attack graph density is reduced from 0.48 to 0.1, losing 79% ($\frac{0.38}{0.48}$) of queries to distinct DGA domains. This means that tuning hyperparameters can increase the small community attack cost and potentially render this attack ineffective.

Similarly, for node2vec, the minimum cost of a certain small community attack is higher than spectral clustering. We compute the attacker graph density only for the white area in Figure 9 without randomness, i.e., the first column and bottom two rows. In contrast to spectral clustering, node2vec requires a much higher minimum cost for a guaranteed small community attack, which indicates that node2vec is more resilient to this attack. The smallest communities in Figure 9 (i.e., the first column and bottom two rows) are likely undersampled, because choosing 15 walks per node and walk length 20 using cluster validity in Section 5.1.3 prefers labeled DGA communities that are relatively bigger, which makes few neighborhood observations for extremely small islands insignificant, and thus allows small community attacks. Note the randomness in the remaining portion of the plot. Since node2vec uses the random walk process to sample the neighborhoods of all nodes, there exists randomness in the neighborhood observations. This shows that the randomness inherent to node2vec makes the attacks succeed at random in the remaining portion of Figure 9. This both suggests a system like Pleiades would benefit from node2vec to reduce the guarantee of attacks, as well as allow a defender to identify if an attacker is evading by chance encounters where the evasion fails over time. While the minimum attack cost is the same with different neighborhood sizes for a guaranteed successful attack, the attack success rate changes. The neighborhood sizes 2, 4, and 6 have attack success rate 65.16%, 60.65%, and 70.65% respectively (Figure 9). We will discuss how we can use different hyperparameters to further reduce the success rate of the small community attack in Section 6.2.

	False Positive Rate			
Model	Pykspa	Gimemo	Suppobox	Murofet
Original	0.32%	0.29%	0%	0%
Model A	1.64%	0.39%	0.10%	0%
Model B	1.62%	0.10%	1.23%	0.30%
Model C	1.46%	1.17%	1.23%	0%

Table 4: False Positive Rate for four DGA families before retraining, and after retraining with three types of noise.

These costs further demonstrate node2vec's superiority over spectral clustering in resisting small community attacks.

6 DEFENSE

Since the noise injection attack and the small community attack violate the fundamental assumptions used by graph clustering techniques, it is very hard to completely eliminate the problem. In this section, we propose two defense techniques that help Pleiades retain its detection capabilities against the two attacks. The first one is to train the classifier with noise, which remediates the noise injection attack to some extent. The second one is to use the small community attack as an adversarial guideline to choose better hyperparameters for graph embeddings, which increases the cost of launching a successful small community attack.

6.1 Training Classifier with Noise

By retraining the classifier, it becomes more resistant to noise that could be injected by the adversary in the unsupervised phase of Pleiades. We used domains from the benign DGA to poison the clusters of malicious DGAs. We retrained the classifier using clusters generated by the noise injection attack variant 1 ("Minimal Benign DGA 1", $m = 1$, Algorithm 1 in Section 5.2) from SVD, yielding *model A*. We tested model A against the adversarial clusters generated by the same noise injection attack under community discovery and node2vec. The first two violins in Figure 6c show that model A increases the overall predicted class probabilities compared to the "After Attack" violins in Figure 6a. In community discovery, the accuracy increased from 2% to 98%; and in node2vec, the accuracy increased from 0.8% to 98%. To summarize, retraining with noisy clusters containing a benign DGA from SVD can remediate the same attack on community discovery and node2vec. We see this same effect even when the noise levels are doubled ($m = 2$, Algorithm 1 in Section 5.2). When models were trained with half the noise ($m = 1$, Algorithm 1 in Section 5.2), they were able to more accurately predict the correct label. Among them, only an average of 7.3% clusters are predicted with the wrong labels, decreased from 100% before retraining.

In comparison with Figure 6a, the average prediction confidence increased significantly. Before retraining, the average prediction confidence of "Minimal Benign DGA 2", "Moderate 2", and "Perfect Long Tail 2" are 10%, 20%, and 20%. After retraining, they increased to 70%, 90%, and 80%, respectively. The accuracy of the models remain roughly the same before and after retraining. However, retraining with noisy clusters increased the false positive rate in most cases (Table 4).

It is important to note that this defense only trains the classifier with noise that has been witnessed. New noise will appear, but the fundamental attack on the unsupervised component remains the same. Therefore, defenders will be alerted by plumetting accuracies in their models. Our defenses are simple and future work should be done to make clustering systems more robust.

6.2 Improving Hyperparameter Selection

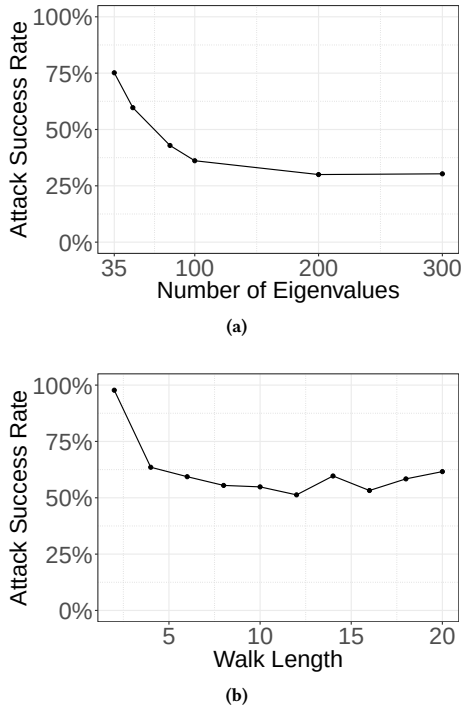


Figure 11: Figure 11a: Using the small community attack to choose the number of eigenvalues for SVD. Figure 11b: Using the small community attack to choose the length of walk for node2vec.

Small community attacks show that the traditional ways of choosing hyperparameters (Section 5.1) is not enough when facing adversaries. Luckily, the small community attack can be used to choose more resistant hyperparameters. We show that better selection can reduce the number of successful small community attack instances from our previous experiments.

We plot the successful attack rate under different number of eigenvalues in Figure 11a. The successful attack rate decreases as the number of eigenvalues computed increases, and the line plateaus after 200 eigenvalues. It means that a defender running Pleiades should select the first 200 eigenvalues, instead of 35 indicated by the scree plot in Figure 4. If we use the small community attack in this way, we can choose better parameters for the system and also know under which parameters the system is vulnerable.

Similarly, for node2vec, using the small community attack to choose hyperparameters can reduce the attack success rate. The

cluster validity metrics suggest we choose neighborhood size six, and walk length of 20. However, if we evaluate the graph clustering using the success rate of the small community attack, these hyperparameters are not optimal. First, for the neighborhood size, Figure 9 shows that a smaller neighborhood size of four introduces a lower attack success rate. Second, we plot the attack success rate under different walk lengths in Figure 11b. This figure shows that a walk length of 12 is preferred over 20, because the former allows 51.29% attack success rate compared to 61.61% of the latter. In other words, the smaller neighborhood size and shorter walk length can tolerate the small community attack better, presumably because they do not oversample larger communities with more distinct neighborhood observations. In other words, smaller communities are not undersampled. We recommend using the small community attack success rate to evaluate the clustering hyperparameter selection, in addition to traditional cluster validity indices.

7 DISCUSSION

We acknowledge that details surrounding the implementation of the attacks are specific to Pleiades, however, the graph representation suggests the attacks may work on other graph-based systems. In this section, we briefly discuss issues to consider to generalize the attacks.

Nodes and edges can be trivially injected or removed in the graph Pleiades uses, which are generated by malware resolving domain names. In other security contexts, the set of injectable/removable nodes varies. It is possible that some nodes and edges must exist in order for certain attack actions to succeed. For example, a phishing email using a malicious attachment requires at least the *read* system call to successfully infect a host, which cannot be removed from the system call graph. On the other hand, it can be difficult to add certain nodes and edges. Therefore, in addition to the anomaly cost (Section 5.2.4) and agility cost (Section 5.3.4), the action of graph manipulation itself has costs depending on the data that underlies the graph representation. This should be carefully considered when generalizing the attacks to other systems and we leave this to be future work. Tighter costs may exist, but our approaches point in a promising direction.

8 CONCLUSIONS

We have demonstrated that generic attacks on graphs can break a real-world system that uses several popular graph-based modeling techniques. These attacks can often be performed by limited adversaries at low cost; however, simple defenses can reduce their effectiveness or likelihood of success. To summarize how defenders can improve their systems: hyperparameter selection should be optimized for reducing the success rate of small community attacks, and retraining can be used to lessen the impact of noise injection attacks. Furthermore, state of the art graph embedding techniques like node2vec appear to be more resistant against small community attacks, which suggests Pleiades and other systems would be harder to adversarially manipulate using node2vec over community finding, or spectral methods (see Figure 9 vs. Figure 7).

9 ACKNOWLEDGMENTS

We thank our anonymous reviewers for their invaluable feedback, and Dr. Rosa Romero-Gómez for her help in visualization. This material is based upon work supported in part by the US Department of Commerce grants no. 2106DEK and 2106DZD, National Science Foundation (NSF) grant no. 2106DGX and Air Force Research Laboratory/Defense Advanced Research Projects Agency grant no. 2106DTX. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Department of Commerce, National Science Foundation, Air Force Research Laboratory, or Defense Advanced Research Projects Agency.

REFERENCES

[1] Accessed in May 2017. Amazon Machine Learning. https://aws.amazon.com/machine-learning. (Accessed in May 2017).
[2] Accessed in May 2017. BigML. https://bigml.com/. (Accessed in May 2017).
[3] Accessed in May 2017. Google Cloud Prediction API. https://cloud.google.com/prediction/docs/. (Accessed in May 2017).
[4] Accessed in May 2017. IOC Bucket. https://www.iocbucket.com/. (Accessed in May 2017).
[5] Accessed in May 2017. Microsoft Azure. https://azure.microsoft.com. (Accessed in May 2017).
[6] Accessed in May 2017. OpenIOC DB. https://openiocdb.com/. (Accessed in May 2017).
[7] Accessed in May 2017. PredictionIO. http://prediction.io. (Accessed in May 2017).
[8] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. 2010. Building a Dynamic Reputation System for DNS. In *USENIX security symposium*. 273–290.
[9] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 491–506.
[10] Johannes Bader. 2017. Domain Generation Algorithms. https://github.com/baderj/domain_generation_algorithms. (2017).
[11] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, Behavior-Based Malware Clustering.. In *NDSS*, Vol. 9. Citeseer, 8–11.
[12] Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. 2013. Is Data Clustering in Adversarial Settings Secure?. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM, 87–98.
[13] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2014. Poisoning Behavioral Malware Clustering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*. ACM, 27–36.
[14] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast Unfolding of Communities in Large Networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.
[15] Mark Braverman, Young Kun Ko, Aviad Rubinstein, and Omri Weinstein. 2017. ETH hardness for densest-k-subgraph with perfect completeness. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1326–1341.
[16] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE.
[17] Raymond B Cattell. 1966. The scree test for the number of factors. *Multivariate behavioral research* 1, 2 (1966), 245–276.
[18] Duen Horng Polo Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-Scale Graph Mining and Inference for Malware Detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 131–142.
[19] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. 2014. DNS Noise: Measuring the Pervasiveness of Disposable Domains in Modern DNS Traffic. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 598–609.
[20] Yizheng Chen, Panagiotis Kintis, Manos Antonakakis, Yacin Nadji, David Dagon, Wenke Lee, and Michael Farrell. 2016. Financial Lower Bounds of Online Advertising Abuse. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 231–254.

[21] Yizheng Chen, Yacin Nadji, Rosa Romero-Gómez, Manos Antonakakis, and David Dagon. 2017. Measuring Network Reputation in the Ad-Bidding Process. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 388–409.

[22] CYBERWARZONE. Accessed in May 2017. 30 Malicious IP List and Block Lists Providers 2015. http://cyberwarzone.com/30-malicious-ip-list-and-block-lists-providers-2015/. (Accessed in May 2017).

[23] Hermit Dave. Accessed in May 2017. Frequency Words in Subtitles. https://github.com/hermitdave/FrequencyWords/tree/master/content/2016/en. (Accessed in May 2017).

[24] Prahlad Fogla and Wenke Lee. 2006. Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques. In *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 59–68.

[25] Amir Globerson and Sam Roweis. 2006. Nightmare at Test Time: Robust Learning by Feature Deletion. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 353–360.

[26] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.

[27] Shuang Hao, Nadeem Ahmed Syed, Nick Feamster, Alexander G Gray, and Sven Krasser. 2009. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. In *USENIX Security Symposium*, Vol. 9.

[28] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. 2011. Adversarial Machine Learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 43–58.

[29] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. 2014. Nazca: Detecting Malware Distribution in Large-Scale Networks. In *NDSS*, Vol. 14. 23–26.

[30] Kurucz, Miklós and Benczúr, András A. 2010. Geographically Organized Small Communities and the Hardness of Clustering Social Networks. In *Data Mining for Social Network Data*. Springer.

[31] Kevin J Lang. 2005. Fixing Two Weaknesses of the Spectral Method. *Advances in Neural Information Processing Systems* (2005).

[32] Angsheng Li and Pan Peng. 2012. The small-community phenomenon in networks. *Mathematical Structures in Computer Science* 22, 03 (2012), 373–407.

[33] Zhou Li and Alina Oprea. 2016. Operational Security Log Analytics for Enterprise Breach Detection. In *First IEEE Cybersecurity Development Conference (SecDev)*.

[34] Kun Liu and Evimaria Terzi. 2008. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 93–106.

[35] Daniel Lowd and Christopher Meek. 2005. Adversarial Learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 641–647.

[36] Daniel Lowd and Christopher Meek. 2005. Good Word Attacks on Statistical Spam Filters. In *CEAS*.

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*. 3111–3119.

[38] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, David Dagon, and Wenke Lee. 2013. Beheading Hydras: Performing Effective Botnet Takedowns. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 121–132.

[39] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, and Wenke Lee. 2013. Connected Colors: Unveiling the Structure of Criminal Networks. In *International Workshop on Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 390–410.

[40] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. 2015. WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths.. In *USENIX Security Symposium*. 1025–1040.

[41] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. 2016. Towards Measuring and Mitigating Social Engineering Software Download Attacks. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 773–789.

[42] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.

[43] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv preprint arXiv:1605.07277* (2016).

[44] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[45] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *NSDI*, Vol. 10. 14.

[46] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. 2015. Segugio: Efficient Behavior-Based Tracking of Malware-Control Domains in Large ISP Networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE, 403–414.

[47] Erzsébet Ravasz, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai, and A-L Barabási. 2002. Hierarchical organization of modularity in metabolic networks. *science* 297, 5586 (2002), 1551–1555.

[48] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic Analysis of Malware Behavior Using Machine Learning. *Journal of Computer Security* 19, 4 (2011), 639–668.

[49] Nedim Rndic and Pavel Laskov. 2014. Practical Evasion of a Learning-Based Classifier: A Case Study. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 197–211.

[50] Marta Sales-Pardo, Roger Guimera, André A Moreira, and Luís A Nunes Amaral. 2007. Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences* 104, 39 (2007), 15224–15229.

[51] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. Avclass: A Tool for Massive Malware Labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 230–253.

[52] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. 2016. I Am Robot:(deep) Learning to Break Semantic Image Captchas. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 388–403.

[53] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF Detection Using Metadata and Structural Features. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 239–248.

[54] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 8–pp.

[55] Kymie MC Tan, Kevin S Killourhy, and Roy A Maxion. 2002. Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 54–73.

[56] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction Apis. In *USENIX Security*.

[57] Ulrike Von Luxburg. 2007. A Tutorial on Spectral Clustering. *Statistics and computing* 17, 4 (2007), 395–416.

[58] David Wagner and Paolo Soto. 2002. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 255–264.

[59] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. 2003. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*. Springer, 91–109.

[60] Gregory L Wittel and Shyhtsun Felix Wu. 2004. On Attacking Statistical Spam Filters. In *CEAS*.

[61] Network X. Accessed in May 2017. Community API. http://perso.crans.org/aynaud/communities/api.html. (Accessed in May 2017).

[62] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically Evading Classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*.

[63] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. 2013. Beehive: Large-Scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 199–208.

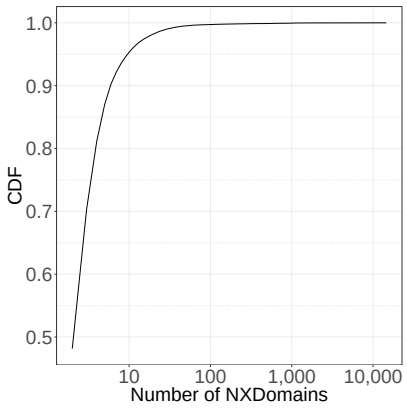


Figure 12: Cumulative distribution of distinct number of NXDOMAINs queried by each host in 12/18/2016.

DGA Family	# of Domains	# of Feature Vectors
Chinad	4,608	18
Corebot	720	18
Gozi	864	72
Locky	360	36
Murofet	54,720	56
Necurs	36,864	18
NewGOZ	18,000	18
PadCrypt	1,728	36
Qadars	3,600	18
Qakbot	180,000	35
Ranbyus	720	18
Sisron	739	19
Symmi	1,152	18
Pykspa	90,300	48
Pykspa	1,190	40
Gimemo	9,144	17
Suppobox	12,846	40

Table 5: DGA families contained within our ground truth dataset.

A UNIQUE DOMAINS QUERIED BY HOSTS

Figure 12 shows the cumulative distribution for distinct number of NXDOMAINs queried by hosts seen on 12/18/2016 in the network datasets from the telecommunication network. The CDF shows that a host querying two distinct NXDOMAINs is at the 48^{th} percentile, and a host querying 10 distinct NXDOMAINs is at the 95^{th} percentile.

B LABELED DGA FAMILIES

We use default parameters to generate different versions of the malware families for 18 different seed dates. The number of domains generated for each malware family is recorded in the top part of Table 5.

C REIMPLEMENTING PLEIADES

We implement a simplified version of the Pleiades DGA detection system. We follow the exact next steps to implement the graph clustering and modeling components of Pleiades.

(1) From the NXDOMAIN query data, we filter out hosts that only queried one domain name in a day (as the authors of Pleiades did).
(2) We construct an association matrix representing the bipartite graph between hosts and the NXDOMAINs they queried. Each row represents one host and each column represents one NXDOMAIN. If host i queried NXDOMAIN j in that day, we assign weight $w_{ij} = 1$ in the matrix. Otherwise, we assign $w_{ij} = 0$. Then, each row is normalized such that the sum of weights is one.
(3) Next, we do Singular Value Decomposition (SVD) over this matrix and keep the first N eigenvalues. For our dataset, we choose $N = 35$ according to the scree plot of Eigenvalues. Figure 4 shows that the Eigenvalues line plateaus after $N >= 35$.
(4) The resulting eigenvectors are used for XMeans clustering.
(5) Once we have the clusters of NXDOMAINs, we extract a feature vector for each cluster, which will be used for classification. We have four feature families: length, entropy, pairwise jaccard distance of character distribution, and pairwise dice distance of bigram distribution. This yields a 36-length feature vector for classification that relies on properties of the domain strings themselves. Please refer to Section 4.1.1 in the original Pleiades paper [9] for further details.
(6) Finally, the classifier uses the feature vectors of the clusters to detect existing, known DGAs and identify never-before-seen DGAs.

To obtain DGA domains as a training dataset for the classifier, we analyzed dynamic malware execution traffic and executed reverse-engineered DGA algorithms. Firstly, we identified NXDOMAINs that were queried by malware md5s by analyzing malware pcaps obtained from a security vendor. We used AVClass [51] to get the malware family labels of those md5s. Using this method, we labeled pykspa, suppobox, and gimemo malware families, which were active in our dataset. We extract one feature family per cluster for these. Secondly, we use reverse engineered DGA domains to compensate limited visibility of DGAs active in the network dataset. Although only Pykspa, Suppobox, and Murofet domains have matches in active clusters, we extract one feature vector for each version's daily domains of 14 DGA families from the reversed engineered DGA domain dataset. Table 5 shows the distribution of the number of features vectors from the reverse engineered DGAs (top) and those seen in clusters (bottom).

We trained the classifier with 17 classes, including 16 malware families and one manually labeled benign class. We labeled benign class from clusters containing mixture of all kinds of benign domains, as well as clusters containing disposable domains (e.g., DNS queries to Anti-Virus online reputation products [19]).

We performed model selection to choose among the following algorithms: Naive Bayes, Linear SVM, Random Forest, Logistic Regression and Stochastic Gradient Descent Classifier. After the analysis of the performance of the different classifiers, we chose to use Random Forest as our classifier. Random Forests are similar to Alternative Decision Trees, a boosted tree-based classifier, which

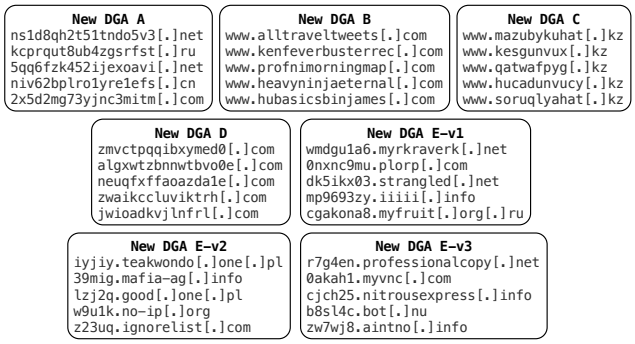


Figure 15: Newly found DGAs.

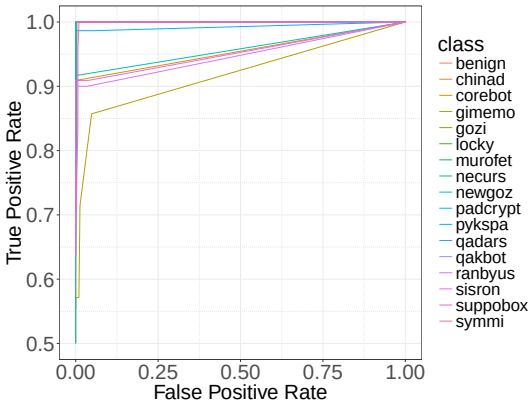


Figure 13: ROC curves for 16 malware DGA classes and one benign class.

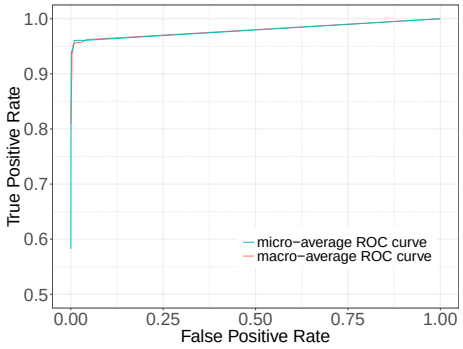


Figure 14: Micro and macro ROC curves.

were used in the original Pleiades paper. We tested our classifier with five fold cross-validation and measured an average accuracy at 96.08%, and a false positive rate of 0.9%. Figure 13 shows the multi-class ROC curves of the classifier performance. Figure 14 shows the micro and macro ROC curves of the multi-class classifier in our implementation of Pleiades.

D CURRENT DGA LANDSCAPE

We ran the DGA detection system over anonymized network traffic from a Recursive DNS server in a telecommunication provider, from December 18, 2016 to December 29, 2016.

Newly Discovered DGAs. We found 12 new DGA malware families. Figure 15 shows 5 of them. New DGA A is classified as similar to the DGA Chinad, with a total of 59,904 domains. The generated domains have a fixed length of 18 characters and use five different tlds: .com, .net, .cn, .biz, and .ru. Chinad has similar characteristics in domain names, but its domain length is 16 characters, and it uses two additional tlds: .info and .org. New DGA B is a dictionary-words DGA that is classified as similar to Gozi. Gozi generates domains by combining words from word lists such as Requests for Comments (RFC), the Ninety Five Theses of Martin Luther in its original Latin text, and GNU General Public License (GNU GPL). In 12 days, we observed 9815 domain names from this DGA, with 10,435 infected hosts. New DGA C is classified as similar to Gimemo. It repeatedly uses bigrams and trigrams as units for composing domain name strings. We found 6,738 domains for new DGA C. Most of the domains from DGA C follow a pattern of consonant-vowel-consonant at the beginning, usually followed by another similar pattern or a sequence of vowel-consonant-vowel, which makes the generated domains appear almost readable. Nevertheless, New DGA C generated domains did not follow the character frequency distribution for any of the languages that use the English or similar alphabets. The length of the generated domains is not fixed but it appears to be around 10 characters with either a character added or removed. New DGA D uses `.com` tld, and second-level labels varying between 12 and 18 characters. New DGA E-v1 iterates through both algorithm-generated second level domains and child labels.

Evasion Attempts in the Wild. The DGAs of qakbot and pykspa provide us with evidence that the malware authors are attempting to avoid or obstruct detection. A special mode of Qakbot is triggered when the malware detects that it is running inside a sandbox environment. Specifically, the seed of the algorithm is appended to generate redundant domains that won't be used as actual C&C. Similary, Pykspa generates two sets of domains based on two different seed sets, which appear identical to a human analyst as if there were only one set of generated domains. Different than Qakbot, in normal operation Pykspa queries both sets of domains, along a list of benign domains. This kind of behavior could be a method to detect analysis efforts. If an analyst sets the environment to provide answers to these "bogus" queries, it could indicate anomaly to the malware. Generating a large number of "fake" domains could also increase the cost of sinkholing the botnets. It makes the sinkholing operation more likely to fail to cover all of the actual C&C domains [38]. These efforts appear to be in their infancy in terms of complexity and effectiveness at this point. If malware authors unleash their creativity in the future, we might come across more elaborate evasion cases that require a lot more effort to identify and detect.

Furthermore, we identify instances of DGAs already evading the classification part of Pleiades by introducing a child label. Our classifier has low confidence for detecting new DGAs B, C, and E-v1. Since there are no DGA domains with child labels in the training dataset, the classifier does not have the requisite knowledge to predict such DGAs. After deploying the classifier for 12 days,

we retrained the classifier with additional DGA families observed from the network. After retraining, our classifier has successfully identified the following new variants with high confidence: DGA E-v2 and DGA E-v3.