SAND2014-18640C

Attacking DBSCAN for Fun and Profit

Jonathan Crussell Philip Kegelmeyer

Abstract

Many security applications depend critically on clustering. However, we do not know of any clustering algorithms that were designed with an adversary in mind. An intelligent adversary may be able to use this to her advantage to subvert the security of the application. Already, adversaries use obfuscation and other techniques to alter the representation of their inputs in feature space to avoid detection. For example, malware is often packed and spam email often mimics normal email. In this work, we investigate a more active attack, in which an adversary attempts to subvert clustering analysis by feeding in carefully crafted data points.

Specifically, in this work we explore how an attacker can subvert DBSCAN, a popular density-based clustering algorithm. We explore a "confidence attack," where an adversary seeks to poison the clusters to the point that the defender loses confidence in the utility of the system. This may result in the system being abandoned, or worse, waste the defender's time investigating false alarms. While our attacks generalize to all DBSCAN-based tools, we focus our evaluation on AnDarwin, a tool designed to detect plagiarized Android apps. We show that an adversary can merge arbitrary clusters by connecting them with "bridges", that even a small number of merges can greatly degrade clustering performance, and that the defender has limited recourse when relying solely on DBSCAN. Finally, we propose a remediation process that uses machine learning and features based on outlier measures that are orthogonal to the underlying clustering problem to detect and remove injected points.

1 Introduction

Clustering is a useful tool for analyzing unlabeled data. It can be used to find plagiarized Android apps [4], to classify network traffic [8], to identify similar queries at search engines [15], and for many other applications.

Clustering is often applied to "found data", as well as "controlled data". In the latter case, the data is collected from physical measurements such as gene expression for gene analysis [7]. In the former case, the origin and integrity of the data is unclear. Thus, an adversary could tamper with a clustering result by creating specially crafted malicious samples to be "found".

In the case of plagiarized Android applications (apps), a plagiarist may seek to subvert the clustering algorithm in at least two ways. First, she can try to manipulate the apps that she copies in such a way that the copied app is no longer similar to the original app. We call this form of attack an *evasion attack*. For example, spam emails will often copy characteristics of normal emails to try to avoid detection. Second, she may seek to manipulate the cluster structure by adding specifically crafted apps (which we call *data mines*). This form of attack may also allow a plagiarizer to evade detection, however, we here explore how it may be used to poison the clustering to undermine confidence in the tool. For this reason, we call this form of attack a *confidence attack*.

In this paper, we explore and evaluate the effectiveness of the proposed confidence attack. First, we describe how an attacker may select an ordering of clusters to merge. In a real-world scenario, we assume that an attacker would have a specific goal, however, we evaluate many orderings to develop intuition about possible attacks and to evaluate which most quickly degrades the accuracy of plagiarism detection. Then, we discuss how *bridges* can be generated using data mines to arbitrarily merge clusters. These bridges span the gaps between clusters, leading the clustering algorithm to interpret the data as a single cluster. Next, we measure how the quality of the clustering degrades as a function of the number of data mines an attacker creates. Finally, we propose an additional remediation phase that the defender can use to prune data mines from her dataset based on outlier measurements.

Our contributions are as follows: 1) we present a methodology for selecting and then merging arbitrary clusters, 2) we evaluate the effectiveness of various attacks in a real-world scenario, 3) we propose metrics for attacker and defender cost and measure the trade-offs, 4) we find DBSCAN alone is insufficient for adversarial settings, and 5) we propose a remediation methodology to remove data mines from a dataset based on outlier measurements.

2 Background

2.1 DBSCAN DBSCAN [9] is a density-based clustering algorithm for large spatial databases. It clusters points using two parameters: T and $MinPts$. T is the distance threshold that determines the size of each point's neighborhood. $MinPts$ is the number of neighbors a point must have in its T-neighborhood to be considered a core point. A cluster is defined as all the points that are *density-reachable* from a core point p. For a point, q, to be *density-reachable* from core point p, it must either be in the T-neighborhood of p or there must be a series of core points $p_0, \ldots, p_n$ such that p_0 is in the T-neighborhood of p, p_i is in the T-neighborhood of $p_i - 1$ $(0 < i <= n)$, and q is in the T-neighborhood of p_n.

DBSCAN is an attractive clustering algorithm for a number of reasons. First, it does not require the number of clusters to be specified ahead of time. Second, combined with locality sensitive hashing (LSH) [14] that can approximately identify the nearest-neighbors of a point in logarithmic time, DBSCAN can cluster all points in linearithmic time.

2.2 AnDarwin AnDarwin [4] is a tool developed to detected cloned Android apps that are distributed through app markets such as Google Play. The majority of apps [1] are available for free but include ads for monetization. Unfortunately for developers, apps are often cloned by plagiarists who alter the app to redirect the ad revenue stream into their own accounts [10].

As described in detail in the original work, AnDarwin detects cloned apps on a large scale, by converting each app to a large number of binary features and then by clustering those feature vectors so tightly that only cloned apps will share a cluster. Although not so noted in the original work, this process for building clusters is equivalent to DBSCAN with a $MinPts$ value of 2, where every point with at least one neighbor is a core point. In the original DBSCAN paper, the authors suggest using values for T and $MinPts$ that represent the "thinnest" cluster that should not be considered noise. This is problematic for AnDarwin, as AnDarwin's goal is to find app plagiarism which could consist of just one original app and one (similar) plagiarized app. If AnDarwin uses a value larger than $MinPts = 2$, then AnDarwin may miss many instances of app plagiarism.

3 Related Work

There are several works that study the affects of adversarial input on machine learning. In a supervised context [5, 12], the problem is often analyzed using game theory. Specifically, these works seek to find an equilibrium between the defender who uses a classifier and one or more attackers. In this work, we focus on unsupervised machine learning. Others have looked at adversarial input on unsupervised machine learning. Notably, Dutrisac et al. [6], describe an attack similar to the one outlined in this paper – the process of bridging the gap between two clusters to merge them. In this work, we explore this process using DBSCAN as the clustering algorithm. We also formalize the attacker cost based on how many points she must generate to bridge gaps. Finally, we explore how effective these attacks are using a dataset and tool with real-world applications and whether the attacks can be remediated.

Biggio et al. [2] develop an *adversarial clustering* theory for performing security evaluations of clustering algorithms including a model for the attacker's goals, knowledge, capabilities, and strategy. They explore two forms of attacks: *cluster poisoning attacks* and *obfuscation attacks*. These are analogous to the confidence and evasion attacks described in the introduction. To demonstrate their framework, they evaluate single-linkage clustering. Specifically, they show how an attacker can use a small number of points, heuristically chosen, to poison the clustering. In this work, we present an closed-form equation for the number of points required to bridge two clusters for DBSCAN and perform an extensive evaluation of the attacks against AnDarwin. Additionally, we show that the attacks apply to density-based clustering algorithms.

DBSCAN's clustering algorithm is similar to the single linkage clustering that is used for agglomerative clustering. A well-known issue with single linkage clustering is the chaining phenomenon. The chaining phenomenon occurs because the algorithm merges two clusters even when there is only a single pair of points that are similar between them. We exploit this weakness when building bridges to span the gap between clusters. Other linkage methods, such as complete-linkage clustering, avoid this phenomenon but may create many smaller clusters.

4 Threat Model

We assume that the attacker can generate arbitrary points in feature space and that the attacker can inject those points into our dataset. For example, in the case of the Android apps, the attacker could generate apps with arbitrary feature sets by copying methods from apps whose features the attacker wishes to include into her app. Since AnDarwin does not perform any dead code analysis, each of these injected methods is treated as a regular feature regardless of whether it is needed for app functionality or not. This exploits the semantic gap between program analysis and program execution [11]. To get her apps into our dataset, the attacker can create an account on a third-party Android market and upload her apps there for us to acquire.

In terms of Biggio et al. [2], the adversary has perfect knowledge. The attacker knows the complete dataset, the feature space, the algorithm, and the algorithm's parameters. We discuss the feasibility of this attacker model in Section 8.1.

Though our methodology applies to all tools based on DBSCAN, to be concrete in the current work we use AnDarwin as a specific application of DBSCAN. This allows us to more concretely discuss the generation of the data mines used to merge clusters. Specifically, we assume that points represent a set of binary features and that points are compared using their Jaccard similarity.

In this work, we explore a *confidence attack* that an attacker might use against a clustering tool to undermine the defender's confidence in the tool and underlying algorithms. If the defender's confidence is undermined, she may not trust the tool's results and/or abandon use of the tool completely.

To provide a concrete objective for the attacker, we investigate how the attacker's injected points degrades the accuracy of plagiarism detection. Specifically, for a given clustering, we can determine whether an app is an original or plagiarizing and compare that label to the label for the same app in the untampered clustering. From these labelings, we can compute the overall accuracy of plagiarism detection in the presence of some number of cluster merges.

5 Methodology

In this section, we describe the two critical components of how an attacker would carry out her attack: 1) the order in which she picks clusters to merge, and 2) the process of generating *data mines* that will cause DBSCAN to merge her selected clusters. Before describing the mechanisms, we first we outline how we identify which apps are originals and which are plagiarizing based on a clustering and then describe the metrics we will use to measure how much the attacker's cluster merges degrade the clustering. Finally, we outline a methodology to remove data mines from the dataset based on outlier measurements.

5.1 Identifying Plagiarism To identify plagiarizing apps, we leverage the owner merging methodology from Gibler et al. [10]. Specifically, we seek to partition apps in a given cluster based on the owner that published the app. We determine ownership in two ways: 1) the developer account name that is associated with the app when it was crawled, and 2) the public key fingerprint for the private key that the owner used to cryptographically sign the app. If two apps share either of these two identifiers, we consider them to be from the same owner. Once we have partitioned a cluster into apps from the same owner, we then assume that the owner with the most apps is the original owner and that all others are plagiarizing. While this may not always be accurate, it does ensure that we do not overestimate the number of plagiarizing apps.

5.2 Clustering Performance To quantify how much the clustering degrades as the attacker merges clusters, we compute four relative performance metrics for the clusterings. These performance metrics are all supervised; they compare the clustering after some number of merges to the original clustering. The first three metrics are generic clustering comparison metrics: Homogeneity, Adjusted Rand Index, and Adjusted Mutual Info. Our last metric is the plagiarism detection accuracy. This is computed using a confusion matrix for the original and plagiarizing labels given to apps as described in the previous section. All four metrics have 1.0 as a perfect score, and make no assumptions about the cluster structure. Both Homogeneity and plagiarism detection accuracy are not normalized with respect to random labeling.

5.3 Merge Ordering Algorithms An attacker performing a confidence attack may choose any order to merge clusters. As stated in Section 4, we assume that the attacker wishes to optimally degrade the accuracy of plagiarism detection. Therefore, we propose the following merge algorithms to develop intuition about how different orderings may affect the degradation of the clustering quality:

- **Random:** Cluster pairs are selected at random.
- **Cluster Size, Decreasing and Increasing:** Cluster pairs are selected based on the size of the merged cluster they would create. For decreasing, this will start by merging the two largest clusters. For increasing, this picks pairs such that the merge produces a cluster of the smallest size possible (not including points to merge the two clusters).
- **Original Size, Decreasing:** Similar to the previous algorithms except instead of using the size of the cluster, the algorithm uses the number of original apps in the cluster.
- **Author Size, Decreasing:** This algorithm starts by finding the author with the most apps across all clusters and merging clusters containing all her apps first. Then, it proceeds to merge in the remaining clusters from biggest to smallest, by cluster size.

- **Nearest-neighbor:** An initial seed cluster is chosen at random and then clusters are chosen in order of decreasing similarity to the seed cluster.

- **Cluster Similarity, Decreasing and Increasing:** Cluster pairs are selected based on their similarity. For decreasing, this will start by merging related clusters before merging unrelated clusters. Note: the similarity of clusters is computed as the minimum similarity of any two points that span the clusters.

- **Greedy Pessimal - Accuracy:** Find the two unmerged clusters that, when merged, degrade the plagiarism detection accuracy the most. Repeat this process until all clusters have been merged.

During the merge ordering algorithms, the similarity is not recomputed after each merger (even though the data mines may influence unmerged clusters' similarities). To ensure that the orderings do not contain obvious redundant merges, the algorithms keep track of which clusters have been merged and will skip pairs that have already been merged.

Due to the computational complexity, we do not try to evaluate all merge orderings. For each of the above algorithms, there may be many merge orderings that can be produced. As a trivial example, different random seeds will lead to completely different random merge orderings. Instead of evaluating all merge orderings, we take one ordering generated from each of the above algorithms as representative of that type of attack. We anticipate that the greedy pessimal algorithm will degrade the plagiarism detection accuracy most quickly, however, it may not produce an optimal ordering. We leave exploring whether there are even more pernicious merge orderings to future work.

5.4 Data Mine Generation To merge two clusters, the adversary must change the dataset so that two previously distinct clusters meet the criteria to be a single cluster (a core point in one cluster is density-reachable from the other). She does this by generating a series of data mines between the two clusters that, to the DBSCAN algorithm, look like core points. Based on the DBSCAN algorithm, this will merge the original two clusters. For now, assume that the $MinPts$ parameter used by DBSCAN to determine the minimum neighborhood size of a core point is 2. This effectively makes every point with at least one neighbor a core point as points are in their own T-neighborhood.

Let p_S and p_T be two points, the start and target points, in different clusters that the attacker wants to merge. To merge these clusters, she must generate a series of $n-1$ (n will be discussed below) data mines $(p_1, \ldots, p_{n-1})$ such that:

$$\begin{aligned} \forall i \in [1, n) : Dist(p_i, p_{i+1}) &\leq T \\ Dist(p_S, p_1) &\leq T \\ Dist(p_{n-1}, p_T) &\leq T \end{aligned} \tag{5.1}$$

For notational convenience, and without loss of generality, let $p_0 = p_S$ and $p_n = p_T$. We can then render Equation 5.1 more compactly as:

$$\forall i \in [0, n) : Dist(p_i, p_{i+1}) \leq T \tag{5.2}$$

Where $Dist$ is an arbitrary distance function and T is the threshold used for DBSCAN to determine the neighborhood size. With a $MinPts$ value of 2, this series of mines merges the two clusters that p_0 and p_n were in, achieving the attacker's goal.

Clearly, the number of data mines $(n-1)$ an attacker must craft is proportional to T: the larger the value of T, the more mines the attacker must generate. If n were sufficiently large, say in the thousands, we may discount this as a too noisy for real-world use. To determine if this is the case, we analyze how the choice of T affects n. To minimize the number of data mines, an adversary should create points such that:

$$\forall i \in [0, n) : Dist(p_i, p_{i+1}) = T \tag{5.3}$$

Figure 1 depicts the geometry of these relationships. In Section 4, we made the assumption that instead of using the distance function, $Dist$, we are using a similarity function, Jaccard Similarity (J), and that each point, p_i, is represented by a set of features. Instead of letting T represent a maximum distance for determining the neighborhood size, let it represent a minimum similarity for the same purpose. Further, we make the worst case (for the attacker) assumption that the two points to be merged have completely disjoint feature sets. Then, the attacker can generate each mine, p_i, using a portion (x) of the features from p_{i-1} and adding a portion $(1-x)$ of the features from p_n:

$$p_i = xp_{i-1} + (1-x)p_n \tag{5.4}$$

Assuming that the adversary knows T, and that p_0 and p_n are approximately the same size ($|p_{i-1}| = |p_n|$ for all i), x depends on how dissimilar the intermediate data mines can be, which is a function of T:

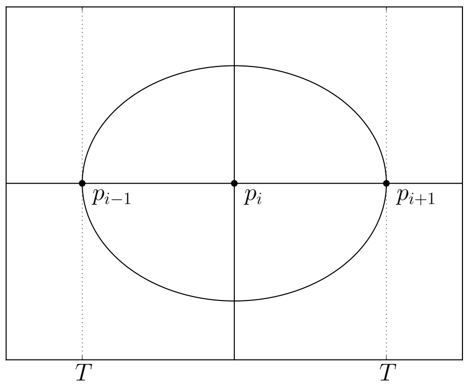


Figure 1: Geometry for a three-point chain. This chain shows how to generate datamines when $MinPts \leq 3$.

$$J(p_{i-1}, p_i) = T = \frac{p_{i-1} \cap (xp_{i-1} + (1-x)p_n)}{p_{i-1} \cup (xp_{i-1} + (1-x)p_n)} \tag{5.5}$$

$$= \frac{|xp_{i-1}|}{|p_{i-1}| + |(1-x)p_n|} = \frac{x}{2-x} \tag{5.6}$$

$$\Rightarrow x = \frac{2T}{T+1} \tag{5.7}$$

Then, the total number of samples needed to generate p_n using Equation 5.4 is determined by the number of times a portion of p_n needs to be added to the original starting point to include all the features of p_n:

$$UBAC(T) = \frac{1}{1-x} - 1 = \frac{1}{1 - \frac{2T}{1+T}} - 1 \tag{5.8}$$

$$= \frac{1+T}{1-T} - 1 \tag{5.9}$$

This equation represents an upper-bound for the attacker cost ($UBAC$) since we assumed that the two points have completely disjoint feature sets. Using the equation, for a threshold of $T = \frac{1}{2}$, an attacker must generate 2 data mines to merge target points p_0 and p_3. For $T = \frac{9}{10}$, 18 data mines must be created.

In the supplementary materials, we show how to remove the assumptions that the executables have a similar number of features (Section A-1.1) and that $MinPts = 2$ (Section A-1.2). In summary, we show how to generate scaling data mines to match up the feature set sizes and derive a two-parameter equation for the number of mines to merge two clusters as a parameter of both T and $MinPts$:

$$UBAC(T, MinPts) = \frac{1 + \sqrt[\frac{MinPts-1}{2}]{T}}{1 - \sqrt[\frac{MinPts-1}{2}]{T}} - 1 \tag{5.10}$$

5.5 Remediation Before applying DBSCAN to her dataset, the defender has a chance to perform sanity checks on the dataset to determine if any apps should be removed. Since data mines are constructed to minimally span the gaps between clusters, we hypothesize that outlier measurements can identify these points. Outlier measurements are commonly used when analyzing data to identify points that are uncharacteristically distinct from others in the dataset. For example, we can identify observations that are statistically unlikely given the population's mean and standard deviation. There are many different outlier measurements, some are based on neighborhood relations, others are based on local densities [3], and others are based on angles between points [13]. Rather than rely on a single outlier measurement to predict whether an app is a data mine or not, we instead propose a supervised approach using an ensemble of outlier measures. Specifically, we propose computing outlier measurements on a data set that we have tampered with, training a classifier to identify the data mines we injected, and then applying the classifier to data sets that may have been tampered with by an adversary. For our initial experiments, we compute the following outlier measurements for each point to use as features for our classifier:

- The number of neighbors in the $T^{\frac{1}{4}}$, $T^{\frac{1}{2}}$, T, T^2, T^3 neighborhoods. Where T is the DBSCAN clustering threshold.
- The angle between the two nearest-neighbors.
- The variance in the angle between all pairs of points in the same cluster (similar to the Angle-base outlier factor [13]).

In general, outlier measurements are fairly computationally expensive, so we select measurements that are tractable for our target data set. However, we can easily add other outlier measurements to our ensemble with only the additional computational cost.

6 Dataset

The full dataset used for AnDarwin [4] consists of 265,359 apps crawled from 17 Android markets. From those apps, AnDarwin extracted a total of 90,144,000 semantic vectors which it then clustered into 2,952,245 features. AnDarwin clustered the apps into 28,495 clusters of which 4,679 clusters contain apps from more than one owner. Ownership is determined using the methodology described in Section 5.1.

We develop our attacks using a subset of the results from AnDarwin. Specifically, we randomly select 273 clusters consisting of 1,394 apps for our experiments. Among these clusters, 229 and 44 clusters contain apps from a single owner and multiple owners, respectively.

7 Evaluation

In this section, we evaluate the effectiveness of the confidence attack. Specifically, we generate a series of cluster merges using each of the ordering algorithms (Section 5.3) and then merge clusters by generating data mines (Section 5.4). For the sake of thoroughness, we evaluate the effectiveness of these attacks to "completion," when the attacker has merged all clusters into a single cluster. In reality, an attacker would likely not perform such an attack and would instead have a budget on the number of data mines or a specific goal (e.g. merge clusters X, Y, and Z).

First, we look at the number of data mines required to perform the attack to "completion." Then, we evaluate how the clustering degrades using our clustering performance metrics. Next, we perform an analysis to detect inadvertent merges. Then, we evaluate a potential defense against this attack: increasing T and $MinPts$, and evaluate the cost to both the defender and the attacker. Finally, we look at two ways to remediate the clusters and explore how well they recover the original plagiarism detection accuracy.

7.1 Cluster Merging In Figure 2, we plot the values of two of the four clustering performance metrics described in Section 5.2 versus the number of data mines injected into the dataset. Interestingly, the number of data mines is quite similar across all algorithms except for Decreasing Cluster Similarity which required approximately 35% more data mines than the other algorithms. For the other merge algorithms, the number of data mines required to merge all clusters is slightly more than half the number of apps in the original dataset (800 versus 1,394). This means that the attacker has to inject a significant number of mines relative to the size of the original dataset to merge all clusters to completion. However, the number of data mines required is not a function of the number of apps in the dataset; it is a function of the number of clusters. Using Equation 5.9 and the DBSCAN parameters of $T = 0.5$ and $MinPts = 2$, the number of mines to required to merge two arbitrary clusters is 2. Therefore, to merge the 273 clusters, a total of $2 * 272$ data mines should be needed. The discrepancy between the theoretical and actual values is due to the fact that not all apps have the same number of features. As a result, additional scaling mines are required (as introduced in Section 5.4) to match up the apps' feature set sizes, increasing the total number of mines to merge two clusters.

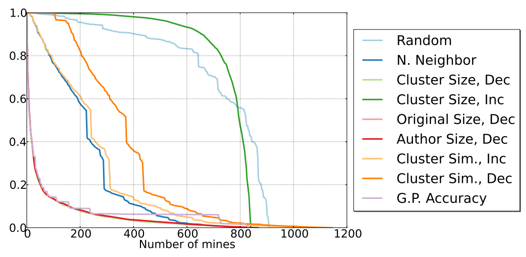
In addition to the number of data mines injected, Figure 2 shows how similar the clustering is after adding some number of data mines to the original clustering. The plots for Homogeneity and Adjusted Mutual Info are available in the supplementary materials (Figure A-2) and are similar to the Adjust Rand Index (ARI) plot although neither degrades as quickly. We can make a number of interesting observations about these plots:

First, the relative ordering of merge algorithms is largely consistent across the generic clustering comparison metrics. Decreasing Cluster Size and Decreasing Original Size tend to do the best while Increasing Cluster Size and Random do the worst, from the attacker's perspective. These metrics are all roughly based on the number of points that are clustered correctly, which degrades the quickest if the larger clusters are merged first.

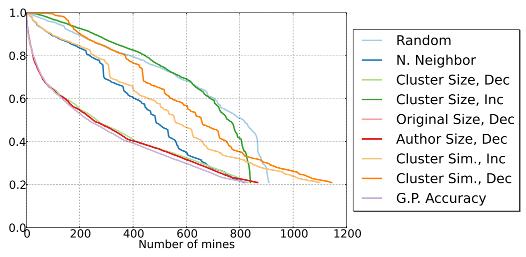
Second, the results for the application-specific metric, the plagiarism detection accuracy, are mostly consistent with the merge algorithm effectiveness as determined by the generic clustering comparison metrics. Noticeably, the Greedy Pessimal algorithm that merges the adversary's best pair at each stage of merging is only just more effective than the Decreasing Cluster and Author Size algorithms.

Finally, the ARI metric seems to be the most sensitive to the cluster poisoning. After only a few hundred mines for the big to small merge ordering, the metric drops to close to zero while the other metrics remain above 0.8. This presents an opportunity for the defender to quickly detect cluster poisoning. If the defender were performing incremental clustering using DBSCAN over a period of time, she could compare today's clustering to previous clusterings to see how much they differ. If the adversary acts too quickly, a large drop in the ARI could alert the defender.

7.2 Attacker and Defender Costs In this section, we explore one possible defense against the cluster merging attack: increasing T and $MinPts$. By increasing T and $MinPts$, the defender can increase the number of data mines the attacker must generate to merge two clusters (Equation 5.10). This increases the cost to the attacker as she will have to generate many more data mines to bridge clusters. However, this is not without cost to the defender. If the defender increases $MinPts$ from 2, she will no longer be able to detect apps that have been copied just once. Drastically increasing $MinPts$ will allow her to only detect frequently copied apps. If the defender increases T, she may miss some



(a) Adjusted Rand Index



(b) Plagiarism Detection Accuracy

Figure 2: Cluster degradation plots. These show how two of the four clustering performance metrics degrade as a function of the number of data mines the attacker has injected into the dataset. From an attacker's perspective, algorithms with less area under the curve are better since they drop the clustering performance quicker.

copies. A defender must balance her own cost against that of the attacker if she is to use this as her defense.

In Table 1, we show the attacker's cost (Equation 5.10) computed for different values of T and $MinPts$. From this table, we can see that the attacker cost does not increase when $MinPts$ is increase from 2 to 3. This is because the chaining geometry always creates mines such that the number of points in the T-neighborhood of a point is 3. At higher values of T and $MinPts$, the attacker cost is relatively high; she must generate more than 50 points to merge two clusters.

	$MinPts$				
T	2	3	5	11	19
0.5	2.00	2.00	4.83	13.45	24.98
0.6	3.00	3.00	6.87	18.59	34.25
0.7	4.67	4.67	10.24	27.05	49.47
0.8	8.00	8.00	16.94	43.82	79.67
0.9	18.00	18.00	36.97	93.92	169.8

Table 1: The attacker's cost for varied T and $MinPts$, as measured by the number of points required to merge two clusters (Equation 5.10).

In Table 2, we show the defender's cost computed for different values of T and $MinPts$. A goal of AnDarwin is to find plagiarized Android apps. Therefore, we measure the defender cost as the number of apps that are no longer detected as plagiarizing with the new parameter values. For this reason, the cost is zero when $T = 0.5$ and $MinPts = 2$. Originally, we classify 196 of the 1,394 apps as plagiarizing. With only minor increases in T and $MinPts$, we can see that the number of plagiarizing apps drops by about 10% (see Section 8.4 for a discussion of whether all these apps are indeed plagiarizing in the first place).

Finally, in Table 3, we compare the attacker's and the defender's costs. Specifically, we compute the defender's cost divided by the attacker's cost:

	$MinPts$				
T	2	3	5	11	19
0.5	N/A	14	28	62	70
0.6	12	25	42	75	81
0.7	39	53	69	92	93
0.8	52	63	75	92	93
0.9	137	146	159	174	174

Table 2: The defender's cost for varied T and $MinPts$, as measured by the number of plagiarizing apps no longer detected as plagiarizing.

$$Rel_{cost}(T, MinPts) = \frac{\text{Missed Plagiarisms}}{UBAC(T, MinPts)} \tag{7.11}$$

When selecting T and $MinPts$, the defender wants to minimize this value while balancing her own cost. If she selects $T = 0.9$ and $MinPts = 19$, she can minimize Rel_{cost} but she will only be able to detect plagiarizing apps for apps that have at least 19 copies and that are all 90% similar. That is, consulting Table 2, she will fail to find 174 plagiarizing apps.

	$MinPts$				
T	2	3	5	11	19
0.5	N/A	7.0	5.80	4.61	2.80
0.6	4.00	8.33	6.11	4.03	2.37
0.7	8.36	11.36	6.74	3.40	1.88
0.8	6.50	7.88	4.43	2.10	1.17
0.9	7.61	8.11	4.30	1.85	1.02

Table 3: $Rel_{cost}(T, MinPts)$ for varied T and $MinPts$.

Based on this analysis, we find that increasing T and $MinPts$ is an insufficient defense for preventing a confidence attacks that seeks to poison the clustering.

7.3 Remediation In order to test our proposed clustering remediation, we partition our dataset into two partitions: A) 700 apps forming 153 clusters and B) 694 apps forming 120 clusters. The partitioning was performed randomly by cluster until the number of apps in each partition was approximately equal. In this section, we explore how the proposed remediation methods change the plagiarism detection accuracy of the latter partition after some number of merges. We generate mines using a random merge ordering and leave exploring whether the outlier measurements are affected by the merge ordering to future work.

Our remediation experiment is conducted as follows. First, we compute outlier measurements for apps in the training partition, varying the number of clusters that were merged before computing the features. We then train a classifier with the presumed number of merges and test the classifier on the testing partition, varying the number of actual merges. We varied the presumed number of merges between 5 and 110 and the actual number of merges between 0 and 110. In both cases, the ranges were inclusive and we computed the results in increments of 5 merges. In Figure 3, we plot the diagonal of the matrix which is the ideal case when the presumed level of tampering matches the actual level of tampering, for each of the partitions of the dataset (and using the other partition for training).

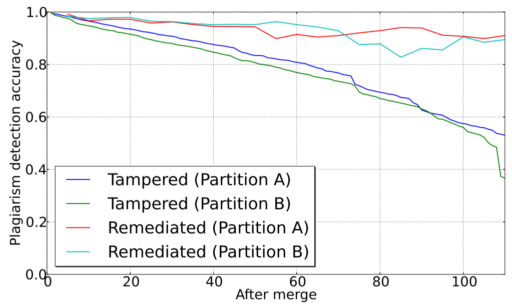


Figure 3: The plagiarism detection accuracy with and without remediation for partitions A and B. For the outlier remediation curve, the presumed number of merges in the training partition matched the actual number of merges in the testing partition. Clusters were ordered using the Random merge algorithm.

From the Figure, we can see that the outlier-based remediation approach is fairly successful at recovering the untampered plagiarism detection accuracy under the ideal circumstance where the presumed amount of tampering matches the actual amount of tampering. In fact, as long as the presumed amount of tampering is less than the actual amount of tampering, the outlier-based remediation approach does well (see Figure A-3 in the supplementary materials). Surprisingly, training a classifier on data with just five merges leads to near-perfect remediation, regardless of the number of merges present in the actual dataset.

8 Discussion

8.1 Attack Feasibility In our threat model, we assumed that the attacker has perfect knowledge: she knows the complete dataset, the feature space, the algorithm, and the algorithm's parameters. We explore this scenario as the worst case behavior both for the sake of general insight into the problem and for scoping the specific vulnerability of clustering tools based on DBSCAN. Further, in the worst case, the attacker is an insider who, as an insider, does have perfect knowledge.

In the specific case of our AnDarwin application, the dataset is comprised of publicly available applications crawled from Android Markets; this could be replicated by an attacker. Even if such an attacker's collection did not perfectly match the defender's collection, our mechanism for generating bridges between clusters still applies. Admittedly, the attack would likely be suboptimal; characterizing the degree of suboptimality as a function of matching the defender's data collection is an interesting problem for future work.

8.2 Merge Algorithms In Section 7, we evaluated the clustering performance degradation using one instance of each of the merge algorithms described in Section 5.3. However, for each merge algorithm, there are many instances of the attack. Some of these instances will outperform the others in terms of how quickly they degrade the clustering performance. Ultimately, there are "optimal" attacks that degrade the clustering performance the fastest for a particular metric with the fewest points. Discovering truly optimal attacks is a combinatorial problem as every ordering of pairs of clusters must be considered. Our greedy algorithm attempts to approximate the optimal ordering.

8.3 Suboptimal Data Mines In Section 7.3, we evaluated how well our outlier-based remediation approach was able to remove data mines from the dataset to recover the original plagiarism detection accuracy. We built and tested our classifiers for data mine detection using outlier features computed for clusters merged with the fewest number of mines possible. However, an adversary may not attempt to minimize the number of data mines she uses. In fact, based on the results of our remediation experiments, the adversary should not use optimally-placed data mines to avoid detection. An interesting problem for future work is to explore the degree of suboptimality required to evade the outlier-based remediation approach. Two potential approaches include simply generating data mines paths with higher values of T and $MinPts$, making more dense bridges, and adding jitter to “widen” the bridges.

8.4 Plagiarizing apps For evaluating the defender cost of altering the DBSCAN parameters T and $MinPts$ in Section 7.2, we assumed that the original clustering was correct. Specifically, we assumed that all the apps that are identified as plagiarizing with the original parameters are indeed plagiarizing. This, however, is not necessarily the case. False alarms in the original clustering will increase the defender's cost even though they are false alarms. In fact, we could be improving the clustering with the different values of T and $MinPts$. Knowing the ground truth clustering of this dataset is outside the scope of this work and the evaluation was designed to measure, in the worst case, the cost to the defender when T and $MinPts$ are set to increase the attackers cost.

9 Conclusion

In this work we showed how to subvert DBSCAN with a confidence attack that poisons the clustering. We illustrated our approach with AnDarwin, a tool designed to detect plagiarized Android apps. We showed how an attacker can craft apps to bridge the gap between clusters that leads to their merger. As the attacker generates more bridges, we showed how the quality of the clustering degrades. Next, we investigated adjusting the DBSCAN clustering parameters to prevent this form of attack. We found that defenders should use caution when relying on the clusterings produced by DBSCAN for security applications. Finally, we proposed an outlier-based approach to detect data mines within a dataset.

Acknowledgements Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] AppBrain. Free vs paid Android apps. http://www.appbrain.com/stats/free-and-paid-android-applications.

[2] Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. Is data clustering in adversarial settings secure? In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, pages 87–98. ACM, 2013.

[3] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *ACM Sigmod Record*, volume 29, pages 93–104. ACM, 2000.

[4] Jonathan Crussell, Clint Gibler, and Hao Chen. An-Darwin: Scalable detection of semantically similar Android applications. In *Computer Security–ESORICS 2013*, pages 182–199. Springer Berlin Heidelberg, 2013.

[5] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 99–108. ACM, 2004.

[6] JG Dutrisac and David B Skillicorn. Hiding clusters in adversarial settings. In *IEEE International Conference on Intelligence and Security Informatics, 2008*, pages 185–187. IEEE, 2008.

[7] Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[8] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, pages 281–286. ACM, 2006.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

[10] Clint Gibler, Ryan Stevens, Jonathan Crussell, Hao Chen, Hui Zang, and Heesook Choi. AdRob: Examining the landscape and impact of Android application plagiarism. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, pages 431–444. ACM, 2013.

[11] Suman Jana and Vitaly Shmatikov. Abusing file processing in malware detectors for fun and profit. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 80–94. IEEE, 2012.

[12] Murat Kantarcıoğlu, Bowei Xi, and Chris Clifton. Classifier evaluation and attribute selection against active adversaries. *Data Mining and Knowledge Discovery*, 22(1-2):291–335, 2011.

[13] Hans-Peter Kriegel, Arthur Zimek, et al. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–452. ACM, 2008.

[14] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *Signal Processing Magazine, IEEE*, 25(2):128–131, 2008.

[15] Ji-Rong Wen, Jian-Yun Nie, and Hong-Jiang Zhang. Query clustering using user logs. *ACM Transactions on Information Systems*, 20(1):59–81, 2002.