Is Data Clustering in Adversarial Settings Secure?

Battista Biggio
Università di Cagliari
Piazza d'Armi
09123, Cagliari, Italy
battista.biggio@diee.unica.it

Ignazio Pillai
Università di Cagliari
Piazza d'Armi
09123, Cagliari, Italy
pillai@diee.unica.it

Samuel Rota Bulò
FBK-irst
Via Sommarive, 18
38123, Trento, Italy
rotabulo@fbk.eu

Davide Ariu
Università di Cagliari
Piazza d'Armi
09123, Cagliari, Italy
davide.ariu@diee.unica.it

Marcello Pelillo
Università Ca' Foscari di Venezia
Via Torino, 155
30172 Venezia-Mestre
pelillo@dais.unive.it

Fabio Roli
Università di Cagliari
Piazza d'Armi
09123, Cagliari, Italy
roli@diee.unica.it

ABSTRACT

Clustering algorithms have been increasingly adopted in security applications to spot dangerous or illicit activities. However, they have not been originally devised to deal with deliberate attack attempts that may aim to subvert the clustering process itself. Whether clustering can be safely adopted in such settings remains thus questionable. In this work we propose a general framework that allows one to identify potential attacks against clustering algorithms, and to evaluate their impact, by making specific assumptions on the adversary's goal, knowledge of the attacked system, and capabilities of manipulating the input data. We show that an attacker may significantly poison the whole clustering process by adding a relatively small percentage of attack samples to the input data, and that some attack samples may be obfuscated to be hidden within some existing clusters. We present a case study on single-linkage hierarchical clustering, and report experiments on clustering of malware samples and handwritten digits.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive software (*e.g.*, viruses, worms, Trojan horses); G.3 [**Probability and Statistics**]: Statistical computing; I.5.1 [**Models**]: Statistical; I.5.2 [**Design Methodology**]: Clustering design and evaluation; I.5.3 [**Clustering**]: Algorithms

General Terms

Security, Clustering.

Keywords

Adversarial learning, Unsupervised Learning, Clustering, Security Evaluation, Computer Security, Malware Detection.

1. INTRODUCTION

Clustering algorithms are nowadays a fundamental tool for the data analysts as they allow them to make inference and gain insights on large sets of unlabeled data. Applications of clustering span across a large number of different domains, such as market segmentation [14, 26], classification of web pages [10], and image segmentation [12]. In the specific domain of computer security, clustering algorithms have been recently exploited to solve plenty of different problems, *e.g.*, spotting fast-flux domains in DNS traffic [24], gaining useful insights on tools and sources of attacks against Internet websites [25], detecting repackaged Android applications [16] and (Android) mobile malware [9], and even automatically generating signatures for anti-virus software to enable detection of HTTP-based malware [23].

In many of the aforementioned scenarios, a large amount of data is often collected *in the wild*, in an unsupervised manner. For instance, malware samples are often collected from the Internet, by means of honeypots, *i.e.*, machines that purposely expose known vulnerabilities to be infected by malware [28], or other ad hoc services, like VirusTotal.[1] Given that these scenarios are intrinsically *adversarial*, it may thus be possible for an attacker to inject carefully crafted samples into the collected data in order to subvert the clustering process, and make the inferred knowledge useless. This raises the issue of evaluating the *security* of clustering algorithms against carefully designed attacks, and proposing suitable countermeasures, when required. It is worth noting that results from the literature of *clustering stability* [29] can not be directly exploited to this end, since the noise induced by adversarial manipulations is not generally *stochastic* but specifically targeted against the clustering algorithm.

The problem of learning in adversarial environments has recently gained increasing popularity, and relevant research has been done especially in the area of supervised learning algorithms for classification [6, 8, 17, 3], and regression [13]. On the other hand, to the best of our knowledge only few works have implicitly addressed the issue of security evaluation related to the application of clustering algorithms in adversarial settings through the definition of suitable attacks,

[1] `http://virustotal.com`

while we are not aware of any work that proposes specific countermeasures to attacks against clustering algorithms.

The problem of devising specific attacks to subvert the clustering process was first brought to light by *Dutrisac and Skillicorn* [11, 27]. They pointed out that some points can be easily *hidden* within an existing cluster by forming a *fringe* cluster, *i.e.*, by placing such points sufficiently close the border of the existing cluster. They further devised an attack that consists of adding points in between two clusters to merge them, based on the notion of *bridging*. Despite this pioneering attempts, a framework for the systematic security evaluation of clustering algorithms in adversarial settings is still missing, as well as a more general theory that takes into account the presence of the adversary to develop more *secure* clustering algorithms.

In this work we aim to take a first step to fill in this gap, by proposing a framework for the security evaluation of clustering algorithms, which allows us to consider several potential attack scenarios, and to devise the corresponding attacks, in a more systematic manner. Our framework, inspired from previous work on the security evaluation of supervised learning algorithms [6, 17, 3], is grounded on a model of the attacker that allows one to make specific assumptions on the adversary's goal, knowledge of the attacked system, and capability of manipulating the input data, and to subsequently formalize a corresponding *optimal* attack strategy. This work is thus explicitly intended to provide a cornerstone for the development of an *adversarial clustering* theory, that should in turn foster research in this area.

The proposed framework for security evaluation is presented in Sect. 2. In Sect. 3 we derive worst-case attacks in which the attacker has perfect knowledge of the attacked system. In particular, we formalize the notion of (worst-case) *poisoning* and *obfuscation* attacks against a clustering algorithm, respectively in Sects. 3.1 and 3.2. In the former case, the adversary aims at maximally compromising the clustering output by injecting a number of carefully designed attack samples, whereas in the latter one, she tries to *hide* some attack samples into an existing cluster by manipulating their feature values, without significantly altering the clustering output on the rest of the data. As a case study, we evaluate the security of the single-linkage hierarchical clustering against poisoning and obfuscation attacks, in Sect. 4. The underlying reason is simply that the single-linkage hierarchical clustering has been widely used in security-related applications [4, 16, 23, 24]. To cope with the computational problem of deriving an optimal attack, in Sects. 4.1 and 4.2 we propose heuristic approaches that serve well our purposes. Finally, in Sect. 5 we conduct synthetic and real-world experiments that demonstrate the effectiveness of the proposed attacks, and subsequently discuss limitations and future extensions of our work in Sect. 6.

2. ATTACKING CLUSTERING

In this section we present our framework to analyze the security of clustering approaches from an adversarial pattern recognition perspective. It is grounded on a model of the adversary that can be exploited to identify and devise attacks against clustering algorithms. Our framework is inspired by a previous work focused on attacking (supervised) machine learning algorithms [6], and it relies on an attack taxonomy similar to the one proposed in [17, 3]. As in [6], the adversary's model entails the definition of the adversary's goal, knowledge of the attacked system, and capability of manipulating the input data, according to well-defined guidelines.

Before moving into the details of our framework, we introduce some notation. Clustering is the problem of organizing a set of data points into groups referred to as *clusters* in a way that some criteria is satisfied. A clustering algorithm can thus be formalized in terms of a function f mapping a given dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{\mathsf{n}}$ to a clustering result $\mathcal{C} = f(\mathcal{D})$. We do not specify the mathematical structure of $\mathcal{C}$ at this point of our discussion because there exist different types of clustering requiring different representations, while our model applies to any of them. Indeed, $\mathcal{C}$ might be a hard or soft partition of $\mathcal{D}$ delivered by partitional clusterings algorithms such as k-means, fuzzy c-means or normalized cuts, or it could be a more general family of subsets of $\mathcal{D}$ such as the one delivered by the dominant sets clustering algorithm [22], or it can even be a parametrized hierarchy of subsets (*e.g.*, linkage-type clustering algorithms).

2.1 Adversary's goal

Similarly to [6, 17, 3], the adversary's goal can be defined according to the attack specificity, and the security violation pursued by the adversary. The attack specificity can be *targeted*, if it affects solely the clustering of a given subset of samples; or *indiscriminate*, if it potentially affects the clustering of any sample. Security violations can instead affect the *integrity* or the *availability* of a system, or the *privacy* of its users.

Integrity violations amount to performing some malicious activity without significantly compromising the normal system operation. In the supervised learning setting [17, 3], they are defined as attacks aiming at camouflaging some malicious samples (*e.g.*, spam emails) to evade detection, without affecting the classification of legitimate samples. In the unsupervised setting, however, this definition can not be generally applied since the notion of malicious or legitimate class is not generally available. Therefore, we regard integrity violations as attacks aiming at deflecting the grouping for specific samples, while limiting the changes to the original clustering. For instance, an attacker may obfuscate some samples to hide them in a different cluster, without excessively altering the initial clusters.

Availability violations aim to compromise the functionality of the system by causing a denial of service. In the supervised setting, this translates into causing the largest possible classification error [17, 6, 7]. According to the same rationale, in the unsupervised setting we can consider attacks that significantly affect the clustering process by worsening its result as much as possible.

Finally, *privacy violations* may allow the adversary to obtain information about the system's users from the clustered data by reverse-engineering the clustering process.

2.2 Adversary's knowledge

The adversary can have different degrees of knowledge of the attacked system. They can be defined by making specific assumptions on the points (*k.i*)-(*k.iv*) described below.

(*k.i*) **Knowledge of the data $\mathcal{D}$**: The adversary might know the data $\mathcal{D}$ or only a portion of it. More realistically, she may not know $\mathcal{D}$ exactly, but she may be able to obtain a *surrogate* dataset sampled from the same distribution as $\mathcal{D}$.

In practice, this can be obtained by collecting samples from the same source from which samples in $\mathcal{D}$ were collected; *e.g.*, honeypots for malware samples [28].

($k.ii$) **Knowledge of the feature space**: The adversary could know how features are extracted from each sample. Similarly to the previous case, she may know how to compute the whole feature set, or only a subset of the features.

($k.iii$) **Knowledge of the algorithm**: The adversary's could be aware of the targeted clustering algorithm and how it organizes the data into clusters; *e.g.*, the criterion used to determine the cluster set from a *hierarchy* in hierarchical clustering.

($k.iv$) **Knowledge of the algorithm's parameters**: The attacker may even know how the parameters of the clustering algorithm have been initialized (if any).

Perfect knowledge. The worst-case scenario in which the attacker has full knowledge of the attacked system, is usually referred to as *perfect knowledge* case [6, 7, 19, 8, 17, 3]. In our case, this amounts to knowing: ($k.i$) the data, ($k.ii$) the feature representation, ($k.iii$) the clustering algorithm, and ($k.iv$) its initialization (if any).

2.3 Adversary's capability

The adversary's capability defines how and to what extent the attacker can control the clustering process. In the supervised setting [17, 6], the attacker can exercise a *causative* or *exploratory* influence, depending on whether she can control training and test data, or only test data. In the case of clustering, however, there is not a test phase in which some data has to be classified. Accordingly, the adversary may only exercise a *causative* influence by manipulating part of the data to be clustered.[2] This is often the case, though, since this data is typically collected in an unsupervised manner.

We thus consider a scenario in which the attacker can add a maximum number of (potentially manipulated) samples to the dataset $\mathcal{D}$. This is realistic in several practical cases, *e.g.*, in the case of malware collected through *honeypots* [28], where the adversary may easily send (few) samples without having access to the rest of the data. This amounts to controlling a (small) percentage of the input data. An additional constraint may be given in terms of a maximum amount of modifications that can be done to the attack samples. In fact, to preserve their malicious functionality, malicious samples like spam emails or malware code may not be manipulated in an unconstrained manner. Such a constraint can be encoded by a suitable distance measure between the original, non-manipulated attack samples and the manipulated ones, as in [6, 20, 17, 3].

2.4 Attack strategy

Once the adversary's goal, knowledge and capabilities have been defined, one can determine an *optimal* attack strategy that specifies how to manipulate the data to meet the adversary's goal, under the restriction given by the adversary's knowledge and capabilities. In formal terms, we denote by Θ the *knowledge space* of the adversary. Elements of Θ hold information about the dataset $\mathcal{D}$, the clustering algorithm f, and its parametrization, according to ($k.i$)-k($.iv$). To model the degree of knowledge of the adversary we consider a probability distribution μ over Θ. The entropy of μ indicates the level of uncertainty of the attacker. For example, if we consider a perfect-knowledge scenario like the one addressed in the next section, we have that μ is a Dirac measure peaked on an element $\theta_0 \in \Theta$ (with null entropy), where $\theta_0 = (\mathcal{D}, f, \cdots)$ holds the information about the dataset, the algorithm and any other of the informations listed in Sect.2.2. Further, we assume that the adversary is given a set of attack samples $\mathcal{A}$ that can be manipulated before being added to the original set $\mathcal{D}$. We model with the function $\Omega(\mathcal{A})$ the family of sample sets that the attacker can generate according to her capability as a function of the set of initial attack samples $\mathcal{A}$. The set $\mathcal{A}$ can be empty, if the attack samples are not required to fulfill any constraint on their malicious functionality, *i.e.*, they can be generated from scratch (as we will see in the case of *poisoning* attacks). Finally, the adversary's goal given the knowledge $\theta \in \Theta$ is expressed in terms of an objective function $g(\mathcal{A}'; \theta) \in \mathbb{R}$ that evaluates how close the modified data set integrating the (potentially manipulated) attack samples $\mathcal{A}'$ is to the adversary's goal. In summary, the attack strategy boils down to finding a solution to the following optimization problem:

$$\begin{array}{rl} \text{maximize} & \mathbb{E}_{\theta\sim\mu}[g(\mathcal{A}'; \theta)] \\ \text{s.t.} & \mathcal{A}' \in \Omega(\mathcal{A})\,. \end{array} \tag{1}$$

where $\mathbb{E}_{\theta\sim\mu}[\cdot]$ denotes the expectation with respect to θ being sampled according to the distribution μ.

3. PERFECT KNOWLEDGE ATTACKS

In this section we provide examples of worst-case integrity and availability security violations in which the attacker has perfect knowledge of the system, as described in Sect. 2.2. We respectively refer to them as *poisoning* and *obfuscation* attacks. Since the attacker has no uncertainty about the system, we set $\mu = \delta_{\{\theta_0\}}$, where δ is the Dirac measure and θ_0 represents exact knowledge of the system. The expectation in (1) thus yields $g(\mathcal{A}'; \theta_0)$.

3.1 Poisoning attacks

Similarly to poisoning attacks against supervised learning algorithms [7, 19], we define poisoning attacks against clustering algorithms as attacks in which the data is tainted to maximally worsen the clustering result. The adversary's goal thus amounts to violating the system's *availability* by *indiscriminately* altering the clustering output on any data point. To this end, the adversary may aim at maximizing a given distance measure between the clustering $\mathcal{C}$ obtained from the original data $\mathcal{D}$ (in the absence of attack) and the clustering $\mathcal{C}' = f_{\mathcal{D}}(\mathcal{D}')$ obtained by running the clustering algorithm on the contaminated data $\mathcal{D}'$, and restricting the result to the samples in $\mathcal{D}$, *i.e.*, $f_{\mathcal{D}} = \pi_{\mathcal{D}} \circ f$ where $\pi_{\mathcal{D}}$ is a projection operator that restricts the clustering output to the data samples in $\mathcal{D}$. We regard the tainted data $\mathcal{D}'$ as the union of the original dataset $\mathcal{D}$ with the attack samples in $\mathcal{A}'$, *i.e.*, $\mathcal{D}' = \mathcal{D} \cup \mathcal{A}'$. The goal can thus be written as $g(\mathcal{A}'; \theta_0) = d_c(\mathcal{C}, f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}'))$, where d_c is the chosen distance measure between clusterings. For instance, if f is a partitional clustering algorithm, any clustering result can be represented in terms of a matrix $\mathtt{Y} \in \mathbb{R}^{\mathtt{n}\times\mathtt{k}}$, each $(i, k)^{\text{th}}$ component being the probability that the i^{th} sample is assigned

[2]One may however think of an exploratory attack to a clustering algorithm as an attack in which the adversary aims to gain information on the clustering algorithm itself, although she may not necessarily manipulate any data to this end.

to the k^{th} cluster. Under this setting, a possible distance measure between clusterings is given by:

$$d_{\mathrm{c}}(\mathtt{Y}, \mathtt{Y}') = \|\mathtt{Y}\mathtt{Y}^\top - \mathtt{Y}'\mathtt{Y}'^\top\|_F \,, \tag{2}$$

where $\|\cdot\|_F$ is the Frobenius norm. The components of the matrix $\mathtt{Y}\mathtt{Y}^\top$ represent the probability of two samples to belong to the same cluster. When $\mathtt{Y}$ is binary, thus encoding hard clustering assignments, this distance counts the number of times two samples have been clustered together in one clustering and not in the other, or vice versa. In general, depending on the nature of the clustering result, other ad-hoc distance measures can be adopted.

As mentioned in Sect. 2.3, we assume that the attacker can inject a maximum of m data points into the original data $\mathcal{D}$, *i.e.* $|\mathcal{A}'| \le \mathsf{m}$. This realistically limits the adversary to manipulate only a given, potentially small fraction of the dataset. Clearly, the value of m will be considered as a parameter in our evaluation to investigate the robustness of the given clustering algorithm against an increasing control of the adversary over the data. We further define a box constraint on the feature values $\boldsymbol{x}_{\text{lb}} \le \boldsymbol{x} \le \mathbf{x}_{\text{ub}}$, to restrict the attack points to lie in some fixed interval (*e.g.*, the smallest box that includes all the data points). Hence, we define the function Ω encoding the adversary's capabilities as follows:

$$\Omega_{\mathrm{p}} = \left\{ \{\boldsymbol{a}_i'\}_{i=1}^{\mathsf{m}} \subset \mathbb{R}^{\mathsf{d}} \,:\, \boldsymbol{x}_{\text{lb}} \le \boldsymbol{a}_i' \le \boldsymbol{x}_{\text{ub}} \text{ for } i = 1, \cdots, \mathsf{m} \right\} .$$

Note that Ω depends on a set of target samples $\mathcal{A}$ in (1), but since $\mathcal{A}$ is empty in this case, we write Ω_{p} instead of $\Omega(\emptyset)$. The reason is simply that, in the case of a poisoning attack, the attacker aims to find a set of attack samples that do not have to carry out any specific malicious activity besides worsening the clustering process.

In summary, the optimal attack strategy under the aforementioned hypothesis amounts to solving the following optimization problem derived from (1):

$$\begin{array}{rl} \text{maximize} & d_{\mathrm{c}}(\mathcal{C}, f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}')) \\ \text{s.t.} & \mathcal{A}' \in \Omega_{\mathrm{p}} \,. \end{array} \tag{3}$$

3.2 Obfuscation attacks

Obfuscation attacks are violations of the system *integrity* through *targeted* attacks. The adversary's goal here is to *hide* a given set of initial attack samples $\mathcal{A}$ within some existing clusters by *obfuscating* their content, possibly without altering the clustering results for the other samples. We denote by $\mathcal{C}^t$ the target clustering involving samples in $\mathcal{D} \cup \mathcal{A}'$ the attacker is aiming to, being $\mathcal{A}'$ the set of obfuscated attack samples. With the intent to preserve the clustering result $\mathcal{C}$ on the original data samples, we impose that $\pi_{\mathcal{D}}(\mathcal{C}^t) = \mathcal{C}$, while the cluster assignments for the samples in $\mathcal{A}'$ are freely determined by the attacker. As opposed to the poisoning attack, here the attacker is interested in pushing the final clustering towards the target clustering and therefore her intention is to minimize the distance between $\mathcal{C}^t$ and $\mathcal{C}' = f(\mathcal{D} \cup \mathcal{A}')$. Accordingly, the goal function g in this case is defined as $g(\mathcal{A}'; \theta_0) = -d(\mathcal{C}^t, f(\mathcal{D} \cup \mathcal{A}'))$.

As for the adversary's capability, we assume that the attacker can perturb the target samples in $\mathcal{A}$ to some maximum extent. We model this by imposing that $d_{\mathrm{s}}(\mathcal{A}, \mathcal{A}') \le d_{\max}$, where d_{s} is a measure of divergence between the two sets of samples $\mathcal{A}$ and $\mathcal{A}'$ and $d_{\max}$ is a nonnegative real scalar. Consequently, the function Ω representing the attacker's capacity is given by

$$\Omega_o(\mathcal{A}) = \left\{ \{\boldsymbol{a}_i'\}_{i=1}^{|\mathcal{A}|} \,:\, d_{\mathrm{s}}(\mathcal{A}, \mathcal{A}') \le d_{\max} \right\} .$$

The distance d_{s} can be defined in different ways. For instance, in the next section we define $d_{\mathrm{s}}(\mathcal{A}, \mathcal{A}')$ as the largest Euclidean distance among corresponding elements in A and A', *i.e.*,

$$d_{\mathrm{s}}(\mathcal{A}, \mathcal{A}') = \max_{i=1,\ldots,\mathsf{m}} \|\boldsymbol{a}_i - \boldsymbol{a}_i'\|_2 \tag{4}$$

where we assume $\mathcal{A} = \{\boldsymbol{a}_i\}_{i=1}^{\mathsf{m}}$ and $\mathcal{A}' = \{\boldsymbol{a}_i'\}_{i=1}^{\mathsf{m}}$. This choice allows us to bound the divergence between the original target samples in $\mathcal{A}$ and the manipulated ones, as typically done in adversarial learning [20, 17, 8, 6].

In summary, the attack strategy in the case of obfuscation attacks can be obtained as the solution of the following optimization program derived from (1):

$$\begin{array}{rl} \text{minimize} & d_{\mathrm{c}}(\mathcal{C}^t, f(\mathcal{D} \cup \mathcal{A}')) \\ \text{s.t.} & \mathcal{A}' \in \Omega_{\mathrm{o}}(\mathcal{A}) \,. \end{array} \tag{5}$$

4. A CASE STUDY ON SINGLE-LINKAGE HIERARCHICAL CLUSTERING

In this section we solve a particular instance of the optimization problems (3) and (5), corresponding respectively to the poisoning and obfuscation attacks described in Sects. 3.1 and 3.2, against the single-linkage hierarchical clustering. The motivation behind this specific choice of clustering algorithm is that, as mentioned in Sect. 1, it has been frequently exploited in security-sensitive tasks [4, 16, 23, 24].

Single-linkage hierarchical clustering is a bottom-up algorithm that produces a *hierarchy* of clusterings, as any other hierarchical *agglomerative* clustering algorithm [18]. The hierarchy is represented by a *dendrogram*, *i.e.*, a tree-like data structure showing the sequence of cluster fusion together with the distance at which each fusion took place. To obtain a given partitioning of the data into clusters, the dendrogram has to be *cut* at a certain height. The leaves that form a connected sub-graph after the cut are considered part of the same cluster. Depending on the chosen distance between clusters (*linkage* criterion), different variants of hierarchical clustering can be defined. In the *single-linkage* variant, the distance between any two clusters $\mathcal{C}_1, \mathcal{C}_2$ is defined as the minimum Euclidean distance between all pairs of samples in $\mathcal{C}_1 \times \mathcal{C}_2$.

For both poisoning and obfuscation attacks, we will model the clustering output as a binary matrix $\mathtt{Y} \in \{0,1\}^{\mathsf{n} \times \mathsf{k}}$, indicating the sample-to-cluster assignments (see Sect. 3.1). Consequently, we can make use of the distance measure d_c between clusterings defined in Eq. (2). However, to obtain a given set of clusters from the dendrogram obtained by the single-linkage clustering algorithm, we will have to specify an appropriate *cut* criterion.

4.1 Poisoning attacks

For poisoning attacks against single-linkage hierarchical clustering, we aim to solve the optimization problem given by Eq. (3). As already mentioned, since the clustering is expressed in terms of a hierarchy, we have to determine a suitable dendrogram cut in order to model the clustering output

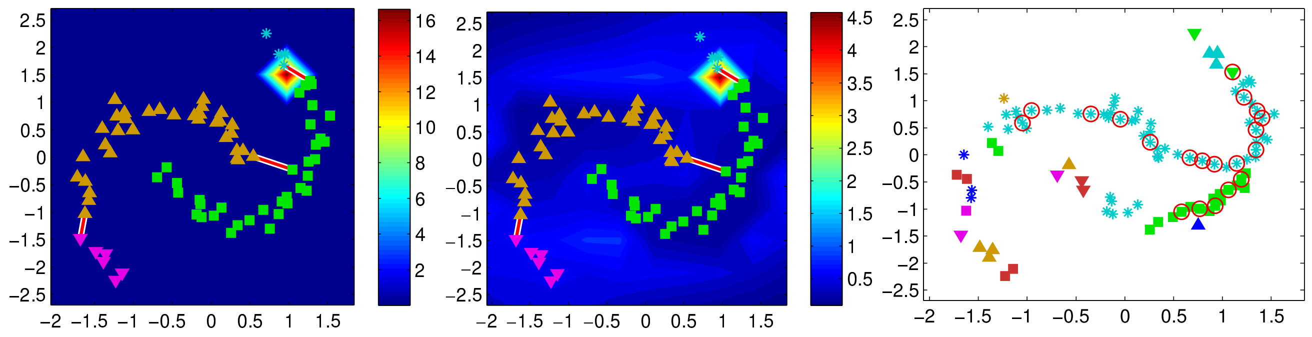
Figure 1: Poisoning single-linkage hierarchical clustering. In each plot, samples belonging to different clusters are represented with different markers and colors. The left and middle plot show the initial partitioning of the given 100 data points into $k = 4$ clusters. The objective function of Eq. 3 (shown in colors) for our greedy attack ($|\mathcal{A}'| = 1$) is respectively computed with hard (left plot) and soft assignments (middle plot), *i.e.*, with binary $\mathtt{Y}$ and posterior estimates. The $k - 1 = 3$ *bridges* obtained from the dendrogram are highlighted with red lines. The rightmost plot shows how the partitioning changes after $\mathsf{m} = 20$ attack samples (highlighted with red circles) have been greedily added.

as a binary matrix $\mathtt{Y}$. In this case, we assume that the clustering algorithm selects the cut, *i.e.*, the number of clusters, that achieves the minimum distance between the clustering obtained in the absence of attack $\mathcal{C}$ and the one induced by the cut, *i.e.*, $\min d_c(\mathcal{C}, f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}'))$. Although this may not be a realistic *cut* criterion, as the ideal clustering $\mathcal{C}$ is not known to the clustering algorithm, this worst-case choice for the adversary gives us the minimum performance degradation incurred by the clustering algorithm under attack.

Let us now discuss how Problem (3) can be solved. First, note that it is not possible to predict analytically how the clustering output $\mathtt{Y}'$ changes as the set of attack samples $\mathcal{A}'$ is altered, since hierarchical clustering does not have a tractable, underlying analytical interpretation.[3] One possible answer consists in a stochastic exploration of the solution space (*e.g.* by simulated annealing). This is essentially done by perturbing the input data $\mathcal{A}'$ a number of times, and evaluating the corresponding values of the objective function by running the clustering algorithm (as a black box) on $\mathcal{D} \cup \mathcal{A}'$. The set $\mathcal{A}'$ that provides the highest objective value is eventually retained. However, to find an optimal configuration of attack samples $\mathcal{A}'$, one should repeat this procedure a very large number of times. To reduce computational complexity, one may thus consider efficient search heuristics specifically tailored to the considered clustering algorithm.

For the above reason, we consider a greedy optimization approach where the attacker aims at finding a local maximum of the objective function by adding one attack sample at a time, *i.e.*, $|\mathcal{A}'| = \mathsf{m} = 1$. In this case, we can more easily understand how the objective function changes as the inserted attack point varies, and define a suitable heuristic approach. An example is shown in the leftmost plot of Fig. 1. This plot shows that the objective function exhibits a global maximum when the attack point is added in between clusters that are sufficiently *close* to each other. The reason is that, when added in such a location, the attack point operates as a *bridge*, causing the two clusters to be merged in a single cluster, and the objective function to increase.

Bridge-based heuristic search. Based on this observation, we devised a search heuristic that considers only $k - 1$ potential attack samples, being k the actual number of clusters found by the single-linkage hierarchical clustering at a given dendrogram cut. In particular, we only considered the $k - 1$ points lying in between the connections that have been cut to separate the k given clusters from the top of the hierarchy, highlighted in our example in the leftmost plot of Fig. 1. These connections can be directly obtained from the dendrogram, *i.e.*, we do not have to run any post-processing algorithm on the clustering result. Thus, one is only required to evaluate the objective function $k - 1$ times for selecting the best attack point. We will refer to this approach as *Bridge (Best)* in Sect. 5.1. The rightmost plot in Fig. 1 shows the effect of our greedy attack after that $\mathsf{m} = 20$ attack points have been inserted. Note how the initial clusters are fragmented into smaller clusters that tend to contain points which originally belonged to different clusters.

Approximating $\mathtt{Y}'$. To further reduce the computational complexity of our approach, *i.e.*, to avoid re-computing the clustering and the corresponding value of the objective function $k - 1$ times for each attack point, we consider another heuristic approach. The underlying idea is simply to select the attack sample (among the $k - 1$ *bridges* suggested by our bridge-based heuristic search) that lies in between the largest clusters. In particular, we assume that the attack point will effectively merge the two adjacent clusters, and thus modify $\mathtt{Y}'$ accordingly (without re-estimating its real value by re-running the clustering algorithm). To this end, for each point belonging to one of the two clusters, we set to 1 (0) the value of $\mathtt{Y}'$ corresponding to the first (second) cluster. Once the estimated $\mathtt{Y}'$ is computed, we evaluate the objective function using the estimated $\mathtt{Y}'$, and select the attack point that maximizes its value. We will refer to this approach as *Bridge (Hard)* in Sect. 5.1.

Approximating $\mathtt{Y}'$ with soft clustering assignments. Finally, we discuss another variation to the latter discussed heuristic approach, which we will refer to as *Bridge (Soft)*, in Sect. 5.1. The problem arises from the fact that our objective function exhibits really *abrupt* variations, since it is

[3]In general, even if the clustering algorithm has a clearer mathematical formulation, it is not guaranteed that a good analytical prediction can be found. For instance, though k-means clustering is well-understood mathematically, its variability to different initializations makes it almost impossible to reliably predict how its output may change due to data perturbation.

computed on hard cluster assignments (*i.e.*, binary matrices $\mathtt{Y}'$). Accordingly, adding a single attack point at a time may not reveal *connections* that can potentially merge large clusters after few attack iterations, *i.e.*, using more than one attack sample. To address this issue, we approximate $\mathtt{Y}'$ with soft clustering assignments. To this end, the element y'_{ik} of $\mathtt{Y}'$ is estimated as the posterior probability of point $\boldsymbol{x}_i$ belonging to cluster c_k, *i.e.*, $y'_{ik} = p(c_k|\boldsymbol{x}_i) = p(\boldsymbol{x}_i|c_k)p(c_k)/p(\boldsymbol{x}_i)$. The prior $p(c_k)$ is estimated as the number of samples belonging to c_k divided by the total number of samples, the likelihood $p(\boldsymbol{x}_i|c_k)$ is estimated with a Gaussian Kernel Density Estimator (KDE) with bandwidth parameter h:

$$p(\boldsymbol{x}_i|c_k) = \frac{1}{|c_k|}\sum_{\boldsymbol{x}_j \in c_k} \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{h}\right) \quad , \qquad (6)$$

and the evidence $p(\boldsymbol{x}_i)$ is obtained by marginalization over the given set of clusters.

Worth noting, for too small values of h, the posterior estimates tend to the same value, *i.e.*, each point is likely to be assigned to any cluster with the same probability. When h is too high, instead, each point is assigned to one cluster, and the objective function thus equals that corresponding to the original hard assignments. In our experiments we simply avoid these limit cases by selecting a value of h comparable to the average distance between all possible pairs of samples in the dataset, which gave reasonable results.

An example of the smoother approximation of the objective function provided by this heuristic is shown in the middle plot of Fig. 1. Besides, this technique also provides a *reliable* approximation of the true objective: although its values are significantly re-scaled, the global maximum is still found in the same location. The smooth variations that characterize the approximated objective influence the choice of the best candidate attack point. In fact, attack points lying on *bridges* that may potentially connect larger clusters after some attack iterations may be sometimes preferred to attack points that can directly connect smaller and closer clusters. This may lead to a larger increase in the true objective function as the number of injected attack points increases.

4.2 Obfuscation attacks

In this section we solve (5) assuming the worst-case (perfect-knowledge) scenario against the single-linkage clustering algorithm. Recall that the attacker's goal in this case is to manipulate a given set of non-obfuscated samples $\mathcal{A}$ such that they are clustered according to a desired configuration, *e.g.*, together with points in an existing, given cluster, without altering significantly the initial clustering that would be obtained in the absence of manipulated attacks.

As in the previous case, to represent the output of the clustering algorithm as a binary matrix $\mathtt{Y}$ representing clustering assignments, and thus compute d_c as given by Eq. 2, we have to define a suitable criterion for cutting the dendrogram. Similarly to poisoning attacks, we define an advantageous criterion for the clustering algorithm, that gives us the lowest performance degradation incurred under this attack: we select the dendrogram cut that minimizes $d_c(\mathcal{C}^\star, f(\mathcal{D} \cup \mathcal{A}'))$, where $\mathcal{C}^\star$ represents the optimal clustering that would be obtained including the non-manipulated attack samples, *i.e.*, $\mathcal{C}^\star = f(\mathcal{D} \cup \mathcal{A})$. The reason is that, to better contrast an obfuscation attack, the clustering algorithm should try to keep the attack points corresponding to the non-manipulated set $\mathcal{A}$ into their original clusters. For instance, in the case of malware clustering, non-obfuscated malware may easily end up in a well-defined cluster, and, thus, it may be subsequently categorized in a well-behaved malware family. While the adversary tries to manipulate malware to have it clustered differently, the best solution for the clustering algorithm would be to obtain the same clusters that would be obtained in the absence of attack manipulation.

We derive a simple heuristic to get an approximate solution of (5) assuming d_s to be defined as in (4). We assume that, for each sample $\boldsymbol{a}_i \in \mathcal{A}$, the attacker selects the closest sample $\boldsymbol{d}_i \in \mathcal{D}$ belonging to the cluster to which $\boldsymbol{a}_i$ should belong to, according to the attacker's desired clustering $\mathcal{C}^t$. To meets the constraint given by Ω_o in Eq. 5, the attacker then determines for each $\boldsymbol{a}_i \in \mathcal{A}$ a new sample $\boldsymbol{a}'_i \in \mathcal{A}$ along the line connecting $\boldsymbol{a}_i$ and $\boldsymbol{d}_i$ in a way not to exceed the maximum distance d_max from $\boldsymbol{a}_i$, *i.e.*, $\boldsymbol{a}'_i = \boldsymbol{a}_i + \alpha(\boldsymbol{d}_i - \boldsymbol{a}_i)$, where $\alpha = \min(1, d_\mathrm{max}/\|\boldsymbol{d}_i - \boldsymbol{a}_i\|_2)$.

5. EXPERIMENTS

We present here some experiments to evaluate the effectiveness of the poisoning and obfuscation attacks devised in Sect. 4 against the single-linkage hierarchical clustering algorithm, under perfect knowledge of the attacked system.

5.1 Experiments on poisoning attacks

For the poisoning attack, we consider three distinct cases: a two-dimensional artificial data set, a realistic application example on malware clustering, and a task in which we aim to cluster together distinct handwritten digits.

5.1.1 Artificial data

We consider here the standard two-dimensional banana-shaped dataset from PRTools,[4] for which a particular instance is shown in Fig. 1 (right and middle plot). We fix the number of initial clusters to $k = 4$, which yields our original clustering $\mathcal{C}$ in the absence of attack.

We repeat the experiment five times, each time by randomly sampling 80 data points. In each run, we add up to $\mathsf{m} = 20$ attack samples, that simulates a scenario in which the adversary can control up to 20% of the data. As described in Sect. 4.1, the attack proceeds greedily by adding one sample at a time. After adding each attack sample, we allow the clustering algorithm to change the number of clusters from a minimum of 2 to a maximum of 50. The criterion used to determine the number of clusters is to minimize the distance of the current partitioning with the clustering in the absence of attack, as explained in details in Sect. 4.1.

We consider five attack strategies, described in the following.

Random: the attack point is selected at random in the minimum box that encloses the data.

Random (Best): $k-1$ attack points are selected at random, being k the actual number of clusters at a given attack iteration. Then, the objective function is evaluated for each point, and the best one is chosen.

Bridge (Best): The $k-1$ bridges suggested by our heuristic approach are evaluated, and the best one is chosen.

[4]`http://prtools.org`

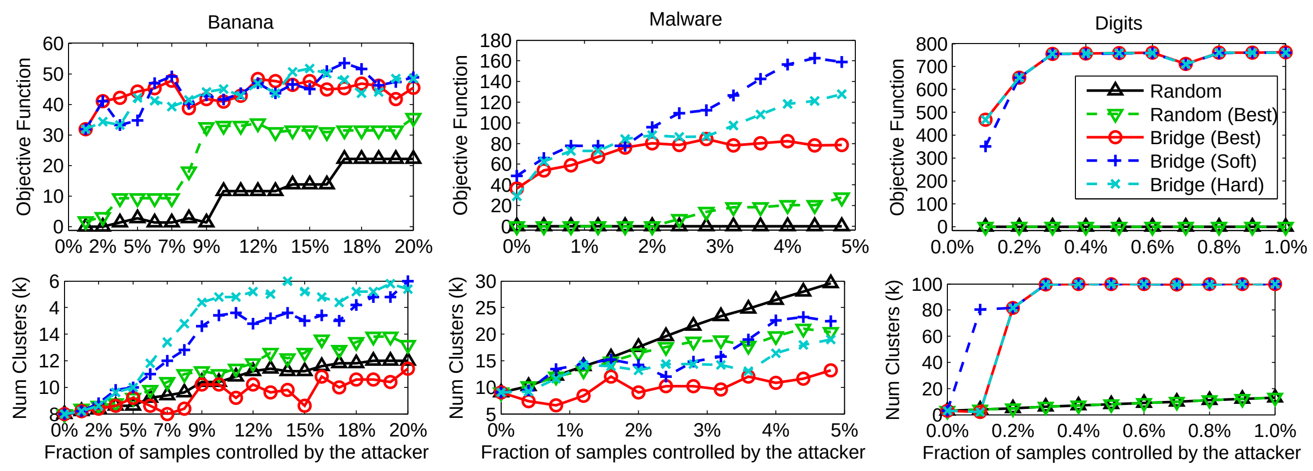

Figure 2: Results for the poisoning attack averaged over five runs on the Banana-shaped dataset (first column), the Malware dataset (second column), and the Digit dataset (third column). Top plots show the variation of the objective function $d_c(f(\mathcal{D}), f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}'))$ as the fraction of samples controlled by the adversary increases. Bottom plots report the number of clusters selected after the insertion of each attack sample.

	Banana (20%)		Malware (5%)		Digits (1%)	
	Split	*Merge*	*Split*	*Merge*	*Split*	*Merge*
Random	1.15 ± 0.22	1.29 ± 0.06	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Random (Best)	1.40 ± 0.34	1.54 ± 0.30	1.00 ± 0.00	1.34 ± 0.39	1.00 ± 0.00	1.00 ± 0.00
Bridge (Best)	2.40 ± 0.60	1.40 ± 0.23	1.49 ± 0.23	1.31 ± 0.17	33.9 ± 0.15	1.02 ± 0.00
Bridge (Soft)	3.85 ± 1.35	1.22 ± 0.11	2.76 ± 0.84	1.12 ± 0.09	33.9 ± 0.15	1.02 ± 0.00
Bridge (Hard)	3.75 ± 1.43	1.21 ± 0.23	2.41 ± 0.73	1.10 ± 0.10	34.0 ± 0.00	1.02 ± 0.00

Table 1: Split and Merge averaged values and standard deviations for the Banana-shaped dataset (at 20% poisoning), the Malware dataset (at 5% poisoning), and the Digit dataset (at 1% poisoning).

Bridge (Hard): The $k-1$ bridges are evaluated here by predicting the clustering output $\mathtt{Y}'$ as discussed in Sect. 4.1 (*i.e.*, assuming that the corresponding clusters will be merged), using hard clustering assignments.

Bridge (Soft): This is the same strategy as *Bridge (Hard)*, except for the fact that we consider soft clustering assignments when modifying $\mathtt{Y}'$. To this end, as discussed in Sect. 4.1, we use a Gaussian KDE. We set the kernel bandwidth h as the average distance between each possible pair of samples in the data. On average, $h \approx 2$ in each run.

It is worth remarking that *Random (Best)* and *Bridge (Best)* require the objective function to be evaluated $k-1$ times at each iteration to select the best candidate attack sample. This means that the clustering algorithm has to be run $k-1$ times at each step. Instead, the other methods do not require us to re-run the clustering algorithm to select the attack point. Their complexity is therefore significantly lower than the aforementioned methods.

The results averaged over the five runs are reported in Fig. 2 (first column). From the top plot one may appreciate how the methods based on the *bridge*-based heuristics achieve similar values of the objective function, while clearly outperforming the *random*-based methods. Further, as reasonably expected, *Random (Best)* outperforms *Random* since it considers the best point over $k-1$ attempts. Nevertheless, even selecting a random attack sample, in this case, turned out to significantly affect the clustering results.

The bottom plot provides us a better understanding of how the attack effectively works. The main effect is indeed to *fragment* the original clusters into a high number of smaller clusters. In particular, after the insertion of $\mathtt{m} = 20$ data points, *i.e.*, when 20% of the data is controlled by the attacker, the selected number of clusters increases from 4 to about 7-14 clusters depending on the considered method.

To further clarify the effect of the attack on the clustering algorithm, we consider two measures referred to as *Split* and *Merge* in Table 5.1, which are given as follows. Let $\mathcal{C}$ and $\mathcal{C}'$ be the initial and the final clustering restricted to elements in $\mathcal{D}$, respectively, and let $\mathtt{C}$ be a binary matrix, each entry $\mathtt{C}_{kk'}$ indicating the co-occurrence of at least one sample in the kth cluster of $\mathcal{C}$ and in the k'th cluster of $\mathcal{C}'$. Then, the above measures are given as:

$$\text{Split} = \operatorname*{mean}_i \sum_j \mathtt{C}_{ij} \, , \; \text{Merge} = \operatorname*{mean}_j \sum_i \mathtt{C}_{ij} \, .$$

Intuitively, *split* quantifies to what extent the initial clusters are fragmented across different final clusters, while *merge* quantifies to what extent the final clusters contain samples that originally belonged to different initial clusters.

From Table 5.1, it can be appreciated how, for the most effective attacks, *i.e.*, *Bridge (Soft)* and *Bridge (Hard)*, the initial clusters are split into approximately 3.8 clusters, while the final clusters merge approximately 1.2 initial clusters, on average. This clarifies how the proposed attack eventually compromises the initial clustering: it tends to fragment the initial clusters into smaller ones, and to merge together final clusters which originally came from different clusters. *Bridge (Best)* tends instead to induce a lower number of final clus-

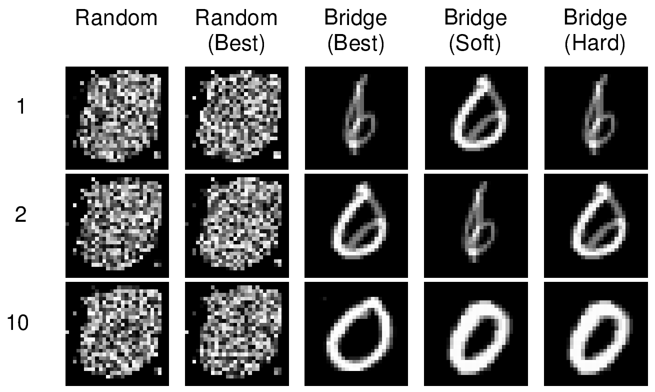

Figure 3: Attack samples produced by the five strategies at iterations 1, 2 and 10, for the digit data.

ters, *i.e.*, the clustering algorithm tends to merge more final clusters than splitting initial ones. However, this is not the optimal choice according to the attacker's goal.

5.1.2 Malware clustering

We consider here a more realistic application example involving malware clustering, and in particular a simplified version of the algorithm for behavioral malware clustering proposed in [23]. The ultimate goal of this approach is to obtain malware clusters that can aid the automatic generation of high quality network signatures, which can be used in turn to detect botnet command-and-control (C&C) and other malware-generated communications at the network perimeter. With respect to the original algorithm, we made the following simplifications:

(a) we consider only the first of the two clustering steps carried out by the original system. The algorithm proposed in [23] clusters samples through two consecutive stages, named *coarse-grain* and *fine-grain* clustering, respectively. Here, we just focus on the *coarse-grain* clustering, which is based on a set of numeric features.

(b) We consider a subset of six statistical features (out of the seven used by the original algorithm). They are: (1) number of GET requests; (2) number of POST requests; (3) average length of the URLs; (4) average number of parameters in the request; (5) average amount of data sent by POST requests; and (6) average length of the response. We exclude the seventh feature, *i.e.*, the total number of HTTP requests, as it is redundant with respect to the first and the second feature. All feature values are re-scaled in $[0, 1]$ as in the original work.

(c) We use the single-linkage hierarchical clustering instead of the BIRCH algorithm [30], since this modification does not significantly affect the quality of the clustering results, as the authors demonstrated in [23].

For the purpose of this evaluation, we use a subset of 1,000 samples taken from *Dataset 1* of [23]. This dataset consists of distinct malware samples (no duplicates) collected during March 2010 from a number of different malware sources, including MWCollect [1], Malfease [2], and commercial malware feeds. As in the previous setting, we repeat the experiments five times, by randomly selecting a subset of 475 samples from the available set of $1,000$ malware data in each run. The initial set of clusters $\mathcal{C}$, as in [23], is selected as the partitioning that minimizes the value of the Davies-Bouldin Index (DBI) [15], a measure that characterizes dispersion and closeness of clusters. We consider the cuts of the initial dendrogram that yield from 2 to 25 clusters, and choose the one corresponding to the minimum DBI. This yields approximately 9 clusters in each run. While the attack proceeds, the clustering algorithm can choose a number of clusters ranging from 2 to 50. The attacker can inject up to 25 attack samples, that amounts to controlling up to 5% of the data. The value of h for the KDE used in *Bridge (Soft)* is set as the average distance between pairs of samples, which turns out to be approximately 0.2 in each run.

Results are shown in Fig. 2 (second column). The effect of the attack is essentially the same as in the previous experiments on the Banana-shaped data, although here there is a significant difference among the performances of the *bridge*-based methods. In particular, *Bridge (Soft)* gradually outperforms the other approaches as the fraction of injected samples approaches 5%. The reason is that, as qualitatively discussed in Sect. 4.1, this heuristic approach tends to bridge clusters which are too far to be bridged with a single attack point, and are thus disregarded by *Bridge (Best)* and not always chosen by *Bridge (Hard)*. It is also worth noting that, in this case, the Random approach is totally ineffective. In particular, no change in the objective function is observed for this method, and the number of clusters increases linearly as the attack proceeds. This means simply that the clustering algorithm produces a new cluster for each newly-injected attack point, making the attack totally ineffective. The behavior exhibited by the different attack strategies is also confirmed by the Split and Merge values reported in Table 5.1. Here, the most effective methods, *i.e.*, again *Bridge (Soft)* and *Bridge (Hard)*, split the 3 initial clusters each into 2.7 and 2.4 final clusters, on average, yielding a total number of clusters of about 20-25 clusters. Similarly to the previous experiments, *Bridge (Best)* yields a lower number of final clusters, as it induces more the clustering algorithm to cluster together samples that originally belonged to different initial clusters.

5.1.3 Handwritten digits

We finally repeat the experiments described in the previous sections on the MNIST handwritten digit data [21].[5] In this dataset, each digit is size-normalized and centered, and represented as a grayscale image of 28×28 pixels. Each pixel is raster-scan ordered and its value is directly considered as a feature. The dimensionality of the feature space is thus 784, a much higher value than that considered in the previous cases. We further normalize each feature (pixel) in $[0, 1]$ by dividing its value by 255.

We focus here on a subset of data consisting of the three digits '0', '1', and '6'. To obtain three initial clusters, each representing one of the considered digits, we first compute the average digit for each class (*i.e.*, the average '0', '1', and '6'), and then select 700 samples per class, by retaining the closest samples to the corresponding average digit. We repeat the experiments five times, each time by randomly selecting 330 samples per digit from the corresponding set of 700 pre-selected samples. While the attack proceeds, the clustering algorithm can choose a number of clusters ranging from 2 to 100. We assume that the attacker can inject up to 10 attack samples, that amounts to controlling up to 1% of

[5]This dataset is publicly available in Matlab format at `http://cs.nyu.edu/~roweis/data.html`.

the data. The value of h for the KDE used in *Bridge (Soft)* is set as in the previous case, based on the average distance between all pairs of samples. For this dataset, it turns out that $h \approx 1$ in each run.

Results are shown in Fig. 2 (third column). With respect to the previous experiments on the Banana-shaped data, and on the Malware data, the results here are significantly different. In particular, note how the *Random* and *Random (Best)* approaches are totally ineffective here. Similarly to the previous case in malware clustering, the clustering algorithm essentially defeats the attack influence by creating a new cluster for each attack sample. The underlying reason is that, in this case, the feature space has a very high dimensionality, and, thus, sampling only $k-1$ points at random is not enough to find a suitable attack point. In other words, if an attack sample is not very well crafted, it may be easily isolated from the rest of the data. Although increasing the dimensionality may thus seem a suitable countermeasure to protect clustering against random attacks, this drastically increases its vulnerability to well designed attack samples. Note indeed how the clustering is already significantly worsened when the adversary only controls a fraction as small as of 0.2% of the data. In fact, the number of final clusters raises immediately to the maximum allowed number of 100. This is also clarified in Table 5.1, where it can be appreciated how the initial clusters are fragmented into an average of 33 final clusters for the *bridge*-based methods. Note however that, in this case, the final clusters are almost *pure*, *i.e.*, the attack algorithm does not succeed in merging together samples coming from different initial clusters.

In Fig. 3 we also show some of the attack samples that are produced by the five attack strategies, at different attack iterations. The *random*-based attacks clearly produce very noisy images which yield a completely ineffective attack, as already mentioned. Instead, the initial attacks considered by *bridge*-based methods (at iteration 1 and 2) resemble effectively the digits corresponding to the two initial clusters that they aim to connect ('0' and '6', and '1' and'6'). Since the attack completely destroys the three initial clusters after very few attack samples have been added, at later iterations (*e.g.*, iteration 10), the *bridge*-based methods tend to enforce some connection within the cluster belonging to the '0' digit, probably trying to merge some of the final clusters together. However, since the maximum number of allowed clusters has been already reached, no further improvement is observed in the objective function.

5.2 Experiments on obfuscation attacks

For the obfuscation attack, we present an experiment on handwritten digits, using again the MNIST digit data described in Sect. 5.1.3.

5.2.1 *Handwritten digits*

We consider the same initial clusters of Sect. 5.1.3, consisting of 330 samples for each of the following digits: '0', '1', and '6'. As in the previous case, we average the results over five runs, each time selecting the initial 330 samples per cluster from the pre-selected sets of 700 samples per digit. In this case, however, we consider a further initial cluster of 100 samples corresponding to the digit '3' (which are also randomly sampled from a pre-selected set of 700 samples of '3', chosen with the same criterion used in Sect. 5.1.3 to end up in the same cluster, initially). These represent the attack

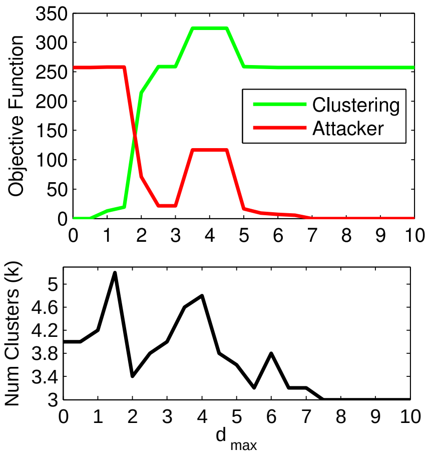

Figure 4: Results for the obfuscation attack averaged over five runs on the Digit dataset. The top plots shows the variation of the objective function for the attacker $d_c(\mathcal{C}^t, f(\mathcal{D} \cup \mathcal{A}'))$ and for the clustering algorithm $d_c(f(\mathcal{D} \cup \mathcal{A}), f(\mathcal{D} \cup \mathcal{A}'))$ as the maximum amount of modifications $d_{\max}$ to the initial attack samples $\mathcal{A}$ increases. The bottom plot reports the corresponding average number of selected clusters.

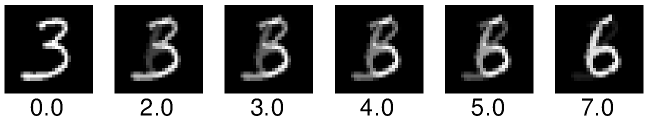

Figure 5: An example of how a digit '3' is gradually manipulated to resemble the closest '6', for different values of $d_{\max}$.

samples $\mathcal{A}$ that the attacker aims to obfuscate. We remind the reader that the attacker's goal in this case is to manipulate some samples to have them clustered according to a desired criterion, without affecting significantly the normal system operation. In particular, we assume here that the attacker can manipulate samples corresponding to the digit '3' in order to have them clustered together with the cluster corresponding to the digit '6', while preserving the initial clusters. In other words, the desired clustering output for the attacker consists of three clusters: one corresponding to the '0' digit, one corresponding to the '1' digit, and the latter corresponding to the digits '6' and '3'. These constraints can be easily encoded as a desired clustering output $\mathcal{C}^t$ through a binary matrix $\mathbf{Y}^t$. This reflects exactly Problem 5, where the attacker aims at minimizing $d_c(\mathcal{C}^t, f(\mathcal{D} \cup \mathcal{A}'))$.

On the other hand, as explained in Sect. 3.2, the clustering algorithm attempts to keep the attack points corresponding to the digit '3' into a well-separated cluster from the remaining digits, *i.e.*, it selects the number of clusters that minimizes $d_c(\mathcal{C}^\star, f(\mathcal{D} \cup \mathcal{A}'))$, which can thus be regarded as the objective function for the clustering algorithm. In this case, $\mathcal{C}^\star$ is the clustering obtained on the initial data and the non-manipulated attack samples, *i.e.*, $C^\star = f(\mathcal{D} \cup \mathcal{A})$.

The results for the above discussed obfuscation attack are given in Fig. 4, where we report the values of the objective function for the attacker and for the clustering algorithm, as well as the number of selected clusters, as a function of the maximum amount of allowed modifications to the attack samples, given in terms of the maximum Euclidean distance $d_{\max}$ (see Eq. 4). The results clearly show that the objective function of the attacker tends to decrease, while that of the clustering algorithm generally increases. The reason is that, initially, the clustering algorithm correctly separates the four clusters associated to the four distinct digits, whereas as $d_{\max}$ increases, the attack digits '3' are more and more altered to resemble the closest '6's, and are then gradually merged to their cluster. The number of clusters does not decrease immediately to 3 as one would expect since, while manipulating the attack samples, their cluster is fragmented into smaller ones (typically, two or three clusters). The reason is that, to remain as close as possible to the ideal $\mathcal{C}^\star$, the clustering algorithm avoids some of the '3's to immediately join the cluster of '6's by fragmenting the cluster of '3's.

When $d_{\max}$ takes on values approximately in $[3, 4]$, the clustering algorithm creates only three clusters, corresponding effectively to the attacker's goal $\mathcal{C}^t$ (this is witnessed by the fact that the averaged attacker's objective is almost zero). Surprisingly, though, as soon as $d_{\max}$ becomes greater than 4, the number of clusters raises again to 4, and some of the attack samples are again separated from the cluster of '6's, worsening the adversary's objective. This is due to the fact that, when $d_{\max} \approx 3$ or 4, some of the attack points work as *bridges* and successfully connect the remaining '3's to the cluster of '6's, whereas when these points are further shifted towards the cluster of '6's, the algorithm can successfully split the two clusters again. Based on this observation, a *smarter* attacker may even manipulate only a very small subset of her attack samples to create proper *bridges* and connect the remaining non-manipulated samples to the desired cluster. We however left a quantitatively investigation of this approach to future work.

In Fig. 5 we finally report an example of how a digit '3' is manipulated by our attack to be hidden in the cluster associated to the digit '6'. It is worth noting how, when $d_{\max} \in [2, 4]$, the original attack sample still significantly resembles the initial '3': this shows that the adversary's goal can be achieved without altering too much the initial attack samples, which is clearly a strong *desideratum* for the attacker in adversarial settings.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the problem of evaluating the security of clustering algorithms in adversarial settings, by providing a framework for simulating potential attack scenarios. We devised two attacks that can significantly compromise availability and integrity of the targeted system. We demonstrated with real-world experiments that single-linkage clustering may be significantly vulnerable to deliberate attacks, either when the adversary can only control a very small fraction of the input data, or when she slightly manipulates her attack samples. This shows that attacking clustering algorithms with tailored strategies can significantly alter their output to meet the adversary's goal.

Admittedly, one of the causes of the vulnerability of single-linkage resides in its inter-cluster distance, which solely depends on the closest points between clusters, and thus allowed for an efficient constructing of bridges. It is reasonable to assume that algorithms based on computing averages (*e.g.*, k-means) or density estimation might be more robust to poisoning, although not necessarily robust to obfuscation attacks. However, the results of our empirical evaluation can not be directly generalized to different algorithms, and more investigation should thus be carried out in this respect.

In general, finding the optimal attack strategy given an arbitrary clustering algorithm is a hard problem and we have to rely on heuristic algorithms in order to carry out our analysis. For the sake of efficiency, these heuristics should be heavily dependent on the targeted clustering algorithm, as in our case. However, it would be interesting to exploit more general approaches that ideally treat the clustering algorithm as a black box and find a solution by performing a stochastic search on the solution space (*e.g.* by simulated annealing), or an educated exhaustive search (*e.g.* by using branch-and-bound techniques).

In this work we did not address the problem of *countering attacks* by designing more *secure* clustering algorithms. We only assumed that the clustering algorithm can select a different number of clusters (optimal according to its goal) after each attack iteration. More generally, though, one can design a clustering algorithm that explicitly takes into account the adversary's presence, and her optimal attack strategy, *e.g.*, by modeling clustering in adversarial settings as a game between the clustering algorithm and the attacker. This has been done in the case of supervised learning, to improve the security of learning algorithms against evasion attempts [8], and similarly, in the regression setting [13]. Other approaches may more directly encode explicit assumptions on how the data distribution changes under attack, similarly to [5]. We left this investigation to future work.

Another possible future extension of our work would be to consider a more realistic setting in which the attacker has limited knowledge of the attacked system. To this end, the upper bound on the performance degradation incurred under attack provided by our worst-case analysis may be exploited to evaluate the effectiveness of attacks devised under limited knowledge (*i.e.*, how close they can get to the worst case).

One limitation of our approach may be the so-called *inverse feature-mapping* problem [17, 6], *i.e.*, the problem of finding a real attack sample corresponding to a desired feature vector (as the ones suggested by our attack strategies). In the reported experiments, this was not a significant problem since modifications to the given feature values could be directly mapped to manipulations on the *real* attack samples. Although inverting the feature mapping may be a cumbersome task for more complicated feature representations, this remains a common problem of optimal attacks in adversarial learning, and it has to be addressed in an application-specific manner, depending on the given feature space.

As a further future development, we plan to establish a link between the evaluation of the security of clustering algorithms and the problem of determining the *stability* of a clustering, which has been already addressed in the literature and used as a device for model selection (see, *e.g.*, [29]). Indeed, stable clusterings can be regarded as secure under specific non-targeted attacks like, *e.g.*, perturbation of points with Gaussian noise.

Understanding robustness of clustering algorithms against carefully targeted attacks under a more theoretical perspective (*e.g.*, by devising theoretical bounds that evaluate the impact of single attack points on the clustering output) may also be a promising research direction. Some results from clustering stability may be also exploited to this end.

7. ACKNOWLEDGMENTS

This work has been partly supported by the Regional Administration of Sardinia (RAS), Italy, within the projects "Security of pattern recognition systems in future internet" (CRP-18293), and "Advanced and secure sharing of multimedia data over social networks in the future Internet" (CRP-17555). Both projects are funded within the framework of the regional law *L.R. 7/2007, Bando 2009*. The opinions, findings and conclusions expressed in this paper are solely those of the authors and do not necessarily reflect the opinions of any sponsor.

8. REFERENCES

[1] Collaborative Malware Collection and Sensing. `https://alliance.mwcollect.org`.

[2] Project Malfease. `http://malfease.oarci.net`.

[3] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS '06: Proc. 2006 ACM Symposium on Information, Computer and Communications Security*, pages 16–25, NY, USA, 2006. ACM.

[4] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. Scalable, behavior-based malware clustering. In *NDSS*. The Internet Society, 2009.

[5] B. Biggio, G. Fumera, and F. Roli. Design of robust classifiers for adversarial environments. In *IEEE Int'l Conf. on Systems, Man, and Cybernetics (SMC)*, pages 977–982, 2011.

[6] B. Biggio, G. Fumera, and F. Roli. Security evaluation of pattern classifiers under attack. *IEEE Trans. on Knowledge and Data Eng.*, 99(PrePrints):1, 2013.

[7] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In J. Langford and J. Pineau, editors, *29th Int'l Conf. on Machine Learning*. Omnipress, 2012.

[8] M. Brückner, C. Kanzow, and T. Scheffer. Static prediction games for adversarial learning problems. *J. Mach. Learn. Res.*, 13:2617–2654, 2012.

[9] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proc. 1st ACM workshop on Security and Privacy in Smartphones and Mobile devices*, SPSM '11, pages 15–26, NY, USA, 2011. ACM.

[10] C. Castillo and B. D. Davison. Adversarial web search. *Foundations and Trends in Information Retrieval*, 4(5):377–486, May 2011.

[11] J. G. Dutrisac and D. Skillicorn. Hiding clusters in adversarial settings. In *IEEE Int'l Conf. on Intelligence and Security Informatics (ISI 2008)*, pages 185–187, 2008.

[12] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2011.

[13] M. Großhans, C. Sawade, M. Brückner, and T. Scheffer. Bayesian games for adversarial regression problems. In *J. Mach. Learn. Res. - Proc. 30th Int'l Conf. on Machine Learning (ICML)*, volume 28, 2013.

[14] P. Haider, L. Chiarandini, and U. Brefeld. Discriminative clustering for market segmentation. In *Proc. 18th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, KDD '12, pages 417–425, NY, USA, 2012. ACM.

[15] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, Dec. 2001.

[16] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song. Juxtapp: a scalable system for detecting code reuse among android applications. In *Proc. 9th Int'l Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'12, pages 62–81, Berlin, Heidelberg, 2013. Springer-Verlag.

[17] L. Huang, A. D. Joseph, B. Nelson, B. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *4th ACM Workshop on Artificial Intelligence and Security (AISec 2011)*, pages 43–57, Chicago, IL, USA, 2011.

[18] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., NJ, USA, 1988.

[19] M. Kloft and P. Laskov. Online anomaly detection under adversarial impact. In *Proc. 13th Int'l Conf. on Artificial Intell. and Statistics*, pages 405–412, 2010.

[20] A. Kolcz and C. H. Teo. Feature weighting for improved classifier robustness. In *Sixth Conf. on Email and Anti-Spam (CEAS)*, CA, USA, 2009.

[21] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In *Int'l Conf. on Artificial Neural Networks*, pages 53–60, 1995.

[22] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(1):167–172, 2007.

[23] R. Perdisci, D. Ariu, and G. Giacinto. Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks*, 57(2):487 – 500, 2013.

[24] R. Perdisci, I. Corona, and G. Giacinto. Early detection of malicious flux networks via large-scale passive DNS traffic analysis. *IEEE Trans. on Dependable and Secure Comp.*, 9(5):714–726, 2012.

[25] F. Pouget, M. Dacier, J. Zimmerman, A. Clark, and G. Mohay. Internet attack knowledge discovery via clusters and cliques of attack traces. *J. Information Assurance and Security, Vol. 1, Issue 1, March 2006.*

[26] G. Punj and D. W. Stewart. Cluster analysis in marketing research: Review and suggestions for application. *J. Marketing Res.*, 20(2):134, May 1983.

[27] D. B. Skillicorn. Adversarial knowledge discovery. *IEEE Intelligent Systems*, 24:54–61, 2009.

[28] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Professional, 2002.

[29] U. von Luxburg. Clustering stability: An overview. *Foundations and Trends in Machine Learning*, 2(3):235–274, 2010.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proc. 1996 ACM SIGMOD Int'l Conf. on Management of data*, SIGMOD '96, pages 103–114, NY, USA, 1996. ACM.