Poisoning Behavioral Malware Clustering

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

Konrad Rieck
University of Göttingen
Goldschmidtstraße 7
37077, Göttingen, Germany
konrad.rieck@uni-goettingen.de

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

Christian Wressnegger
University of Göttingen
Goldschmidtstraße 7
37077, Göttingen, Germany
christian.wressnegger@cs.uni-goettingen.de

Igino Corona
Università di Cagliari
Piazza d'Armi
09123, Cagliari, Italy
igino.corona@diee.unica.it

Giorgio Giacinto
Università di Cagliari
Piazza d'Armi
09123, Cagliari, Italy
giacinto@diee.unica.it

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

ABSTRACT

Clustering algorithms have become a popular tool in computer security to analyze the behavior of malware variants, identify novel malware families, and generate signatures for antivirus systems. However, the suitability of clustering algorithms for security-sensitive settings has been recently questioned by showing that they can be significantly compromised if an attacker can exercise some control over the input data. In this paper, we revisit this problem by focusing on behavioral malware clustering approaches, and investigate whether and to what extent an attacker may be able to subvert these approaches through a careful injection of samples with poisoning behavior. To this end, we present a case study on Malheur, an open-source tool for behavioral malware clustering. Our experiments not only demonstrate that this tool is vulnerable to poisoning attacks, but also that it can be significantly compromised even if the attacker can only inject a very small percentage of attacks into the input data. As a remedy, we discuss possible countermeasures and highlight the need for more secure clustering algorithms.

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 Machine Learning; Unsupervised Learning; Clustering; Security Evaluation; Computer Security; Malware Detection

1. INTRODUCTION

Automated techniques for behavioral clustering of malware have been found to be effective for the development of analysis, detection and mitigation strategies against a broad spectrum of malicious software. Such techniques can significantly ease the identification of polymorphic instances of well-known malware as well as novel attack types and infection strategies, reducing by orders of magnitude the burden of the analysis task [e.g., 20, 24, 27, 28].

Behavioral clustering is motivated by a key assumption: albeit malware writers can generate a large number of polymorphic variants of the same malware, e.g., using executable packing and other code obfuscation techniques [12, 15], these polymorphic variants will eventually perform similar activities when executed. To expose these behavioral similarities, malware binaries are usually executed in a monitored sandbox environment, in order to identify malware families characterized by similar host-level events [e.g., 1, 3, 20, 27] or network traffic patterns [e.g., 13, 14, 24, 25].

However, regardless the behavioral features being used, all these proposals suffer from the same vulnerability: *clustering algorithms have not been originally devised to deal with data from an adversary.* As outlined in recent work [4, 9], this may allow an attacker to devise carefully-crafted attacks

AISec'14, November 7, 2014, Scottsdale, Arizona, USA.
Copyright 2014 ACM 978-1-4503-2953-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2666652.2666666.

that can significantly compromise the clustering process itself, and invalidate subsequent analyses.

In this work, we also show that the effectiveness of clustering algorithms — in particular, single-linkage clustering — can be dramatically reduced by a skilled adversary through a proper, deliberate manipulation of malware samples, in the context of a more realistic application scenario that those considered in [4, 9]. To this end, we first review the attacker's model proposed in [4, 9], as it can also be exploited as a general threat model for behavioral malware clustering, and then investigate a worst-case attack against `Malheur` [28], an open-source malware clustering tool. We emulate an attacker who *adds* specially-crafted poisoning actions to the original behavior of malware samples, thus leaving intact their original malicious goals. Our experimental results clearly show that even a small fraction of 3% of poisoning samples may completely subvert the clustering process, leading to poor clustering results. Thus, our case study highlights the need for *robust* malware clustering techniques, capable of coping with malicious noise. As a consequence, a safe application of clustering algorithms for malware analysis remains an open research issue. Throughout the paper we sketch some promising ways of research towards this goal.

Contributions. In summary, the main contribution of this paper is to extend and adapt the poisoning attacks proposed in [4, 9] against the single-linkage hierarchical clustering algorithm to target `Malheur` [28], an open-source tool for behavioral malware clustering. In this case, the main difficulty with respect to previous work relies in constructing real malware samples that correspond to the desired, optimal feature vectors found by the optimal attack strategy, while accounting for application-specific constraints on the manipulation of the feature values of each sample. This is a well-known issue in the field of adversarial machine learning, referred to as the problem of *inverting* the feature mapping [7, 17]. To assess the effectiveness of poisoning attacks against behavioral malware clustering, we finally report an extensive set of experiments that highlight the vulnerability of such approaches to well-crafted attacks, as well as the need for identifying suitable countermeasures, for which we identify some interesting ways of research.

Organization. The remainder of this paper is structured as follows. In Sect. 2, we give an overview of recent work on behavioral malware clustering. The previously-proposed framework for the security evaluation of clustering algorithms [4, 9] is discussed in Sect. 3. In Sect. 4, we review the derivation of (worst-case) *poisoning* attacks, in which the attacker has perfect knowledge of the targeted system. In Sect. 5, we describe `Malheur`, the malware clustering tool exploited as a case study to evaluate our poisoning attacks. The latter are defined as variants of the previously-proposed poisoning attacks, to deal with the specific feature representation exploited by `Malheur`, in Sect. 6, where we also report the results of our experimental evaluation. Conclusions are discussed in Sect. 7, along with possible future research directions.

2. MALWARE CLUSTERING

The urgent need for automated analysis of malware naturally comes with the ever-growing number of malicious codes on the Internet. In recent years, machine learning techniques have received attention in this area, as they enable improving the automation of malware analysis. One prominent representative are clustering algorithms. These algorithms enable grouping similar malware automatically and can thereby reduce the manual efforts required for developing mitigation and detection techniques. Several approaches for such a clustering have been devised in the last years, most notably, (a) *clustering of network traffic*, and (b) *clustering of program behavior*.

Clustering of network traffic. Network communication is a key component of malware and thus several malware families can be solely characterized by their network traffic. For example, Gu et al. correlate spatial-temporal relations in botnet communication using clustering [14]. To this end, the authors make use of hierarchical clustering on the basis of q-grams over a so-called "activity log" which describes a botnet's network communication in terms of different types of responses. This approach is then extended to a more general concept of C&C communication [13], where the authors attempt to be agnostic to the protocol used as well as the concrete hierarchy of the botnet.

In a similar line of research, Perdisci et al. [25] focus on HTTP-based malware with the objective to automatically generate network signatures for malware. In particular, they use single-linkage clustering over three stages, mainly to reduce computational complexity: first, a "coarse-grained" clustering is performed; each of the corresponding clusters is then subdivided into a more "fine-grained" set of clusters; and, eventually, similar clusters are merged together to avoid redundant signature generation. An extension by the same authors [24] focuses more on the scalability of the proposed approach, in terms of the number of samples that the system is able to process in a given amount of time (i.e., the so-called *throughput*). The authors utilize an approximate clustering algorithm for the first stage of their approach. This not only speeds up the initial stage but also decreases the need of a merging phase, thus yielding a significant increase of the overall throughput of the system.

Clustering of program behavior. A second strain of research has considered program behavior of malware for identifying related samples. Despite polymorphism and obfuscation, variants of the same malware family often show similar program behavior. Bailey et al. [1] have been the first to apply clustering algorithms to this information. In particular, they obtain a single-linkage clustering by computing pairwise distances between sequences of host-level events. This approach, however, has a quadratic runtime complexity and therefore quickly reaches its limits in terms of the possible throughput.

Bayer et al. [3] counter this shortcoming with an approximate clustering using locality sensitive hashing (LSH). This makes it possible to scale the analysis to several thousand malware samples. The behavioral analysis is powered by the malware analysis system Anubis [18]. Closely related to this approach is the tool `Malheur` [28], which we use in our case study to demonstrate the effectiveness of our attacks. `Malheur` makes use of program behavior monitored by CWSandbox in MIST Format [31, 35] and is described in more detail in Sect. 5.

More recently, several extensions have been proposed for improving behavioral clustering of malware in practice. For example, Jang et al. [20] apply feature hashing for clustering large sets of malware binaries, Perdisci & U [26] propose an automatic procedure for calibrating clustering algorithms, and Hu & Shi [16] combine behavioral clustering with static code analysis.

Although each of the presented approaches provides advantages for keeping abreast of malware development, all approaches employ standard clustering algorithms which have not been originally designed to explicitly cope with malicious noise. Consequently, the attacks proposed in this paper can be potentially adapted to several of these approaches with minor modifications.

3. SECURITY EVALUATION OF CLUSTERING ALGORITHMS

In this section we briefly review the framework proposed by Biggio et al. [4, 9] for the security evaluation of *unsupervised* learning algorithms (including clustering) against adversarial attacks. Similarly to previous work on the security evaluation of *supervised* learning algorithms [2, 7, 17], this framework relies on a threat model that consists of defining the adversary's goal, knowledge of the attacked system, and capability of manipulating the input data, in order to formalize an *optimal* attack strategy.

In the sequel, we describe this framework using the same notation defined in Biggio et al. [4, 9]. We refer to any clustering algorithm as a function f that maps a given dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{\mathsf{n}}$ to a clustering result $\mathcal{C} = f(\mathcal{D})$, without specifying the structure of $\mathcal{C}$ at this stage, as it depends on the given clustering algorithm.

3.1 Adversary's Goal

The adversary's goal can be defined in terms of the desired security violation, and of the so-called attack specificity [2, 4, 7, 9, 17]. A security violation may compromise the system *integrity*, its *availability*, or the *privacy* of its users. *Integrity violations*, in general, aim to perform some malicious activity without compromising the normal system operation. In the unsupervised learning setting, they have thus been defined as attacks aimed at changing the clustering of a given set of samples, without significantly altering the clustering result on the rest of the data. *Availability violations* aim to compromise system operation, causing a Denial of Service (DoS). Therefore, in the unsupervised setting, availability attacks have been defined as attacks that aim to subvert the clustering process by altering its result as much as possible. By contrast, *privacy violations* are defined as attacks that may allow the attacker to gather information about the system's users by reverse-engineering the clustering process. The attack specificity can be *targeted* or *indiscriminate*, depending on whether the attack aims to modify the clustering output only on a specific subset of samples, or indiscriminately on any sample.

3.2 Adversary's Knowledge

In order to achieve her goal, the adversary may exploit information at different abstraction levels about the targeted system. We summarize them in the following. First, the attacker may know the whole dataset $\mathcal{D}$, a subset of it, or more realistically, only a *surrogate* dataset $\mathcal{S}$, that might be obtained from the same source of $\mathcal{D}$, e.g., publicly available malware blacklists. Second, the adversary might be aware of, and reproduce, the extraction process of the whole feature set, or a portion of it. Indeed, when it comes to attacking open-source tools such as `Malheur`, the adversary clearly has full knowledge of the feature set. Finally, the adversary might be aware of the targeted clustering algorithm, as well as of its initialization parameters (if any). In the case of `Malheur`, this translates into knowing the user-specified configuration of the tool.

Perfect knowledge. The worst-case scenario in which the attacker has full knowledge of the targeted system is usually referred to as *perfect knowledge* [2, 5–8, 10, 17, 22]. In our case, this amounts to knowing the data, the feature space, the clustering algorithm and its initialization (if any).

3.3 Adversary's Capability

The adversary's capability specifies how and to what extent the adversary can manipulate the input data to alter the clustering process. In several cases it is realistic to consider that the attacker can add a maximum number of (potentially manipulated) samples to the dataset $\mathcal{D}$, without affecting the rest of the data. For instance, anyone, including a skilled adversary, can submit novel malware samples to publicly-available malware-analysis services such as VirusTotal [33] and Anubis [18], which can in turn be used as sources to collect malware by registered users. If malware is collected from them, and clustered afterwards, the adversary may actually control a (small) percentage of the input data given to the clustering algorithm.

An additional constraint may be given in terms of how malware samples can be manipulated. In fact, to preserve its malicious functionality, malware code may not be manipulated in an unconstrained manner. Such a constraint can be often encoded by a suitable distance measure between the original, non-manipulated attack samples and the manipulated ones, as in [2, 7, 17, 23]. However, this strictly depends on the specific application and feature representation.

3.4 Attack Strategy

Based on the presented threat model, consisting of assumptions on the adversary's goal, knowledge and capabilities, we can finally define the *optimal* strategy for attacking a clustering algorithm as:

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

In this formulation, as in [4, 9], the adversary's knowledge is characterized by a parameter vector θ, whose elements embed information about the input data $\mathcal{D}$, the clustering algorithm f, and its parameters (as discussed in Sect. 3.2). The uncertainty of the adversary about the elements of θ is captured by a probability distribution μ defined over the set of all possible configurations θ. Moreover, the objective function $g(\mathcal{A}'; \theta) \in \mathbb{R}$ measures the extent to which the adversary's goal is fulfilled by the set of attack samples $\mathcal{A}'$ used to taint the initial data $\mathcal{D}$, given the knowledge θ. In the above formulation, we consider the maximization of the expected value of this function with respect to θ sampled from the distribution μ, denoted as $\mathbb{E}_{\theta \sim \mu}[\cdot]$. Finally, the adversary's capability is encoded by the set $\Omega(\mathcal{A})$, which denotes the possible manipulations that the attacker can make on a given a set of attack samples $\mathcal{A}$ before adding them to the original set $\mathcal{D}$. The set $\mathcal{A}$ of initial attacks can be empty, e.g., if the attack samples can be generated from scratch without preserving or exhibiting any malicious functionality.

It is finally worth remarking that the above optimization problem is formulated in terms of the considered feature representation, as many other adversarial machine learning problems [5–7, 9, 17]. In practice, after solving this problem, we are given a set of *optimal feature vectors* for which we have to subsequently build a set of corresponding *real samples* to practically execute the attack. This is clearly an application-specific problem that may not be trivial to solve depending on the given feature representation. However, it can be mitigated by incorporating specific constraints on the manipulation of the feature values of the attack samples, while defining the set Ω, as we will see in the next sections.

4. POISONING ATTACKS WITH PERFECT KNOWLEDGE

Following the framework described in the previous section, *poisoning attacks* are defined as indiscriminate availability violations (i.e., DoS attacks) in which the attacker aims to maximally alter the clustering result on any of the input samples through the injection of well-crafted *poisoning* samples. In the case of malware clustering, this amounts to adding carefully-designed malware samples to the input data to avoid the correct clustering of malware exhibiting similar behavior and, thus, the correct identification of both known and novel malware families.

As in previous work [4, 9], we are interested in analyzing the worst possible performance degradation that the system may incur under this attack. We therefore assume that the attacker has perfect knowledge of the targeted system, as described in Section 3.2. Accordingly, the expectation in Eq. (1) vanishes and the objective simply becomes $g(\mathcal{A}'; \theta_0)$, being θ_0 the set of parameters representing perfect knowledge of the system. Further, for this kind of attack, the objective function $g(\mathcal{A}'; \theta_0)$ can be defined as a distance function between the clustering result $\mathcal{C}$ obtained from the untainted data $\mathcal{D}$ and the clustering result $\mathcal{C}' = f_{\mathcal{D}}(\mathcal{D}')$ restricted to the same data (through a projection operator $f_{\mathcal{D}}$), but obtained from the tainted data $\mathcal{D}' = \mathcal{D} \cup \mathcal{A}'$ (i.e., including the set $\mathcal{A}'$ of attack samples). The objective can be thus written as $g(\mathcal{A}'; \theta_0) = d_{\mathrm{c}}(\mathcal{C}, f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}'))$, where d_{c} is a suitable distance function between clusterings. Note that poisoning samples are excluded from the computation of the objective function since the attacker's goal is to maximally subvert the clustering output on the *untainted* input data, and not on the poisoning samples (which may otherwise bias the evaluation of the attack's impact).

If the clustering algorithm f assigns each sample to a cluster, the clustering result $\mathcal{C}$ can be represented as a matrix $\mathtt{Y} \in \{0,1\}^{\mathtt{n} \times \mathtt{k}}$ (k being the number of clusters found), where each $(i,j)^{\mathrm{th}}$ component equals 1 if the i^{th} sample is assigned to the j^{th} cluster, and 0 otherwise. Within this setting, a possible distance function between clusterings amounts to counting how many pairs of samples have been clustered together in one clustering and not in the other, or viceversa:

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

where $\| \cdot \|_F$ is the Frobenius norm, and each element of the matrix $\mathtt{Y}\mathtt{Y}^{\top} \in \{0,1\}^{\mathtt{n} \times \mathtt{n}}$ (and, similarly, of $\mathtt{Y}'\mathtt{Y}'^{\top}$) represents whether the corresponding pair of samples has been clustered together (1) or not (0).

As mentioned earlier, to poison the clustering process the adversary can add a set $\mathcal{A}'$ of attack samples to the input data $\mathcal{D}$. We bound the adversary's capability here by limiting the maximum number of injected poisoning samples to $\mathtt{m}$, i.e. $|\mathcal{A}'| \leq \mathtt{m}$. Additional constraints on the set of attack samples can be identified depending on the given feature representation, to facilitate the fabrication of real samples exhibiting the desired feature values. In general, we denote the set of constraints to be fulfilled by the poisoning attack as $\mathcal{A}' \in \Omega_p$. To give a concrete example, consider that `Malheur` can be configured to extract binary feature vectors that are subsequently normalized to have unitary ℓ_2-norm. In this case, the set of constrained attack samples can be expressed as:

$$\Omega_p = \left\{ \{\boldsymbol{a}'_i\}_{i=1}^{\mathtt{m}} \, : \, \boldsymbol{a}'_i \in \{0, 1/||\boldsymbol{a}'_i||_2\}^{\mathtt{d}} \text{ for } i = 1, \cdots, \mathtt{m} \right\} \, , \tag{3}$$

where $\mathtt{d}$ is the number of features, and $|| \cdot ||_2$ denotes the ℓ_2-norm of a vector.

In general, the optimal attack strategy for poisoning attacks with perfect knowledge can be therefore derived from Eq. (1) and written independently from the specific clustering algorithm as:

$$\begin{array}{rl} \text{maximize} & d_{\mathrm{c}}(\mathcal{C}, f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}')) \\ \text{s.t.} & \mathcal{A}' \in \Omega_{\mathrm{p}} \, . \end{array} \tag{4}$$

Unfortunately, this problem can not be solved analytically only if the clustering output is analytically predictable, which is not usually the case. We have thus to resort to suitable heuristics depending on the considered clustering algorithm to devise effective attacks. In the next section we investigate heuristics to solve the above problem [see 4, 9] and poison the single-linkage hierarchical clustering algorithm, as we will exploit them in our case study against `Malheur`.

4.1 Poisoning single-linkage hierarchical clustering

Before describing the heuristics for poisoning the single-linkage clustering algorithm, it is worth pointing out that this algorithm, as any other variant of hierarchical clustering, outputs a *hierarchy* of clusterings [19]. Such a hierarchy is constructed by initially considering each data point as a single cluster, and iteratively merging the closest clusters together, until a single cluster containing all data points is obtained. Clusters are merged according to a given distance measure, also referred to as *linkage* criterion. In the *single-linkage* variant, the distance between any two clusters ($\mathcal{C}_i$, $\mathcal{C}_j$) is defined as the minimum Euclidean distance between all possible pairs of samples in $\mathcal{C}_i \times \mathcal{C}_j$.

To obtain a given data partitioning into clusters, a suitable cutoff distance has to be chosen. This determines the maximum intra-cluster distance for each cluster, and, thus, indirectly, the total number of clusters. We follow the approach of Biggio et al. [4, 9] and select the cutoff distance that achieves the minimum distance between the clustering obtained in the absence of attack $\mathcal{C}$ and the one obtained in the presence of poisoning, i.e., $\min d_c(\mathcal{C}, f_{\mathcal{D}}(\mathcal{D} \cup \mathcal{A}'))$. The reason is that this is the worst-case cutoff criterion for the attack, which is thus expected to work potentially even better under less pessimistic choices of the cutoff distance.

Given a suitable criterion for selecting the cutoff distance, it is possible to model the clustering output as a binary matrix $\mathtt{Y} \in \{0,1\}^{\mathtt{n} \times \mathtt{k}}$ indicating the sample-to-cluster assignments, and thus use the distance measure d_c defined in Eq. (2) as the objective function in Problem (4). This problem has then been solved by means of specialized search heuristics specifically tailored to the considered clustering algorithm. In particular, we have considered greedy optimization approaches in which the attacker aims to maximize the objective function by adding one attack sample at a time, i.e., $|\mathcal{A}'| = \mathsf{m} = 1$. We have found that the objective function is often maximized when the attack point is added in between clusters that are sufficiently *close* to each other. The reason is that such an attack tends to decrease the distance between the two clusters, thus causing the algorithm to potentially merge them into a single cluster.

Bridge-based attacks. Based on this observation, we have thus devised a family of attacks that aim to iteratively *bridge* the closest clusters. Let us assume that at each iteration we are given a set of k clusters, and we have to select the best attack point to be added to the current dataset. Each bridge-based attack generates the same set of $k-1$ candidate attack points, by considering the $k-1$ links between pairs of points that have been cut to separate the current clustering from the top of the hierarchy, i.e., the $k-1$ shortest connections between clusters. Each candidate attack point is then computed as the midpoint between the points in each of the $k-1$ identified pairs, as conceptually represented in Fig. 1. The difference among the bridge-based attacks relies only on how the best attack point is selected at each iteration.

Bridge (Best). This strategy adds each candidate attack point to the current dataset, one at a time, re-runs the clustering algorithm on such data, and chooses the attack point that maximally increases the objective function. This is clearly a computationally-intensive procedure, especially for large datasets.

Bridge (Hard). This strategy aims to improve efficiency by avoiding us to re-run the clustering k times at each attack iteration. The underlying idea is to approximate the clustering result $\mathtt{Y}'$ on the current dataset including the considered candidate attack point, without re-computing the clustering explicitly. To this end, the attack point is assumed to effectively merge the two adjacent clusters. For each point belonging to one of the two adjacent clusters, we thus set to 1 (0) the value of $\mathtt{Y}'$ corresponding to the first (second) cluster. This amounts to considering *hard* clustering assignments. 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.

Bridge (Soft). This is a variant of the latter approach that estimates $\mathtt{Y}'$ using soft clustering assignments instead of hard ones. In particular, the $(i,k)^{\text{th}}$ element of $\mathtt{Y}'$ is estimated as the posterior probability that the i^{th} sample belongs to the k^{th} cluster, using a Gaussian Kernel Density Estimator (KDE) with bandwidth parameter h. When h is too small, the posterior estimates tend to the value of $1/k$, i.e., each point is assigned to any cluster with the same probability. When h is too high, instead, they tend to hard assignments. As a rule of thumb, the value of h should be thus comparable to the average distance between all possible pairs of samples in the dataset. The rationale of this strategy is to try finding connections that can potentially merge large clusters with more than one attack sample, to mitigate the limitation of our greedy approach.

5. A CASE STUDY: MALHEUR

To illustrate the effect of the proposed poisoning attacks in a practical setting, we conduct a case study with the open-source tool `Malheur`.[1] The tool implements techniques for clustering and classification of program behavior and has been applied in different settings for analyzing malware in the wild [11, 16, 28]. The analysis realized by `Malheur` builds on four basic steps.

1. *MIST Representation.* As the first step, the behavior of malware binary is monitored in a sandbox environment and stored as *MIST reports* [31]. In this format, the behavior of a program is described as a sequence of events, where individual execution flows of threads and processes are grouped in a single, sequential report. Each event encodes one monitored system call and its arguments, where the arguments are arranged in different levels of blocks, reflecting behavior with different degree of granularity. Depending on the configuration of `Malheur`, the monitored behavior can be analyzed at these different *MIST levels* [28].

2. *Embedding.* As the next step, `Malheur` embeds the monitored behavior in a high-dimensional vector space, where each dimension is associated with a short sequence of q events—a so called *q-gram.* If a q-gram occurs in the monitored events of a program, the respective dimension is set to 1 in its vector, otherwise it is set to 0. To enable a fair comparison of programs that strongly differ in the amount of observed events, each vector $\boldsymbol{x}$ is additionally normalized, such that $||\boldsymbol{x}||_2 = 1$, namely, projecting the vectors onto a hypersphere of unit radius in the vector space.

3. *Clustering.* For partitioning the embedded behavior into groups, `Malheur` implements an efficient variant of hierarchical clustering that supports single-linkage and complete-linkage hierarchical clustering. To alleviate the quadratic run-time complexity of these clustering algorithms, the tool can approximate the underlying data by limiting the analysis to a small subset of prototypes. For our case study, we disable this functionality and instead apply `Malheur` without prototype-based approximation.

4. *Classification.* Finally, `Malheur` supports assigning unknown behavior to previously discovered clusters. This assignment is realized using a nearest-neighbor classification, where a new vector is assigned to a nearby cluster if it appears within a certain distance to its members. This nearest-neighbor classification can be approximated by searching for nearest neighbors in a set of prototypes instead of all cluster members. We again disable this functionality and operate on the full data for our case study.

Each of the four steps supports different parameters that can be adapted in the configuration of `Malheur`. For our case study, we start with a basic setup by using MIST level 1, setting the q-gram length to 1 and especially considering single-linkage clustering by disabling the prototype-based approximation used by `Malheur`. The use of the latter would indeed imply a sort of complete-linkage pre-processing clustering step, which would in turn require us to significantly revisit the derivation of a proper poisoning attack. We therefore leave this issue to future work. Finally, although this

[1] `http://www.mlsec.org/malheur`

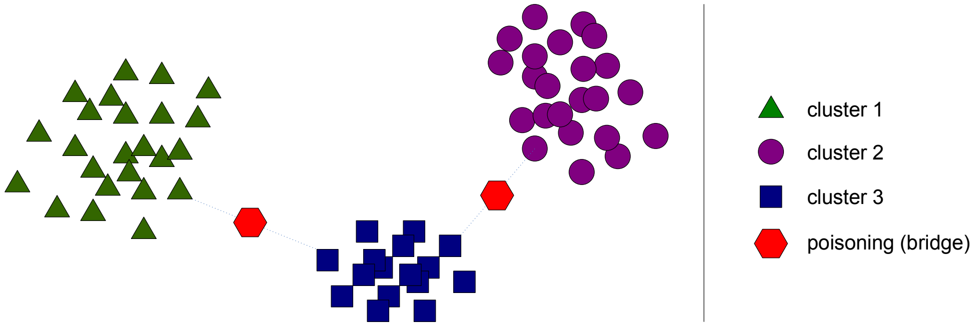

Figure 1: Bridge-based attacks against single-linkage clustering. The candidate attack samples connecting the $k-1$ closest clusters are highlighted as red hexagons.

setup slightly simplifies our attack, previous work has already shown that creating artificial q-grams of system calls, and similarly the use of different MIST levels, is not a challenge for an attacker [29, 30, 34], especially if she has full control over the behavior, as in the case of malware.

6. EXPERIMENTAL EVALUATION

In this section we apply the aforementioned evaluation framework to a concrete case study: we evaluate the worst-case effects of the poisoning attacks described in Section 4.1 using *real* malware samples, and against a real-world tool for behavioral malware clustering. In Section 6.1 we present the datasets employed for our investigation. Then, in Section 6.2 we provide all relevant details about the experimental setup and evaluation metrics. In Section 6.3 we summarize the main attack strategies implemented for the evaluation, including the modifications to the derivation of poisoning attacks that allow us to deal with the specific feature representation exploited by `Malheur`. Finally, in Section 6.4 we present and discuss our experimental results.

6.1 Datasets

For our experiments and evaluation we make use of two different datasets: first, the data that was originally considered by Rieck et al. in [28], and second, a dataset consisting of more recent malware samples collected in 2013.

Malheur data. This dataset consists of a selection of 3131 malware samples collected in a period of 3 years up to August 2009, and made publicly available in the same year.[2] It comprises a *reference dataset* that was used to calibrate the clustering algorithm in [28], and 7 *application datasets* for evaluating and testing their approach. The latter represent malware found on the Internet within 24 hours on 7 consecutive days. For our experiments we stick to a similar setup in order to ensure the comparability with the original approach and optimally show the practicality of our attack.

Recent Malware data. In addition to the data used for the `Malheur` project, we gathered malware samples from most prominent families in 2013. Similarly to [28] we rely on the popular antivirus scanner by Kaspersky Lab for this ranking and labeling of the malware samples. We chose 5 of the top 10 detections according to a recent threat report [21], and selected those families for which we were able to gather more than 100 but at most 150 samples. A summary of the exact numbers is given in Table 1.

Dataset	Number of samples
DangerousObject.Multi.Generic	129
Trojan.Win32.Generic	120
Virus.Win32.Sality.gen	112
Trojan.Win32.Starter.lgb	150
Virus.Win32.Nimnul.a	146
Total	657

Table 1: Summary of the malware families collected in 2013 for the *Recent Malware* data.

When running `Malheur` on the aforementioned datasets using the MIST-level-1 binary embedding discussed in Sect. 5, we have respectively found 85 and 78 distinct feature values (i.e., 1-grams).

6.2 Experimental Setup

To fairly evaluate the clustering process, we randomly split each dataset into two disjunct portions of equal size, namely, T and S. The T portion is used to calibrate the clustering algorithm, and, in particular, to estimate the cutoff distance (see Sect. 4.1). As suggested in [28], we select as the optimal cutoff distance the one that maximizes the F-measure (see below for its definition). The S split is then used to evaluate the calibrated clustering on unseen malware against an increasing percentage of poisoning attacks. This procedure is repeated five times and results are averaged over these repetitions. In our experiments, the value of the cutoff distance has been found to be 0.49 on the *Malheur* dataset and 0.63 on the *Recent Malware* dataset, on average, with a negligible standard deviation in both cases. Although in these experiments we assume that the cutoff distance is known to the attacker, more realistically an attacker can try estimating it from the data in a more conservative manner (i.e., essentially underestimating its real value), eventually poisoning the clustering result at the expense of using more poisoning samples. Clustering results are evaluated according to the three measures given below.

[2] http://pi1.informatik.uni-mannheim.de/malheur/

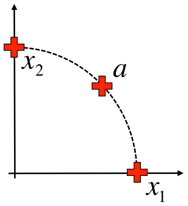

Figure 2: Computation of a bridge-based attack against single-linkage clustering, using the feature representation of `Malheur`, i.e., a binary embedding with points additionally projected onto a unit hypersphere (i.e., with unit ℓ_2 norm). This example considers a simple two-dimensional feature set, where we highlighted the only three admissible points, i.e., the samples $\boldsymbol{x}_1 = (0, 1)$ and $\boldsymbol{x}_2 = (1, 0)$, and their ideal bridging point $\boldsymbol{a} = (1/\sqrt{2}, 1/\sqrt{2})$. Besides this simple case, the creation of effective bridge-based attacks in this space is generally much more challenging than that considered in our previous work [4, 9], due to the restrictions imposed by the given feature representation.

1. The objective function in Eq. (4), with d_c given by Eq. (2), that measures the distance of the current clustering from that obtained in absence of poisoning.

2. The number of clusters, that helps us to gain a better understanding of how the attacks taint the clustering process. In particular, since the considered poisoning attacks are expected to "bridge" clusters, the number of clusters should decrease as the attack progresses.

3. The F-measure [28, 32], defined as the harmonic mean between precision (π) and recall (ρ), i.e., $2\frac{\pi\rho}{\pi+\rho}$. The latter are respectively computed as $\pi = \frac{1}{n}\sum_j \max_i \mathtt{c}_{ij}$, and $\rho = \frac{1}{n}\sum_i \max_j \mathtt{c}_{ij}$, where $\mathtt{c}_{ij}$ is the number of malware samples belonging to the i-th family present in the j-th cluster [28]. Precision reflects how well individual clusters agree with malware families, whereas recall measures to which extent malware families are scattered across clusters. Both measures provide complementary information about the quality of clustering results, summarized by the F-measure.

6.3 Attack Strategies

In this section we explain how we generate the poisoning samples to attack `Malheur`. First, as `Malheur` is configured with binary embedding and ℓ_2 normalization in our case (see Sect. 5), the feature vectors of poisoning samples have to fulfill the constraints given by the set Ω_p in Eq. (3). Besides these constraints, another fundamental pre-requisite that we impose is that poisoning points have to represent realistic malware samples. The reason is that a sample that does not exhibit any malicious or intrusive functionality may not be included into the set of collected malware to cluster, and the attack would be trivially defeated.[3] Therefore, we generate every poisoning sample by first selecting a malware sample from the given S split, and then manipulating its features by only increasing their value. Note that the value of a feature can only be increased from 0 to $1/||\boldsymbol{a}||_2$, being $||\boldsymbol{a}||_2$ the ℓ_2-norm of the attack sample. We thus refer in the following to this kind of manipulation as feature addition, for short. This manipulation indeed preserves the malicious functionality of the initially-selected malware, as it does not compromise the set of instructions required to execute the original malicious code. Moreover, before adding any candidate poisoning point to the data, we verify whether another point with the same feature values is already present. If this is the case, we discard the current sample and choose the next best candidate attack point. This allows us to discard duplicate attack points, as their presence may worsen the attack progress.

An important consequence of the particular embedding used by `Malheur` is that it affects the way we compute the *bridge* between any two points to create our candidate attack samples. This is a rather important distinction with respect to our previous work in [4, 9]. In fact, the midpoint in this case can not be computed as the average of the two neighboring points, as it is instead possible, for instance, when real-valued features are used. However, the point that is as equidistant as possible from each of the two neighboring points can be found by cloning the neighboring point with the smaller norm first, and then starting adding features to it that are not null in the other neighboring point, until the candidate attack point is as equidistant as possible from the two points. A simple two-dimensional example is given in Fig. 2. The only drawback of this procedure is that the candidate attack point may be sometimes farther from the two neighboring points than they are with respect to each other. In these cases, the attack may not be effective, as it may not effectively bridge the two neighboring clusters.

In these experiments we consider six distinct poisoning attack strategies. In addition to the three bridge-based attacks defined in Sect. 4.1, we consider *Random* and *Random (Best)* as in [9], and a variant of our bridge-based attacks named *F-measure (Best)*. *Random* generates any attack point by cloning a randomly-selected malware from the available set S, and adding to it a random number of features. *Random (Best)* works similarly, with the difference that not one but $k - 1$ attack points are selected at random, being k the actual number of clusters at any given attack iteration. Then, the objective function is evaluated for each candidate point by re-running the clustering algorithm, and the best attack point is chosen. *F-measure (Best)* works as *Bridge (Best)*, but chooses the best candidate attack point as the one that minimizes the F-measure instead of maximizing the objective function $d_c(\mathtt{Y}, \mathtt{Y}')$. As *Random (Best)* and *Bridge (Best)*, this strategy also requires evaluating the clustering result $k-1$ times to determine the best attack at each iteration, while the other strategies are computationally lighter. As for *Bridge (Soft)*, we set the kernel bandwidth h as the average distance between each possible pair of samples in the data, which yielded $h \approx 0.2$ in each run. We finally point out that, if more than one candidate attack point exhibit the same value of the desired function (either the objective function or the F-measure, depending on the attack strat-

[3] Note however that the main goal of poisoning samples is not to preserve the malicious functionality of the embedded malware code, which is required here only to avoid having such samples discarded by a simple preliminary antivirus analysis. Instead, their primary goal is to subvert the clustering output on the rest of the data, in order to produce a less effective characterization of malware families. This may indeed not only lead to lower malware detection rates or higher false alarm rates, but it also makes more difficult to identify the proper countermeasures or removal tools in case of infection.

egy), we select the one that produces the smaller number of clusters. If the tie persists, we break it at random.

6.4 Results

Results for the *Malheur* and the *Recent Malware* datasets are presented in Fig. 3. For each dataset, we show how the value of the objective function, the number of clusters, and the F-measure change for an increasing percentage of injected poisoning samples. We observe a similar behavior of these metrics for both datasets, which is summarized below in two points.

1. Simply injecting random points does not allow one to significantly worsen the quality of the resulting clustering. We can in fact observe that, for the *Random* attack, neither the value of the objective function nor the F-measure are affected at all. The reason is that each of the randomly-generated attack points is too far from the other clusters and it is thus clustered as a singleton, without affecting the clustering result on the rest of the samples. *Random (Best)* performs slightly better, as it clearly makes $k-1$ attempts at each iteration to find a better attack point, instead of one.

2. Maximizing the considered objective function actually allows us to reduce the number of clusters, and, thus, to compromise the quality of the resulting clustering, despite it does not incorporate any knowledge of the problem domain, of the clustering algorithm, and of the features used. Furthermore, looking at Fig. 3, we can observe that the bridge-based strategies that maximize the objective function achieve similar performances to *F-measure (Best)*, which instead minimizes the F-measure. Whereas the objective function is general, the F-measure takes into account the ground truth of the problem. We can therefore reasonably argue that the proposed objective function and the consequent attack strategies can be successfully employed to attack also systems different from `Malheur`.

Some further comments can be made separately for the two data sets. What appears evident from the results on the *Malheur* dataset is that injecting an even small percentage of poisoning points reduces significantly the number of clusters. *Bridge (Best)* and *F-measure (Best)* are able to reduce the number of cluster from an initial value of 40 to a value of 5 with only the 2% of injected samples. If we further increase such percentage up to 5% a single, large cluster is created, where all the initial ones are merged. *Bridge (Soft)* and *Bridge (Hard)* appear to be a bit less effective since they require a slightly higher percentage of injected samples to achieve similar results. Nevertheless, it is worth pointing out that, from a computational standpoint, both these strategies are significantly less expensive than the *Best* strategies.

On the *Recent Malware* dataset the considered attacks appear to be less effective. In particular, the bridge-based attacks here are not able to merge all the clusters into a unique cluster. At some point, instead, it happens that the strategies are no longer able to inflict any damage to the current clustering. The reason is that the candidate bridge points in this case are selected too far from their corresponding neighboring points, and the former are thus clustered apart instead of successfully merging the desired clusters. We argue that this may be somehow due to the smaller number of features found in this dataset, as this factor limits the number of manipulations that the attacker can make to find a suitable attack point. This may be an interesting starting point for future work to understand how to improve robustness of clustering algorithms to poisoning attacks by restricting the feature set and the number of potential manipulation the attacker can make on the attack samples. Nevertheless, one should keep in mind that, in this case, the objective function reaches anyway the value of 250 for *Bridge (Best)*, which still means that 250 pair of samples out of 329 samples have changed their clustering assignment with respect to the clustering in the absence of poisoning.

7. CONCLUSIONS AND FUTURE WORK

A widespread approach for coping with the plethora of novel malware are clustering algorithms from the area of machine learning. While these algorithms can help grouping similar malware samples automatically, they have not been originally designed to operate in an adversarial setting. Our work shows that, by leveraging on vulnerabilities of clustering algorithms, an attacker can significantly impact the performance of malware clustering. In our evaluation, only a small fraction of poisoning samples is necessary to largely destroy the recovery of families in a dataset of real malware. In particular, in this work we have considered `Malheur`, i.e., a popular malware clustering tool. We have modified previously-proposed poisoning attacks to cope with its specific feature representation, and to incorporate the corresponding application-specific constraints in the creation of real, poisoning malware samples. Although we have focused on a particular setup of this tool, we argue that attacks to other setups and clustering systems should not be considered a major challenge for a sophisticated attacker. Creating behavioral features artificially may be more or less difficult depending on the underlying sandbox environment, yet the exploited vulnerability resides in the clustering algorithms and thus can hardly be fixed by changing the feature representation. As a result, our work casts serious doubt about the security of *some* clustering algorithms in malware analysis systems, and there may be considerable need for novel algorithms that are more robust against poisoning and malicious noise.

Future extensions of this work may include: investigation of attacks in which the adversary has only limited knowledge of the system, i.e., attacks in which the input data is not known to the attacker, who may realistically only collect surrogate data from the same sources; development of poisoning attacks that may target a larger family of clustering algorithms (instead of considering only specialized heuristics); and development of appropriate countermeasures to improve security of clustering algorithms against adversarial threats and well-crafted attacks.

It is also worth remarking here that poisoning attacks are not the only kind of attack that may be incurred by a clustering-based system operating in an adversarial setting; e.g., if some of the clusters are used to characterize the behavior of legitimate users or software, an attacker may aim to manipulate the malware behavior in order to mimic the legitimate samples, without significantly altering the clustering output on the rest of the data. This attack has been referred to as *obfuscation* attack in [9]. We refer the reader to the same work for a detailed taxonomy of potential attacks against clustering. However, the implementation of such attacks for more realistic application scenarios and

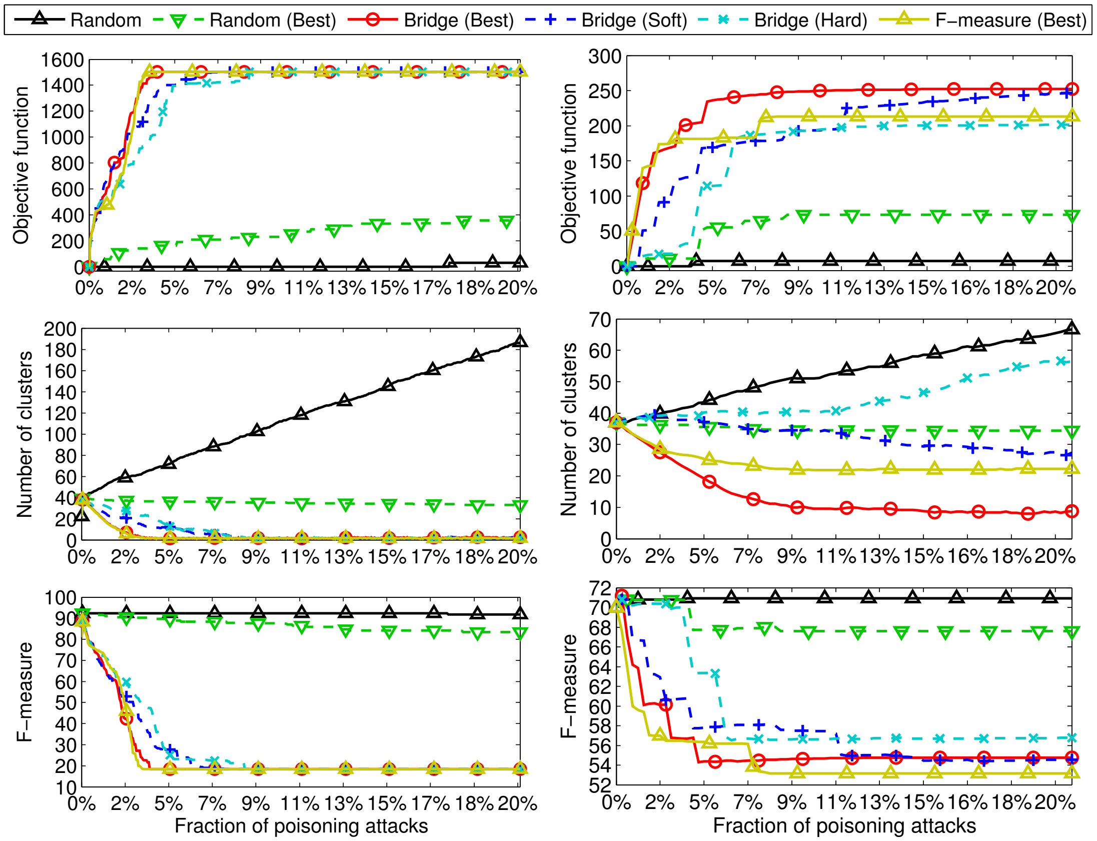

Figure 3: Results for the *Malheur* dataset (left column) and the *Recent Malware* dataset (right column).

specific feature representations remains a non-trivial open issue, which should be addressed as done in this paper for poisoning attacks and behavioral malware clustering.

8. ACKNOWLEDGMENTS

We thank Cristian Milia for supporting during tests. 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.* Furthermore, we acknowledge funding from BMBF under the project PROSEC (FKZ 01BY1145). 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.

9. REFERENCES

1. M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Recent Adances in Intrusion Detection (RAID)*, pages 178–197, 2007.
2. M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, New York, NY, USA, 2006. ACM.
3. U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2009.
4. B. Biggio, S. R. Bulò, I. Pillai, M. Mura, E. Z. Mequanint, M. Pelillo, and F. Roli. Poisoning complete-linkage hierarchical clustering. In *Structural, Syntactic, and Statistical Pattern Recognition*, 2014, In press.
5. B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Part III*, volume 8190 of *Lecture Notes in Computer Science*, pages 387–402. Springer Berlin Heidelberg, 2013.
6. B. Biggio, I. Corona, B. Nelson, B. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli. Security evaluation of support vector machines in adversarial environments. In Y. Ma and G. Guo, editors, *Support Vector Machines Applications*, pages 105–153. Springer International Publishing, 2014.
7. B. Biggio, G. Fumera, and F. Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, April 2014.

8. 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.

9. B. Biggio, I. Pillai, S. R. Bulò, D. Ariu, M. Pelillo, and F. Roli. Is data clustering in adversarial settings secure? In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 87–98, New York, NY, USA, 2013. ACM.

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

11. M. Brunner, M. Epah, H. Hofinger, C. Roblee, P. Schoo, and S. Todt. The fraunhofer aisec malware analysis laboratory. Technical report, Fraunhofer Institute AISEC, 2010.

12. P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2006. USENIX Association.

13. G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of USENIX Security Symposium*, 2008.

14. G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2008.

15. F. Guo, P. Ferrie, and T. Chiueh. A study of the packer problem and its solutions. In *Recent Advances in Intrusion Detection*, 2008.

16. X. Hu and K. G. Shin. DUET: integration of dynamic and static analyses for malware clustering with cluster ensembles. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2013.

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, October 2011.

18. iSeclab. Anubis. `http://anubis.iseclab.org`, visited April, 2014.

19. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

20. J. Jang, D. Brumley, and S. Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, pages 309–320, 2011.

21. Kaspersky Lab. KASPERSKY SECURITY BULLETIN 2013. `http://media.kaspersky.com/pdf/KSB_2013_EN.pdf`, 2014.

22. M. Kloft and P. Laskov. Online anomaly detection under adversarial impact. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 405–412, 2010.

23. A. Kolcz and C. H. Teo. Feature weighting for improved classifier robustness. In *Sixth Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 16/07/2009 2009.

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

25. R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 391–404, 2010.

26. R. Perdisci and M. U. VAMO: towards a fully automated malware clustering validity analysis. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2012.

27. K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 108–125, July 2008.

28. K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4):639–668, 2011.

29. K. Tan, K. Killourhy, and R. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Recent Adances in Intrusion Detection (RAID)*, pages 54–73, 2002.

30. K. Tan and R. Maxion. "Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector. In *Proc. of IEEE Symposium on Security and Privacy*, pages 188–201, 2002.

31. P. Trinius, C. Willems, T. Holz, and K. Rieck. A malware instruction set for behavior-based analysis. In *Proc. of GI Conference "Sicherheit" (Sicherheit, Schutz und Verlässlichkeit)*, pages 205–216, Oct. 2010.

32. C. J. van Rijsbergen. *Information Retrieval.* Butterworth, 1979.

33. VirusTotal. https://www.virustotal.com.

34. D. Wagner and P. Soto. Mimicry attacks on host based intrusion detection systems. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, pages 255–264, 2002.

35. C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2):32–39, 2007.