# Quantifying the Vulnerability of Anomaly Detection Implementations to Nondeterminism-based Attacks

Muyeed Ahmed Iulian Neamtiu
*Department of Computer Science*, *New Jersey Institute of Technology*, Newark, NJ, USA
{ma234, ineamtiu}@njit.edu

***Abstract*—Anomaly Detection (AD) is widely used in security applications such as intrusion detection, but its vulnerability to nondeterminism attacks has not been noticed, and its robustness against such attacks has not been studied. Nondeterminism, i.e., output variation on the same input dataset, is a common trait of AD implementations. We show that nondeterminism can be exploited by an attacker that tries to have a malicious input point (outlier) classified as benign input (inlier). In our threat model, the attacker has extremely limited capabilities – they can only retry the attack; they cannot influence the model, manipulate the AD/IDS implementation, or insert noise. We focus on three concrete, orthogonal attack scenarios: (1) a restart attack that exploits a simple re-run, (2) a resource attack that exploits the use of less computationally-expensive parameter settings, and (3) an inconsistency attack that exploits the differences between toolkits implementing the same algorithm. We quantify attack vulnerability in popular implementations of four AD algorithms – IF, RobCov, LOF, and OCSVM – and offer mitigation strategies. We show that in each scenario, despite attackers' limited capabilities, attacks have a high likelihood of success.**

***Index Terms*—Anomaly Detection, Adversarial ML, Program Nondeterminism**

## I. Introduction

Anomaly, or outlier, detection (AD) is widely used, including in domains such as security, healthcare, or finance, where reliability is critical [1]. Due to nondeterminism in AD implementations, AD results are not stable across runs, which, as we show, makes AD exploitable. In computer security, such as intrusion detection (IDS), AD is increasingly preferred due to the growing threat of zero-day attacks, where signature-based systems are less effective [2]. IDS failures have severe implications: a single mis-identified/mis-classified point can lead to a security breach.[1]

When AD toolkit developers implement an AD algorithm, they make decisions such as when to use randomness, what "control knobs" (parameters) to offer users, and setting default parameter values. While developers' intention is to optimize performance, these decisions can lead to nondeterministic execution, which, as we show, is exploitable by adversaries.

Restarting a task, e.g., a process or thread implementing AD, is routine in modern data centers (e.g., Netflix's Chaos Monkey [4]), and is becoming more common, as major companies embrace "resilience engineering" where *a node restart is a feature, not a bug*. In this paper we show how such restarts, coupled with AD nondeterminism, open the door to attacks; we quantify the vulnerability of popular AD algorithms and implementations, and offer attack mitigation strategies.

In our model, the defender's AD classifies incoming input (e.g., packets, strings) into malicious or benign. The attacker tries to "sneak in" a malicious input (outlier) by having the defender AD's classify it as benign (inlier). Therefore, the fundamental concept in our work is a *flip*: a point whose class switches from outlier to inlier in a subsequent classification, due to nondeterminism, e.g., induced by a restart. A defense strategy susceptible to flips is vulnerable even to attackers with limited capabilities, whose only option is to retry the attack. As discussed shortly, modern IDS configurations favor such simple, low-cost, retry attacks.

We consider three realistic scenarios: a *restart attack* where the AD implementation is restarted (re-run) with the same parameters; a *resource attack* where the defender adjusts the AD parameters to attain faster anomaly detection, though increasing the mis-classification rate, and an *inconsistency attack* where the attacker leverages the fact that the defender has switched the AD implementation to a different toolkit. These attacks are orthogonal (independent): they can be employed separately or combined.

In Sections IV to VI we quantify vulnerability to restart, resource and inconsistency attacks, respectively, and propose effective mitigation strategies. We examined 4 AD algorithms (Isolation Forest, Robust Covariance, One Class SVM, and Local Outlier Factor), implemented in 3 popular toolkits (Scikit-learn, R, and Matlab). In all, we conducted 68,190 experiments on 55 datasets. We found that flips are widely-spread, hence exploitable, across implementations, algorithms, and datasets. We make the following contributions:

- We introduce three novel attack scenarios, (1) restart attack, (2) resource attack, and (3) inconsistency attack.
- We demonstrate and quantify the vulnerability of four popular AD algorithms – IF, RC, LOF, and OCSVM – in the context of the aforementioned attacks.
- We provide an analysis of the causes of such vulnerabilities and present mitigation strategies to reduce them.

## II. Nondeterminism Attacks

We first discuss nondeterminism and instability; next, we define flipping formally; and then argue that attacks are essentially free.

**Program nondeterminism and instability.** Program determinism is defined as a program always producing the

[1] "The malicious activity affects one data sample, e.g., a packet [...] An attack leaves a footprint in a single record" [3].

same output on a given input. The negative consequences of nondeterminism have mainly been studied in non-ML settings, e.g., in the context of concurrent programs, where nondeterminism leads to races, crashes, or incorrect output [5]. The ML literature has extensive studies on the *stability* of ML models, but their focus is primarily on how sensitive the algorithm's output (solution) is to changes in the input data [6], [7], [8], [9], rather than on deterministic stability, i.e., changes in the algorithm's output when there is no change in the input data, and its security implications.

**Definitions: outlier detection and flipping.** Let $D = \{x_1, x_2, \ldots, x_m\}$ be a dataset with $m$ points where each point $x_i$ is a $d$-dimensional vector over the real space. The AD output consists of outliers $O$ and inliers $I$;[2] $O$ and $I$ form a partition of $D$, i.e., $I \cup O = D$ and $I \cap O = \emptyset$. For a specific run $i$, we denote its output (outliers and inliers) as $O_i$ and $I_i$, respectively. Let $j$ be a run subsequent to $i$, with output $O_j$ and $I_j$. The following hold: $I_i \cup O_i = D$, $I_i \cap O_i = \emptyset$, and $I_j \cup O_j = D$, $I_j \cap O_j = \emptyset$. However due to nondeterminism, there exist points $x_f$ such that $x_f \in O_i \wedge x_f \in I_j$ or $x_f \in I_i \wedge x_f \in O_j$. We name such points *flips* because their class label has flipped between runs, from outlier to inlier or vice versa.

Let $F$ be the set of flippable points in $n$ runs, defined as:

$$F = \bigcup_{j=2,n} \{x_f | (x_f \in I_j \wedge x_f \in O_{j-1}) \vee (x_f \in O_j \wedge x_f \in I_{j-1})), 1 \leq f \leq m\}$$

i.e., the union of points that can flip in each execution $j$ with respect to a prior execution. For example, given three runs where points $\{x_a, x_b\}$ flip between $\text{Run}_1$ and $\text{Run}_2$, while $x_c$ flips between $\text{Run}_2$ and $\text{Run}_3$, then $F = \{x_a, x_b, x_c\}$. We define $\frac{|F|}{|D|}$ as the flippable ratio, i.e., the fraction of all points that can flip across $n$ executions.

When ground truth is available, we can define true outliers *TO* (actual malicious input) and true inliers *TI*, as well as false outliers and inliers, *FO* and *FI*. Let $TO_i$ be the set of points correctly classified as outliers in run $i$ and $FI_j$ be the points that are incorrectly classified as inliers in a subsequent run $j$. A *successful attack* happens when $TO_i \cap FI_j \neq \emptyset$ because one or more points flip from an outlier classification in run $i$ to an inlier classification in run $j$. In other words, while the defender will correctly reject one such point in run $i$, the defender will then incorrectly accept that point in run $j$, and the attack succeeds. For brevity, we denote this set as $TO_i \rightarrow FI_j$. Note that a $TI \rightarrow FO$ flip is undesirable as well, causing the defender to reject legitimate input.

We define $\frac{|TO_i \cap FI_j|}{|O|}$ as the flippable ratio from TO to FI between runs $i$ and $j$. Where, $TO_i$ be the set of points that are correctly classified as outliers by run $i$, and $FI_j$ the set of points incorrectly classified as inliers by run $j$. Similarly, we define $\frac{|TI_i \cap FO_j|}{|I|}$ as the $TI_i \rightarrow FO_j$ flippable ratio.

[2]An outlier is an extreme or exceptional data point that deviates significantly from the other points, called inliers. Outliers are detected by finding significant deviations from the inlier group, e.g., using density or distance metrics.

To sum up, the attacker aims to "sneak in" a malicious input by leveraging flipping, i.e., the existence of "attack" points, denoted $x_{attack}$ where $x_{attack} \in F$ whose classification flips from outlier to inlier. Our goal is to quantify the vulnerability of popular AD implementations to flipping.

**Attacks are essentially free.** Modern cyber assets are at high risk of being attacked. For example, studies indicate that bad bots are responsible for 25%–30% of Internet traffic [10]. To counter this, Intrusion Detection Systems (IDS) are used anywhere from core Internet routers to firewalls, to anti-malware running on PCs, phones, or IoT devices. We focus on the scenario where IDS employ AD techniques. Prior work on adversarial AI (Section VII) has made strong assumptions, e.g., the attacker can influence the model, manipulate the AD/IDS implementation, insert noise, or incrementally craft input. In contrast, in our threat model, attackers have extremely limited capabilities – they simply retry the attack, without having to actually *induce* a defender-side event (e.g., restart, performance degradation, or implementation switching). In the current threat environment, attackers can conduct free, persistent retry attacks, e.g., 10–10,000 attacks per hour, which, thanks to node/task restarts, will exploit any of the restart/resource/inconsistency vulnerabilities at the defender.

## III. Algorithms and Experimental Setup

We now describe the AD algorithms we studied and the experimental setup.

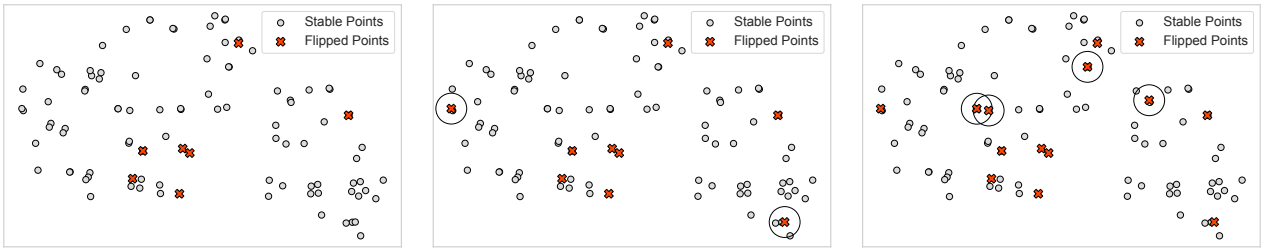
### A. AD Algorithms

We studied 4 AD algorithms: Isolation Forest [11], Robust Covariance [12], One Class SVM [13], Local Outlier Factor [14]. We selected these algorithms as they are popular and have been used in AD benchmarking [1]. We present an overview of each algorithm.

**Isolation Forest** (IF) identifies anomalies in a dataset by randomly dividing the data into subsets and assessing how many divisions are needed to isolate an anomaly: samples that can be separated from the rest with fewer partitions are deemed anomalies. Parameters that influence flipping include (a) *number of estimators*, determining how many estimators the algorithm will create to produce the final result, (b) *samples per estimator*, i.e, the number of samples each estimator examines, and (c) *features per estimator*, setting the fraction of features that each estimator explores.

**Robust Covariance** (RobCov) uses a robust covariance matrix, less affected by extreme data points, to identifies anomalies based on distance: data points with large distances are considered anomalies. Two parameters that affect flipping are (a) *covariance estimation*, which determines the method used to generate the covariance matrix, and (b) *support fraction*, i.e., the portion of the dataset that contributes to the Minimum Covariance Determinant [12].

**One Class SVM** (OCSVM) creates a boundary around normal data points by optimizing a hyperplane's margin in a high-dimensional space. Points outside the boundary are considered anomalies. The most important parameters impacting




(a) $\text{Run}_2$: 7 flips (b) $\text{Run}_3$: 2 new flips (9 accumulated flips) (c) $\text{Run}_4$: 4 new flips (13 accumulated flips)

Fig. 1. Restart attack illustrating the growth of flipped points (shown encircled), as runs accumulate (Sklearn/IF on dataset fertility).

flips are the *kernel function* (e.g., linear, polynomial, RBF), and kernel-specific parameters, (e.g., the RBF $\gamma$, or the degree of polynomial kernels).

**Local Outlier Factor** (LOF) measures the "degree of being an outlier" [14] of each data point by quantifying how it deviates from the typical neighborhood density. LOF determines this degree, i.e., the anomaly score of a point, by comparing its local density to the average density of its $k$-nearest neighbors. Points with lower scores are considered potential anomalies depending on a user-specified threshold, suggesting they are isolated from their local neighborhood. Key flipping parameters in LOF include (a) $k$, the number of neighbors, (b) the *threshold* used to separate normal from anomalous scores, and (c) the *distance metric*.

### B. Experimental Setup

We studied the 4 algorithms' implementations in 3 popular toolkits: Scikit-learn (Sklearn for short), R, and Matlab. We used 55 AD-specific datasets from ODDS [15] and the UCI ML repository [16]. We now discuss each attack and our findings in detail.

## IV. Restart Attack

### A. Definition and Attack in Practice

**Definition.** Let us assume that the attacker wants their input (point), denoted $x_{attack}$, classified as an inlier, but was unsuccessful in Run 1, i.e., the defender's AD implementation $T$ has classified $x_{attack}$ as outlier ($x_{attack} \in O$). The attacker then re-launches the attack and leverages the nondeterminism that comes included with a defender re-running the AD implementation. If, in a subsequent run, e.g., Run 2, due to nondeterminism, the defender classifies $x_{attack}$ as inlier ($x_{attack} \in I$), the attack succeeds.

**Attack in practice.** Server-side software, e.g., as used for Intrusion Detection, has been using a thread-per-request or process-per-request model for decades [17]. In this model, a new task (thread or process) is created for running the AD implementation $T$ to classify input $x$. New tasks run the same code $T$, but due to nondeterminism, the class of $x$ can flip from one run to the next, e.g., from outlier to inlier. Restarting a task is already routine in modern data centers, and is becoming more common. The philosophy of *a node restart is a feature, not a bug* has been embraced by major companies, e.g., Netflix's Chaos Monkey [4], Amazon's GameDay [18], Etsy's fault injection [19], or Facebook's Project Storm [20]. Thanks to increasing adoption of frequent restarts, attackers can increase their success rate by simply retrying a request, because restarting an implementation $T$ induces nondeterministic behavior. The current IDS environment favors attackers, by allowing them the opportunity to retry attacks at negligible cost. For example, CloudFlare[3] suggests the following rate-limits: 20 login requests per hour and 5 query requests per second [21]. Hence the attacker can easily retry close to 20 login attacks and 18,000 ($5 \times 3{,}600$) query attacks per hour before raising a flag.

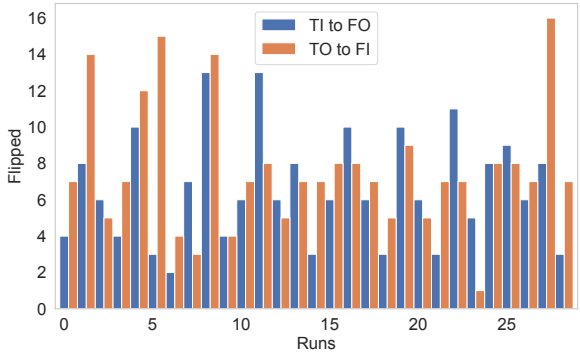



Fig. 2. Sklearn/IF: flipped points after each run on dataset ionosphere.

### B. Examples of Flipping Behavior

**Flips are common.** We first illustrate how flipped points accumulate with successive runs, which increases an attacker's success rate. Figure 1 shows[4] the impact of flipping across 4 Sklearn/IF runs. After an initial run $\text{Run}_1$, we ran the toolkit again on the *same dataset*, fertility, with the *same, default parameters*. The $\text{Run}_2$ outcome is shown in Figure 1 (a). The round points are "stable", i.e., have retained their classification, either inlier or outlier, from $\text{Run}_1$. However, for 7 points, indicated with '×', the algorithm has flipped their classification from inlier to outlier or vice versa. During $\text{Run}_3$ (Figure 1 (b)), 2 more points have flipped classification, leading to a cumulative count of 9 flipped points across 3 runs (or 2 restarts). Subsequently, in $\text{Run}_4$ (Figure 1 (c)), we observed 4 more point flips, hence *in total across three restarts, 13 points*

[3]A prominent CDN and cybersecurity service provider.

[4]Note that the figure is a two-dimensional projection of a 9-dimensional dataset; while some points appear close in the two-dimensional space, they might be substantially distant in 9-dimensional space.

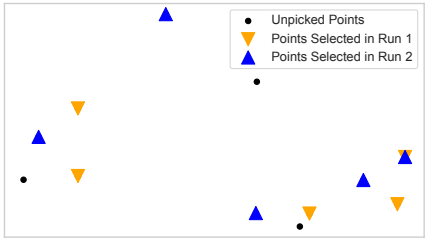



Fig. 3. Sklearn/IF & Matlab/IF: reason behind flips.

*have flipped*. After 30 runs conducted on this dataset consisting of 100 points, a total of 29 points flipped their classification.

**Flipping behavior across runs.** We now illustrate flipping behavior across a longer sequence of runs, focused on attacker-favorable *TO→FI* flips. Figure 2 shows flipping in Sklearn/IF, run 30 times with default parameters, on dataset ionosphere. Each point on the x-axis represents a new run; the bars indicate the number of points that flip in that run, compared to the previous run. The first two bars, corresponding to *x=0*, show the effect of the first restart – points that flipped when comparing the second run, "$\text{Run}_2$", to $\text{Run}_1$. We found that after this first restart, 4 points flipped from *TI* to *FO* and 7 points flipped from *TO* to *FI*. When comparing $\text{Run}_3$ to $\text{Run}_2$, 8 points flipped from *TI* to *FO* and 14 points flipped from *TO* to *FI*. In the worst case, $\text{Run}_{28}$ to $\text{Run}_{29}$, *16 points flipped from TO to FI* and 8 points from *TI* to *FO*. The figure illustrates two characteristics of AD behavior: (a) flipping is a persistent behavior, run after run; (b) both types of flips, *TO→FI* and *TI→FO*, are present across a long streak of restarts.

**What causes flips?** Flips are due to (a) the design of AD algorithms, and (b) the parameters controlling AD algorithm implementations. Figure 3 illustrates flips in IF; this behavior is common across all IF implementations. IF creates a number of estimators, each containing a number of points randomly selected, as determined by the parameter *max_samples*. In this example we have 12 points and 1 estimator, and the *max_samples* is set to 5. Due to random selection, in two separate runs the estimator chooses two different 5-point sets. Using orange down-arrow, we indicate the points selected by the estimator in $\text{Run}_1$, and in blue up-arrow, points chosen by the estimator in $\text{Run}_2$. Notice how only one point out of five gets selected by the estimator in both runs. As a result, the *two runs will produce two different AD outcomes*.

### C. Vulnerability Quantified

We found that, for Sklearn/IF, all but one of the 55 datasets are susceptible to flips, as quantified next; the results for other toolkits will be discussed shortly.

**Fifty restarts.** Figure 4 shows the percentage of points that can be flipped at least once in 50 restarts when running Sklearn/IF with defaults. Each point on the x-axis represents a dataset, and the two corresponding bars represent the flippable percentages of *TI* to *FO*, i.e., $\frac{|TI \rightarrow FO|}{|TI|}$ (in blue) and *TO* to *FI*, i.e., $\frac{|TO \rightarrow FI|}{|TO|}$ (in orange). The datasets are sorted ascendingly by the percentage of points that flipped from *TO* to *FI*. For

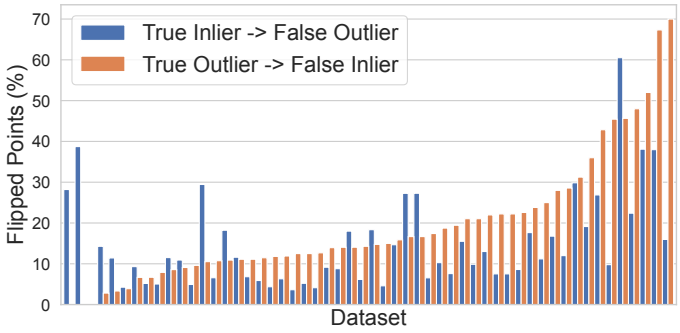



Fig. 4. Sklearn/IF: percentage of flipped points in each of the 55 datasets after **50 runs**; default setting.

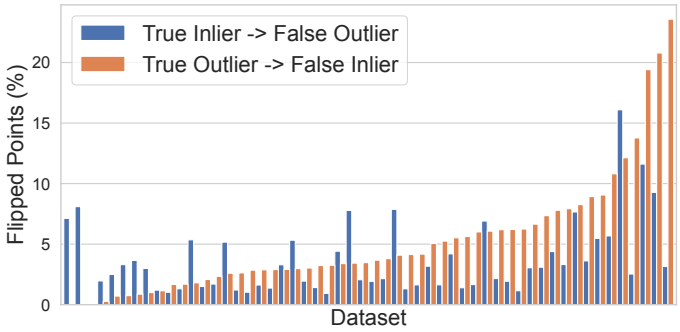



Fig. 5. Sklearn/IF: average percentage of flipped points in each of the 55 datasets after **1 Run**; default setting.

example, the dataset analcatdata_challenger (third from left), was the only dataset where we observed no flips. However, for the dataset pc1 (sixth from left) in 50 restarts, 4.3% of the true inliers were mis-classified as outliers and 3.9% of the true outliers were mis-classified as inliers. In the worst case, for the dataset wine (rightmost point on the graph), 70% of the outliers can be classified as inliers in at least one of the 50 restarts. The figure shows that (a) flips affect virtually any dataset, and (b) *TO→FI* flips, i.e., a malicious outlier input being erroneously classified as inlier after restart, are predominant.

**One restart.** In Figure 5 we show the typical percentage of points that can flip in a *single restart* (computed as the geometric mean across 50 runs). For example, for the dataset wbc, 1.4% of the *TI* will be re-classified as outliers in a typical restart; conversely, 3.2% of the *TO* will be re-classified as inliers. In the worst case, for dataset wine, on average 23.6% of *TO* can be re-classified as inliers in just a *single restart*. More than 5% points flipped from *TI* to *FO* for 12 datasets a single run, and for 21 datasets it took a single restart to convert more than 5% of the *TO* to *FI*. These findings illustrate the vulnerability of Sklearn/IF to just one restart.

### D. Effect of Run Order

So far we have not considered the effect of run *order* on attack success – rather we only looked at default restart outcomes. We now investigate how run order can affect the number of flips, hence the chance of a successful attack.

We first focus on dataset breastw, which has 64 flippable points out of 569 points in total. We studied three possible

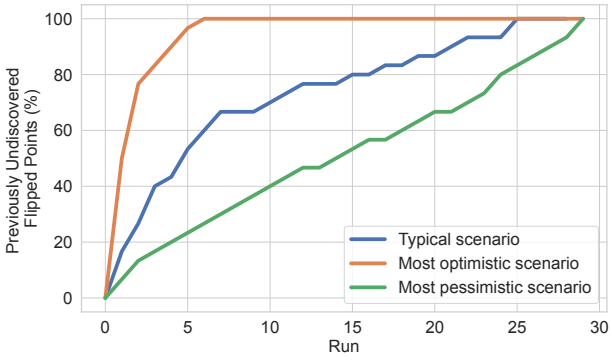



Fig. 6. Sklearn/IF: undiscovered flipped points as runs accumulate.

run orders: typical, optimistic, and pessimistic (from the perspective of the attacker), illustrated in Figure 6. In each case we show the CDF, i.e., the cumulative percentage of flippable points discovered as the number of runs increase.

*Typical.* When restarts follow a random order, in just 6 runs the attacker can flip 50% of flippable points; it takes 27 more runs to flip the remaining 50%.

*Optimistic.* This scenario favors the attacker: we sort runs in decreasing order of flipped points, i.e., the attacker is "lucky" in that the first runs have the highest number of flips. In this scenario, it takes just 8 restarts for the attacker to flip *all* flippable points at least once, and just *one restart* to discover 50% of flippable points.

*Pessimistic.* In this scenario, we sort runs in increasing order by the number of flipped points, i.e., the attacker is "unlucky" in that the first runs are those runs with a low number of flips, and later runs have an increasing number of flips. Note that even in this defender-favorable scenario, in 8 restarts, 20% of the total flippable points will be discovered, and to discover 50% of the total flippable points, it will take 23 restarts.

**Results across all datasets.** Our experiment shows that, for most datasets, 50% of the flippable points were discovered within 7 restarts; this holds for all implementations. To determine the number of restarts required to flip 50% of the points, we ran the implementations on the 55 datasets 50 times and calculated the number of flippable points for each dataset, then we determined how many restarts it will take to identify 50% of those flippable points. The median for each implementation ranged from 3 to 6, which suggests that no particular algorithm or implementation is immune to flips.

### E. Vulnerability Mitigation via Increased Determinism

We now discuss an attack mitigation strategy – increasing determinism – and quantify its effectiveness. Our insight is to run AD in an optimized configuration, designed to reduce nondeterminism, hence reduce the chance of a restart attack. We first examine Sklearn/IF in two configurations: default configuration (parameter settings) which has high nondeterminism, and optimized, low nondeterminism configuration.

**Default, high-nondeterminism settings.** Table I's "Default" columns show the results: the number (#) and percentage (%) of points that flip, and the flip direction. We show the top-5 datasets in terms of flip percentage, as well as results

TABLE I
RESTART ATTACK: POINTS FLIPPED IN A SINGLE RUN: # OF FLIPS AND AS % OF *TI* (OR *TO*, RESPECTIVELY).

| **Dataset** | **Points Flipped** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Default | | | | Optimized for max determinism | | | |
| | *TI→FO* | | *TO→FI* | | *TI→FO* | | *TO→FI* | |
| | # | % | # | % | # | % | # | % |
| *Sklearn/IF* | | | | | | | | |
| climate-model | 80 | 16 | 6 | 11.7 | 13 | 2.5 | 1 | 1.7 |
| v._livestock | 12 | 12 | 5 | 18.5 | 0 | 0 | 0 | 0 |
| optdigits | 470 | 9.3 | 31 | 20.3 | 78 | 1.5 | 3 | 2.2 |
| arsenic-f.-bladder | 37 | 7.9 | 7 | 9.2 | 17 | 3.3 | 3 | 4.3 |
| arsenic-f.-lung | 43 | 8.1 | 1 | 4.5 | 19 | 3.5 | 1 | 2.8 |
| *All datasets* | | | | | | | | |
| *Sklearn/IF* | | *2.9* | | *4.4* | | *1.2* | | *1.4* |
| *Sklearn/RobCov* | | *1.1* | | *2* | | *0.9* | | *1.2* |
| *Matlab/RobCov* | | *0.2* | | *0.1* | | *0* | | *0* |
| *Matlab/IF* | | *1.8* | | *3.9* | | *0.8* | | *1.4* |
| *Matlab/OCSVM* | | *13.8* | | *17* | | *2.8* | | *3.7* |

across all datasets. Datasets such as optdigits are among the most vulnerable with 31 points (20.3%) of the outliers being re-classified as inliers, and 470 points (9.3%) of the inliers being re-classified as a outliers after restart. This shows a concerning picture, where 501 points can flip in just one restart. Overall, across all datasets, with default parameter settings, on average 2.87% of *TI* flip to *FO* in a single restart, but the more concerning part is that 4.43% of the outliers were mis-classified as inliers (*TO→FI*).

**Optimized, low-nondeterminism settings.** For each dataset, we explored parameter settings that yield the lowest flipping rates, to obtain an idea of the measures defenders can take, and the efficacy of those measures. To increase Sklearn/IF determinism we used a higher number for *n_estimators* (number of estimators used to predict the final output, by default set to 100). We also tuned the *max_samples* parameter, that determines the number of points used to generate an estimator (by default set to 256). For example, for the dataset climate-model-simulation-crashes we set *n_estimators* to 500 and *max_samples* to 0.5, i.e., 50% of the sample size. This change (shown in the first data row) reduced the *TI→FO* flip from 16% to 2.5% and *TO→FI* from 11.7% to 1.7%. Overall, for Sklearn/IF the *TI→FO* flips went from 2.87% to 1.21% and *TO→FI* flips reduced from 4.43% to 1.43%.

### F. Results for all Toolkits

Besides Sklearn/IF, we also tested the other nondeterministic implementations: RobCov in Sklearn, as well as RobCov, IF, and OCSVM in Matlab. Table I's "Optimized" columns show the results, confirming that our mitigation strategies were effective across all algorithms. With Sklearn/RobCov, on average we observed 1.1% *TI→FO* flips in default settings, and 0.9% in optimized settings; for *TO→FI*, we observed a 2% flip in default settings and 1.2% in optimized settings. This improvement was primarily due to using the optimum value for the *contamination* parameter. With Matlab/RobCov we observed the least amount of *TI→FO* and *TO→FI* flips, where in default settings the averages over 55 datasets were only 0.2% and 0.1%, respectively. We observed no flips in 42

out of the 55 datasets. After optimizing the settings – in this case using the most effective value of the *Method* parameter for each dataset, we were able to increase the number to 55, i.e., no flips in any dataset. In Matlab/OCSVM we observed the most flips in default settings: on average, 13.8% *TI→FO* flips and 17% *TO→FI* flips. Optimizing the hyperparameters, i.e., *ContaminationFraction*, *KernelScale*, *Lambda*, and *BetaTolerance*, reduced flipping 4.5x–4.9x: *TI→FO* and *TO→FI* were reduced to 2.8% and 3.7% respectively.

**Lack of correlation between flips and size, attributes.** We performed a correlation analysis between restart flip percentage and dataset size (#instances aka points) as well as #attributes (aka dimensions). The analysis indicates little correlation; ranging from -0.13 to +0.1 for #instances, and from -0.11 to 0.45 for #attributes. These results indicate that flipping behavior is pervasive across datasets with varying sizes and dimensionalities.

**Concluding remarks and mitigation strategy.** While nondeterminism is systematic, it can be reduced via hyperparameter tuning, thereby reducing the vulnerability potential.

## V. Resource Attack

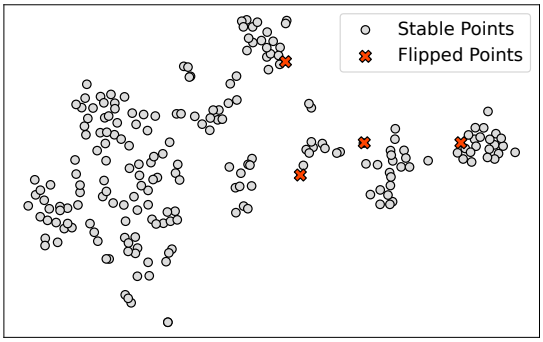
### A. Definition and Attack in Practice

**Definition.** Certain parameters govern AD algorithm complexity, e.g., in IF, the number of estimators (*n_estimators*) or estimator sample size (*max_samples*) [11]. Other algorithms have parameters with a similar functionality.[5] Therefore in practice parameter settings are used to balance precision and efficiency. Note that high efficiency ("efficiency" as in speed) is essential for effective, real-time security [23], [24], therefore AD-based security deployments might decide to switch to a higher-efficiency (but potentially lower accuracy) AD. The impetus to switch to a more resource-friendly AD configuration is even higher under load. Therefore, in this threat model, the attacker leverages the fact that "degrading" a low-flip but slow implementation $T$ to a fast but high-flip implementation $T_{degraded}$ increases the attack success rate.

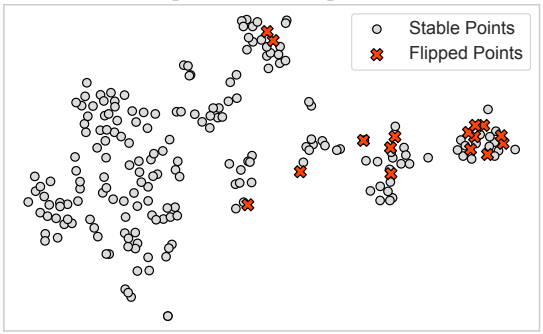
More precisely, we assume that the attacker was unsuccessful in Run 1, as the defender's AD implementation $T$ has classified $x_{attack}$ as outlier. The attacker waits for the defender to run in degraded mode $T_{degraded}$, i.e., the same implementation $T$ but with modified parameters to increase throughput at the expense of precision. If, in subsequent runs, e.g., Run 2, due to the $T \rightarrow T_{degraded}$ switch, the defender system classifies $x_{attack}$ as inlier, the attack succeeds.

**Attack in practice.** The attacker has several options, depending on how urgently they want the attack to succeed. For *non-urgent* scenarios, the attacker can launch the attack for free during high-traffic, resource-intensive periods such as Black Friday or Christmas. For *urgent* scenarios, the attacker can force the defender into a resource crunch by running a Distributed Denial of Service (DDOS) attack, forcing the defender to degrade service. A DDOS attack is easy to purchase and cheap, around $5–$20 per hour [25], [26].

[5]For LOF, lowering $k$ (number of nearest neighbors) reduces running time [11], [14]. For OCSVM, *kernel* selection can determine running time: a *linear kernel* is faster than a *polynomial kernel*, and the latter's time increases with the polynomial degree [22].




(a) Flips with default parameters




(b) Flips with resource-friendly parameters (resource attack)

Fig. 7. Resource attack: increase in vulnerable points when switching to a resource-friendly parameter configuration (Sklearn/IF on dataset glass).

### B. Example: Flipping Behavior When Changing Settings

Figure 7 provides a visual representation of the significant impact that different configurations can have on the number of flips. In Figure 7 (a) default settings were used, resulting in 4 points, indicated with '×', that flip in one restart. However, in Figure 7 (b), we chose a more resource-friendly setting, i.e., reducing *n_estimators* to 50 and *max_samples* to 64, where the implementation runs *twice as fast*. The contrast is immediately evident: 16 points changed classification in a single restart. Hence the second configuration achieves results twice as fast, but at a substantial security loss: a 4x increase in vulnerability.

**Why does vulnerability increase in resource-constrained environments?** We now illustrate how parameter selection that might increase performance can also increase flipping. Figure 8 shows an example of two runs. In one, computationally-expensive run, the parameters are set so the IF implementation uses 5 samples for each estimator, indicated by the orange down-arrow; in the other, computationally-light run, we use 2 samples for each estimator, indicated by blue up-arrow. Due to selecting a lower number of samples in the second run, the estimator will require less time, leading to a faster run; however, with the latter setting, the IF implementation's inlier/outlier estimation will be less reliable.

### C. Efficiency vs. Flipping Trade-off

We begin with a study of efficiency vs. flipping trade-off in Sklearn/IF and Matlab/IF. Recall that the algorithm

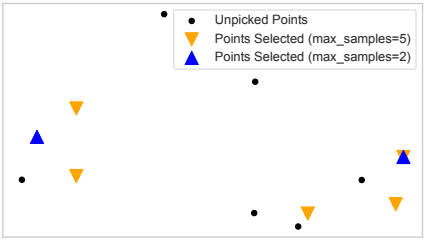



Fig. 8. Sklearn/IF & Matlab/IF: a lower *max_samples* increases vulnerability in a resource-friendly environment.

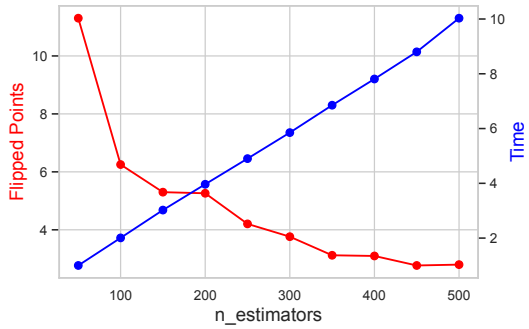



Fig. 9. Sklearn/IF & Matlab/IF: *n_estimators*' effect on time and flipped points (dataset: spambase).

uses a parameter that determines the number of estimators to create (named *n_estimators* in Sklearn and *NumLearners* in Matlab). The estimators are independent of each other, hence if creating one estimator takes time $t$, creating $n$ estimators takes $n * t$. On dataset spambase, we conducted experiments with 10 increasing *n_estimators* values: $[50, 100, ..., 500]$. Figure 9 illustrates the efficiency vs. flipping trade-off: the percentage of flipped points is shown in red, while the execution time (normalized with respect to the base, and fastest value *n_estimators=50*) is shown in blue. We observed a linear increase in execution time as *n_estimators* increases. For example, increasing *n_estimators* from 50 to 100 doubles the AD execution time. More importantly, the graph shows that this parameter effectively dictates attack resilience, i.e., the fraction of points that can be flipped: lowering *n_estimators* increases flip likelihood. When *n_estimators=50*, an average of 11.3 points flipped, whereas when *n_estimators=500*, only 2.8 points flipped on average. We observed similar trade-offs with other parameters of IF, such as *max_samples* and *max_features*, and in other implementations, for instance, *support_fraction* in Sklearn/RobCov.

### D. Efficiency and Attack Resistance in IF

These three examples have illustrated the trade-off between flip ratio and efficiency (execution time) on individual datasets. We now use statistical measures to study the impact of efficiency on attack resistance, in IF; the results for the other algorithms will be discussed shortly, in Section V-E. We compare defaults (*n_estimators=100* and *max_samples=256*) to a fast (efficient) setting, which reduced the number of estimators and sample size (to *n_estimators=50* and *max_samples=64*) and ran it on the 55 datasets. This reduced AD run time by around 50%, but in doing so it increased vulnerability.

TABLE II
RESOURCE ATTACK, POINTS FLIPPED IN A SINGLE RUN: # OF FLIPS AND AS % OF *TI* (OR *TO*, RESPECTIVELY).

| **Dataset** | **Points Flipped** | | | | | | |
|---|---|---|---|---|---|---|---|
| | Fast (imprecise) | | | | Compared to default (multi. runs) | | |
| | | | | | Flip increase | | Time |
| | *TI→FO* | | *TO→FI* | | *TI→FO* | *TO→FI* | Saved |
| | # | % | # | % | | | % |
| *Sklearn/IF* | | | | | | | |
| climate-model. | 84 | 17.8 | 5 | 12.2 | 1.11x | 1.04x | 51 |
| v._livestock | 14 | 12.7 | 5 | 20 | 1.06x | 1.08x | 52 |
| optdigits | 1102 | 22 | 38 | 23.7 | 2.37x | 1.17x | 45 |
| ars.-f.-bladder | 64 | 14 | 8 | 10.4 | 1.77x | 1.13x | 52 |
| ars.-f.-lung | 72 | 14 | 1 | 4.8 | 1.73x | 1.07x | 53 |
| All datasets | | | | | | | |
| *Sklearn/IF* | | *6.26* | | *5.71* | *2.16x* | *1.3x* | *50* |
| *Sklearn/RobCov* | | *2.7* | | *4.9* | *2.45x* | *2.45x* | *9* |
| *Matlab/IF* | | *1.9* | | *3.2* | *1.04x* | *1.22x* | *48* |

Table II quantifies efficiency vs. attack resistance. The 'Fast' columns show the number and percentage of points that flipped when using the efficient parameter settings, whereas the last three columns compare the efficient configuration to the default (slow) configuration. The third-from-last and second-from-last columns show the increase in the number of flips when using the fast configuration. The last column shows time saved by the fast configuration.

The first five rows in Table II show the results for datasets where the number of flips increased the most. For example, for optdigits, time decreased by 45%, but the percentage of points that flip from *TI* to *FO* increased from 9.3% (rate for default parameters, omitted from the table for brevity) to 22%, while *TO→FI* flips increased from 20.3% to 23.7%. Similarly, for dataset arsenic-female-bladder the time decreased by 52%, but *TI→FO* flips increased from 7.9% to 14%, whereas *TO→FI* flips increased from 9.3% to 10.4%. Overall, the execution time reduced by 50%, but the percentage of points that can flip from *TI* to *FO* more than doubled, from 2.87% to 6.26%, while the percentage of points that can flip from *TO* to *FI* increased from 4.43% to 5.71%.

### E. Results for all Toolkits

Besides Sklearn/IF whose aggregate results are shown in the third-to-last row, we also experimented on four other nondeterministic implementations. We discuss Sklearn/RobCov and Matlab/IF, shown in the last two rows of Table II; for the other two implementations, Matlab/RobCov and Matlab/OCSVM, the time savings were negligible hence we omit the flip ratio figures. For Sklearn/RobCov and Matlab/IF we were able to reduce execution time by 15.46% and 50.72%, respectively. However, this led to a increase in flip percentage, where for Sklearn/RobCov both *TI→FO* and *TO→FI* flips increased by 2.45x. For Matlab/IF, *TI→FO* increased by 1.04x and *TO→FI* increased by 1.22x. Similar to Section IV-F, there was little correlation, if any, between flip percentage and dataset size (number of instances) as well as number of attributes. Overall,

Table II shows that *the defender pays a steep price when using a resource-efficient implementation*, as flip potential can increase substantially, e.g., between 1.22x–2.45x for *TO→FI*.

### F. Parameter-induced Trade-offs

We conducted experiments examining key parameters affecting AD operations, by quantifying the trade-off between efficiency and flipping. For each parameter, we varied its value $v$ from *min* to *max*, ran the AD toolkit on all datasets, and recorded the time, as well as flips, associated with each $v$. Next, we computed the $\beta$, i.e., the slope of a line where $v$ was on the x-axis while time (and flips, respectively) were on the y-axis. The goal of this experiment was to study the magnitude and sign of the $\beta$ as $v$ changes between *min* and *max*. These $\beta$ values across 55 datasets form a distribution. Ideally, the defender could find parameter values that minimize both time and flips. In practice though, this is only possible for some parameters, while for others, the defender must balance time against flips. For example, the time's $\beta$ distribution has typically positive values for *support_fraction* parameter in Sklearn/RobCov, with a median of 0.17. In contrast, the flips' $\beta$ distribution has negative values with a median of -0.76. Therefore, raising the value of *support_fraction* leads to time increases and flip decreases. In both Matlab/IF and Sklearn/IF, the parameters *n_estimators* and *max_samples* exhibit similar outcomes. In contrast, increasing the value of *max_features* consistently reduces flips without increasing execution time; a defender could hence choose a high value for *n_estimators* to reduce flipping potential.

**Concluding remarks and mitigation strategy.** Increasing throughput typically aggravates flipping. We quantify an efficient configuration's increase in vulnerability, compared to a default configuration. Our univariate parameter study has (1) shown that defenders have control over time, flips, and their balance, and (2) identified relevant parameters and their effectiveness – the higher the $\beta$, the more effective the control.

## VI. Inconsistency Attack

### A. Definition and Attack in Practice

**Definition.** For a given algorithm and dataset, the toolkit (implementation) has a marked effect on AD performance, typically measured via the F1-score, due to various factors, e.g., implementation strategy or hyperparameter settings [1], [27]. Beyond performance, AD users might have other reasons to switch toolkits, e.g., the emergence of a more suitable open source project [28] or moving away from a toolkit after multiple CVEs were found in that toolkit, or after a single critical vulnerability has been found [29]. When a toolkit change leads to flips, attackers can exploit the flips. Therefore, we quantify the vulnerability of AD to toolkit changes, which we name an *inconsistency attack* due to the inconsistencies between two implementations of the same algorithm.

More precisely, in this case the attacker takes advantage of the defender switching the AD implementation from $T_1$ to $T_2$, e.g., from R to Sklearn. We assume that the attacker was unsuccessful in Run 1, i.e., the defender's AD implementation

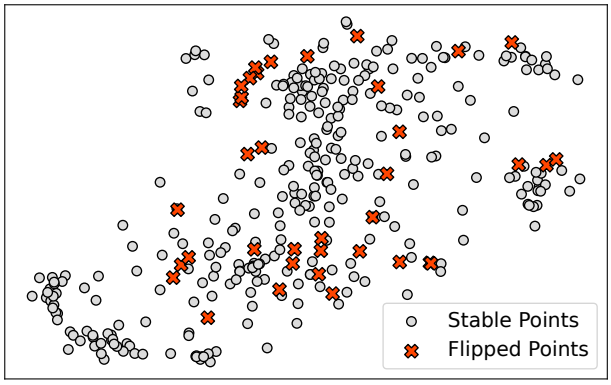



Fig. 10. Inconsistency attack: flips when switching LOF toolkits between Matlab and Sklearn (dataset: ionosphere).

$T_1$ has classified $x_{attack}$ as outlier. If, in a subsequent run, due to the switch from $T_1$ to $T_2$, the defender's AD classifies $x_{attack}$ as inlier, the attack succeeds.

**Attack in practice.** The attacker can use Banner Grabbing[6] to figure out the specific AD toolkit, say X, the defender is currently using, or learn when the defender switches toolkits from Y to X.

### B. Example: Flipping Behavior While Switching Toolkits

Figure 10 illustrates the potential challenges that can arise when transitioning between different toolkits: switching from one toolkit to another can lead to a *significant number of points undergoing a change in classification*. In this example, we ran LOF on dataset ionosphere in both Sklearn and Matlab and compared the output. The '×' points exhibit different outcomes when analyzed by the two toolkits. In other words, if the defender were to switch from using Sklearn to Matlab or vice versa, these marked points would flip their classification. Out of a total 351 points, 37 points switched classification. We now discuss inconsistency results for each algorithm.

### C. Isolation Forest

The IF algorithm is non-deterministic by design, so for every dataset, we ran the algorithm 50 times in each toolkit. We then compared each run of toolkit $A$ with each run of toolkit $B$ and calculated the minimum, geometric mean, and maximum percentage of *TI→FO* and *TO→FI*. We conducted our experiments on 3 toolkits, i.e., 6 toolkit switch scenarios. We used the same values of hyperparameters across all toolkits: *n_estimators=500*, while the number of samples and features used for each estimator was set to 50%.

Table III shows the results. We observed the highest percentage of *TO→FI* flips when we switched from Sklearn to R; typically 17.6% of *TO* flipped to *FI* (between 12.3% and 23.5%, depending on dataset). When switching in the opposite direction, R to Sklearn, we observed the highest percentage of *TI→FO*, with 6.81% typically and 8.74% in the worst case. These results are particularly concerning, as the numbers are computed across all datasets. Notably, for

[6]A technique to expose critical information regarding the software, and operating system, the defender is running [30].

TABLE III
ISOLATION FOREST: INCONSISTENCY ATTACK.

| Toolkits | Flipped (%) | | | | | |
|---|---|---|---|---|---|---|
| | *TI → FO* | | | *TO → FI* | | |
| | Min | Mean | Max | Min | Mean | Max |
| R → Matlab | 3.73 | 4.22 | 4.65 | 0.59 | 0.81 | 1.49 |
| Matlab → R | 0.38 | 0.46 | 0.57 | 9.8 | 11.6 | 12.1 |
| Sklearn → R | 0.02 | 0.09 | 0.17 | 12.3 | 17.6 | 23.5 |
| R → Sklearn | 5.03 | 6.81 | 8.74 | 0.23 | 0.22 | 0.15 |
| Sklearn → Matlab | 0.22 | 0.88 | 1.67 | 1.09 | 2.81 | 5.79 |
| Matlab → Sklearn | 0.99 | 1.95 | 3.35 | 0.49 | 1.13 | 1.87 |

TABLE IV
ONE CLASS SVM: INCONSISTENCY ATTACK.

| Toolkits | Flipped (%) | | | | | |
|---|---|---|---|---|---|---|
| | *TI → FO* | | | *TO → FI* | | |
| | Min | Mean | Max | Min | Mean | Max |
| R → Matlab | 4.5 | 8.1 | 12.7 | 39.4 | 43.2 | 46.9 |
| Matlab → R | 26.3 | 27.7 | 28.8 | 8.96 | 26.9 | 48.1 |
| Sklearn → R | 35.2 | 35.2 | 35.2 | 58 | 58 | 58 |
| R → Sklearn | 58.5 | 58.5 | 58.5 | 20.4 | 20.4 | 20.4 |
| Sklearn → Matlab | 3.4 | 6.6 | 10.1 | 80.1 | 82.9 | 89.4 |
| Matlab → Sklearn | 45.4 | 47.3 | 49.4 | 3.5 | 7.6 | 19.4 |

dataset analcatdata_chlamydia: *all TO predicted by Sklearn were classified as inliers by Matlab; all TI predicted by Matlab flipped to FO when we switched to Sklearn.*

### D. One Class SVM

This algorithm is deterministic in Sklearn and R, but not in Matlab. Therefore, we ran Matlab 50 times on each dataset, and compared its output with the outputs of Sklearn and R. We used the same value for common hyperparameters across toolkits, where supported, e.g., *kernel=Radial Basis Function*, *gamma=1/# of features*, *tolerance=0.001*. Table IV shows the results. Switching from Sklearn to R produces the highest *TI→FO* flips (58.5% on average). The switch from Sklearn to Matlab produces the highest *TO→FI* flip percentage, 80.05% on average. Again, these numbers are concerning as they indicate that *a majority of outliers are expected to flip when switching toolkits*, hence an attacker has a clear advantage.

### E. Local Outlier Factor

While the *Local Outlier Factor* (LOF) algorithm is by design deterministic, outputs differ between toolkits. To minimize the difference in hyperparameters, we used the same value *threshold=1.5*, per LOF's authors [14], and *k=20*, the default value for both Sklearn and Matlab. Table V shows the results. The flip potential is much lower than in IF and OCSVM, typically less than 1%. However, certain datasets exhibit higher flip values. For example, for dataset breastw we observed 22.9% (66 out of the 288) *TI→FO* flips when we switched from Sklearn to R. Similarly, when we switched from Matlab to R, 20.2% points flipped from *TI* to *FO*.

**Lack of correlation between flips and size/attributes.** We ran a correlation analysis between inconsistency flip percentage and #instances, as well as #attributes, for all datasets. We found little correlation: between -0.19 and +0.20 for #instances and between -0.47 and 0.44 for #attributes, but typically less than 0.1 in absolute value, which indicates that inconsistency is present across dataset sizes, algorithms, and implementations.

TABLE V
LOCAL OUTLIER FACTOR: INCONSISTENCY ATTACK.[M: MATLAB, S: SKLEARN]

| Toolkits | Flipped (%) | | Toolkits | Flipped (%) | |
|---|---|---|---|---|---|
| | *TI→FO* | *TO→FI* | | *TI→FO* | *TO→FI* |
| R → M | 0.16 | 0.35 | M → R | 0.18 | 0.40 |
| S → R | 0.28 | 0.39 | R → S | 0.14 | 0.32 |
| S → M | 0.15 | 0.15 | M → S | 0.03 | 0.05 |

**Concluding remarks and mitigation strategy.** We found that toolkit changes favor the attacker: inconsistency affects all algorithms and toolkits, with OCSVM the most prone to flips, and LOF the least prone. Therefore, defenders looking for flexibility in toolkit choice or anticipating toolkit changes might prefer LOF. For other algorithms, the defender can employ "parameter synchronization" between toolkits to reduce the attack surface, as we report next.

*IF.* For parameters that are supported across toolkits, such as *n_estimators*, *max_samples*, and *max_features*, a straightforward defensive measure when switching toolkits is to use the same values. However, synchronizing the contamination fraction parameter is more involved, as the three toolkits handle this parameter differently: Matlab specifies the fraction of outliers, while R specifies a threshold score. Sklearn also uses a fraction of outliers, but by default, it is set to '*auto*' (meaning it determines contamination using a default threshold value). Hence the defender can synchronize parameters according to the following equivalence scheme: $Sklearn(contamination = auto) \equiv R(threshold = 0.5)$; $Sklearn(contamination = n) \equiv Matlab(ContFraction = n), where\ n = (0, 0.5]$.

*LOF.* LOF toolkits use different threshold/contamination values, so the strategy, similar to the one just described for IF, is to synchronize these values between toolkits. Additionally, the defender must synchronize the choice of algorithm used for finding nearest neighbors. While both Matlab and R default to using Kd-tree, Sklearn employs a heuristic to determine the most suitable algorithm based on the dataset. Therefore, when transitioning from Sklearn, the defender must retain the algorithm choice of Sklearn for the particular dataset. Furthermore, parameters such as *k*, *distance metric*, and *leaf size*, need to be synchronized during the transition.

*OCSVM.* Here synchronization follows a similar strategy, though more default values need to be overridden: although all three toolkits employ the same default *kernel* type, the default values for *max_iter* and *ContaminationFraction/nu* differ among toolkits.

## VII. RELATED WORK

We found no prior work on attacks against unsupervised AD. Adversarial AI/ML work has mostly focused on attacks on neural networks (NN), which assume the attacker has the ability to manipulate input data to deceive the NN's model [31], manipulate/perturb graphs [32], [33], [34], insert noise into images [35], etc. We make no such assumptions.

Ahmed et al.'s work [27] studied 4 AD algorithm implementations in 3 toolkits. They found 5 out of 11 implementations to be non-deterministic and all 4 algorithms' implementations to be inconsistent. They also introduced a tool to reduce nondeterminism and inconsistency of these implementations, by determining the most suitable parameter setting for a given dataset [36]. However, they did not study flips, neither have they considered the security implications of AD nondeterminism. Their nondeterminism metrics were ARI between runs and F1 score, orthogonal to this work.

Tran et al. [37] showed that data quality can decisively influence ML-based IDS outcomes, and discussed systems for evaluating data quality in such settings; our work is primarily focused on nondeterministic behavior caused by AD implementations, rather than intrinsic data quality.

Perini et al. [38] introduced a method for evaluating the stability of ML models by creating multiple models using distinct subsets of the training data, and measuring their variance. Park [39] proposed a new loss function to improve Neural Network performance and stability. Gao et al. [40] proposed a method for quantifying the stability of an algorithm by removing elements from the sample set. All these efforts change the input set between runs, whereas we demonstrate attacks that do not change the input set. None of these efforts identified or measured flips, however.

## VIII. Conclusions

When AD is used in security applications, the adversarial angle needs to be studied and quantified. We show that classification flips, inherent in AD due to nondeterminism, can allow malicious input to "fly under the radar". We outline mitigation strategies, based on reducing the likelihood of flips, hence reducing the attackers' success rate.

## Acknowledgments

We thank Yao Ma and the anonymous reviewers for their valuable feedback. This material is based upon work supported by the National Science Foundation under Grant No. CCF-2007730.

## References

[1] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," *NeurIPS'22*, 2022.

[2] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[3] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," *IEEE TNSM'20*, 2020.

[4] Netflix, "Chaos monkey," *Netflix OSS*, 2024. [Online]. Available: https://netflix.github.io/chaosmonkey/

[5] S. Hong and M. Kim, "A survey of race bug detection techniques for multithreaded programmes," *STVR'15*, vol. 25, pp. 191–217, May 2015.

[6] T. Lange, M. Braun, V. Roth, and J. Buhmann, "Stability-based model selection," *NeurIPS'02*, vol. 15, 2002.

[7] R. D. Shah and R. J. Samworth, "Variable selection with error control: another look at stability selection," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 75, pp. 55–80, 2013.

[8] A. Rakhlin and A. Caponnetto, "Stability of $k$-means clustering," *NeurIPS'06*, vol. 19, 2006.

[9] H. Ahmed and J. Lofstead, "Managing randomness to enable reproducible machine learning," in *P-RECS'22*, 2022, pp. 15–20.

[10] Imperva, "Distribution of bot and human web traffic worldwide from 2014 to 2021," April 2022. [Online]. Available: https://www.statista.com/statistics/1264226/human-and-bot-web-traffic-share/

[11] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*. IEEE, 2008, pp. 413–422.

[12] P. J. Rousseeuw and K. V. Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, 1999.

[13] K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo, "One class support vector machines for detecting anomalous windows registry accesses," *ICDM Workshop on Data Mining for Computer Security*, 2003.

[14] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM SIGMOD*, 2000, pp. 93–104.

[15] "ODDS," January 2024, http://odds.cs.stonybrook.edu/.

[16] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[17] R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. Brewer, "Capriccio: Scalable threads for internet services," in *SOSP'03*, 2003.

[18] "Resilience engineering: Learning to embrace failure," *Commun. ACM*, vol. 55, no. 11, p. 40–47, nov 2012.

[19] J. Allspaw, "Fault injection in production: Making the case for resilience testing," *Queue*, vol. 10, no. 8, p. 30–35, aug 2012.

[20] R. Hof, "Interview: How Facebook's Project Storm Heads Off Data Center Disasters," *Forbes*, 2016. [Online]. Available: https://www.forbes.com/sites/roberthof/2016/09/11/interview-how-facebooks-project-storm-heads-off-data-center-disasters/

[21] Cloudflare, "Rate limiting best practices," 2024. [Online]. Available: https://developers.cloudflare.com/waf/rate-limiting-rules/best-practices/

[22] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[23] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," in *RAID '08*. Springer-Verlag, 2008.

[24] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag, "A high-performance network intrusion detection system," in *CCS'99*, 1999, p. 8–17.

[25] "Dark web price index 2020," *Privacy Affairs*, Dec 2022. [Online]. Available: https://www.privacyaffairs.com/dark-web-price-index-2020/

[26] D. Makrushin, "The cost of launching a ddos attack," May 2021. [Online]. Available: https://securelist.com/the-cost-of-launching-a-ddos-attack/77784/

[27] M. Ahmed and I. Neamtiu, "Anomalous anomaly detection," in *AITest'22*, 2022, pp. 1–6.

[28] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, "On the abandonment and survival of open source projects: An empirical investigation," in *ESEM'19*. IEEE, 2019, pp. 1–12.

[29] Cyber Safety Review Board, "Review of the december 2021 log4j event," *Department of Homeland Security*, 2022.

[30] E. Borges, "Banner grabbing: Top tools and techniques explained." [Online]. Available: https://securitytrails.com/blog/banner-grabbing

[31] L. Huang, S. Wei, C. Gao, and N. Liu, "Cyclical adversarial attack pierces black-box deep neural networks," *Pattern Recognition*, 2022.

[32] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *SIGKDD'18*, 2018, pp. 2847–2856.

[33] D. Zügner, O. Borchert, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on graph neural networks: Perturbations and their patterns," *TKDD'20*, vol. 14, no. 5, pp. 1–31, 2020.

[34] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," *NeurIPS'20*, vol. 33, pp. 9263–9275, 2020.

[35] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *CVF'19*, 2019, pp. 588–597.

[36] M. Ahmed and I. Neamtiu, "Deanomalyzer: Improving determinism and consistency in anomaly detection implementations," in *AITest'23*, 2023.

[37] N. Tran, H. Chen, J. Bhuyan, and J. Ding, "Data curation and quality evaluation for machine learning-based cyber intrusion detection," *IEEE Access*, vol. 10, pp. 121 900–121 923, 2022.

[38] L. Perini, C. Galvin, and V. Vercruyssen, "A ranking stability measure for quantifying the robustness of anomaly detection methods," in *ECML-PKDD'20*. Springer, 2020, pp. 397–408.

[39] Y. Park, "Concise logarithmic loss function for robust training of anomaly detection model," 2023. [Online]. Available: https://arxiv.org/abs/2201.05748

[40] W. Gao, Y. Zhang, L. Liang, and Y. Xia, "Stability analysis for ranking algorithms," in *ICITIS'10*. IEEE, 2010, pp. 973–976.