Aggregating unsupervised provenance anomaly detectors

Ghita Berrada
School of Informatics
University of Edinburgh
gberrada@inf.ed.ac.uk

James Cheney
School of Informatics
University of Edinburgh and The Alan Turing Institute
jcheney@inf.ed.ac.uk

Abstract

System-level provenance offers great promise for improving security by facilitating the detection of attacks. Unsupervised anomaly detection techniques are necessary to defend against subtle or unpredictable attacks, such as *advanced persistent threats* (APTs). However, it is difficult to know in advance which views of the provenance graph will be most valuable as a basis for unsupervised anomaly detection on a given system. We present baseline anomaly detection results on the effectiveness of two existing algorithms on APT attack scenarios from four different operating systems, and identify simple score or rank aggregation techniques that are effective at aggregating anomaly scores and improving detection performance.

1 Introduction

Numerous provenance-tracking systems have been introduced with the aim of recording detailed information about activity at the operating system level. A natural and oftencited motivation for such systems is to support security goals, since, if the provenance records are sufficiently detailed and complete, then it should be possible to detect, or even interrupt, attacks on the monitored system. Recent work on systems such as SPADE [11], LPM [6], and CamFlow [22] has shown that it is possible to provide detailed, whole-system provenance efficiently and maintainably on top of commodity operating systems such as Linux using kernel modules. Likewise, the DARPA Transparent Computing program has supported the development of provenance recording systems on mainstream operating systems such as Windows, BSD, Linux, and Android, in order to facilitate detection of sophisticated attacks carried out as part of *Advanced Persistent Threat* (APT) campaigns [5, 24].

Unfortunately, simply recording enough information is only (at most) half of the problem. The resulting provenance records are often large and complex: it is not unusual for a single day's activity to result in several gigabytes of provenance data. Manually inspecting this data to look for suspicious behavior is not feasible, especially considering that APT activity may constitute only a fraction of a percent of the full system provenance record. Therefore, automated analysis of the data is imperative.

One natural (and common) approach [14] to find attacks is to define queries or patterns to use as "alarms", typically corresponding to behavior that attackers are known to perform that is very rare among benign activity. For example, it is suspicious if a process creates a file, executes it, then deletes it. Coming up with such alarms is a time-consuming and manual process, and does not guard against new or varying attacker behavior.

Another natural approach would be to apply supervised machine learning algorithms to learn what attacks look like from the data [19]. While this would potentially help decrease the manual effort required to come up with alarms, there are several obstacles that make it difficult to apply this approach. First, it requires developing a corpus of provenance data containing realistic attacks, together with annotations indicating which parts of the graph are part of an attack. This is time-consuming and labor-intensive. Secondly, since APT-style attacks are typically a very small fraction of the graph, there is a severe *class imbalance* problem. Finally, this approach presumes that the background activity (which we wish to distinguish from attack activity) is stable over time; if this is not the case, then an anomaly detector may lose effectiveness over time due to drift.

Because of these complications, we advocate an *unsupervised* approach in which we assume no access to training data. Indeed, a large number of unsupervised anomaly detection techniques have been studied, both for conventional datasets [1, 8] and for graphs [3]. Unfortunately, work on graph anomaly detection has mostly focused on relatively simple kinds of graphs, e.g. undirected or unlabeled graphs. Provenance graphs are directed, have node and edge labels, and typically also have properties on nodes and edges. Analyzing large provenance graphs to identify anomalies is computationally expensive.

We consider an alternative approach, in which we associate each object with one or more *views* of its behavior, which we call *contexts*(see section A for a brief definition and

TaPP 2019, June 3, 2019, Philadelphia, PA

examples). Each context consists of a set of related attributes or features, and we hypothesize that at least some attacks can be detected by examining these attributes. Moreover, the contexts are essentially records of conventional discrete or boolean features and so a variety of existing anomaly detection algorithms can be applied to them off-the-shelf [16, 23].

In this paper, we focus on contexts based on process activity in OS-level provenance graphs. Typical contexts include the events (i.e. types of system calls performed), process executable name or parent, and network activity. As we shall show, even simple anomaly detection algorithms can be surprisingly effective at finding attacks. However, no single context always provides the best detection performance: it depends to some extent on the type of attack, the operating system, and the background activity against which attacks are supposed to stand out. This is a big disadvantage for an unsupervised system, because the choice of which contexts are worth monitoring depends on information that may not be available in advance. While it is possible to combine all of the contexts into a single large one, and this does help, this is not feasible for approaches whose complexity grows with the number of attributes.

In this paper, we present baseline results of two published anomaly detection algorithms over different views of process activity in data using two scenarios from the DARPA Transparent Computing program. We then consider different ways of aggregating the scores or rankings obtained from different contexts. We present detailed experimental results to investigate which aggregation techniques work best with each of the two algorithms, and compare the results with the naive approach of fusing all the contexts together. Overall, we can conclude that across a wide variety of settings, the *geometric mean* provides a reliable and effective means of combining rankings, typically matching or exceeding the best individual scoring technique. Moreover, aggregation techniques frequently outperform the analysis of the whole dataset, when that analysis is possible.

2 Background

2.1 Anomaly detection

We will use two existing anomaly detection algorithms as baselines. These are not the only available unsupervised algorithms suitable for discrete data. However, the two algorithms we consider are simple and have relatively few parameters that need to be tuned. Aside from the high-level descriptions below, the rest of the paper does not depend on detailed understanding of either algorithm: we can think of both as black boxes which, given a sequence of objects $O = o_1, \ldots, o_n$ each with some Boolean attributes, assign each object a score indicating in some sense the degree of (dis)similarity to the dataset as a whole.

The *Attribute Value Frequency (AVF)* [16] algorithm is very simple: it first scans the data to obtain frequency counts (or equivalently, probability estimates) of each attribute value. In a second pass, each record is scored by adding together the frequencies/probabilities of the actual attribute values present in that record. In this way, records with many rare attributes tend to have a low score. Though this algorithm appears rather simple-minded, it is surprisingly effective on our data. In a previous study [7], we have found that it provides detection performance competitive with more sophisticated techniques such as OC3 (discussed below) while also being trivial to implement over streaming data (unlike OC3).

The *One Class Classification by Compression (OC3)* algorithm is, as the name suggests, based on attempting to compress the dataset (or at least, minimize an estimated compressed size). This approach is justified by the *Minimum Description Length principle* [12], which identifies the best explanation of some dataset as the hypothesis that enables the best compressed representation of it (or equivalently, the best predictions of unseen data). The underlying compression algorithm used by OC3 is Krimp, a system for identifying a subset of "interesting" attribute combinations in a dataset by defining a code table consisting of attribute combinations, and using this table to compress the dataset. As a simple example, if a dataset contains many co-occurrences of $a = 1, b = 1, c = 1$, then Krimp might create a code table entry for this combination, and use it to represent these three attributes whenever they co-occur. OC3 works as follows: first use Krimp to compress the dataset, then the anomaly score of each record is its estimated compressed size. Records that compress poorly (i.e. have high scores) are considered more anomalous: they have few patterns that the compressor can take advantage of to represent succinctly.

2.2 Score and rank aggregation

Because OC3 anomaly scores correspond to estimates of compressed size, there is a natural aggregation technique for them: simply add the scores obtained from different contexts. This is, of course, also possible for AVF, but, if we assign AVF scores of zero to objects not present in a context, then such objects will be considered much more anomalous than they should be (since zero is the most anomalous possible score). Nevertheless, we will consider this approach for both AVF and OC3 and see whether this intuition is borne out by experiments. We call this aggregation approach `sum`.

Objects can also be ranked based on their anomaly scores, generating a ranking list where the objects at the top are the most likely to exhibit suspicious activity (as object ranks increase, the likelihood of the corresponding process having a suspicious behavior decreases). *Rank aggregation* [17] is the problem of combining several rankings of a given set of objects into a single one that represents the different initial rankings well. We consider a ranking of objects $O = \{o_1, \ldots, o_n\}$ to be a function r from $O \to \mathbb{N}^+$ whose range is an initial segment of $\mathbb{N}^+$. Such a ranking can be

obtained from a scoring function $O \to \mathbb{R}$ by breaking ties arbitrarily, after sorting the objects in decreasing order of anomalousness. We assume that all rankings are total, by assigning unranked objects o the rank m of the highest ranked objects.

Rank aggregation is a well-studied problem, related to the problem of preferential voting. We will consider several approaches that are intuitive and easy to implement. In each case, rankings $r_1, \ldots, r_n$ can be aggregated by first "scoring" each object o using the following function:

$$s(o) = agg(r_1(o), \ldots, r_n(o))$$

and then sorting and reranking. This recipe can be instantiated in several ways by choosing different aggregation functions *agg*. We consider the following choices:

$$\begin{aligned}
\texttt{avg}(\vec{r}) &= \frac{1}{n}\sum_{i=1}^{n} r_i \\
\texttt{geom}(\vec{r}) &= \left(\prod_{i=1}^{n} r_i\right)^{\frac{1}{n}} = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \log r_i\right) \\
\texttt{min}(\vec{r}) &= r_i \text{ where } \forall j. r_i \le r_j \\
\texttt{median}(\vec{r}) &= \frac{1}{2}(r_{\lfloor n/2 \rfloor} + r_{\lceil n/2 \rceil})
\end{aligned}$$

Note that, because contexts don't necessarily have the same number of objects (for example, the number of objects in PN is most likely smaller than the number of objects in PE since processes having performed any type of event outnumber processes with network activity), objects that appear in some contexts and not others will have a blank ranking in contexts in which they don't appear (for example, a process with no network activity will have a defined ranking in PE and a blank ranking in PN).

3 Source data

We consider two datasets derived from the DARPA Transparent Computing program. They result from two exercises to evaluate provenance recorders and analysis techniques. The characteristics of the datasets are summarized in Tables 4a and 4b in the Appendix. The first scenario corresponds to around 5 days of data, and includes provenance graphs recorded on four different operating systems, each of which was subject to (part of) an APT-style campaign. The second scenario corresponds to around 8 days of data, under similar conditions to scenario 1. (In scenario 2, a second Linux-based recording system was also used, but we have omitted it since the data resulting from this system in scenario 1 had too many inconsistencies for our data cleaning to work.) There are numerous differences between scenario 1 and scenario 2: the workload is higher, the attacks are more sophisticated, and the provenance graphs are much larger. Also, the underlying recording systems were under active development between the two scenarios.

As described in a previous paper [7], the ADAPT project provides a system that ingests this data into a Neo4j database in a uniform format, and performs duplicate elimination and some other data cleaning. From this ingested data, we extract several contexts. This system, and the contexts and extraction process is described in greater detail elsewhere. In this paper, our starting point is the extracted data, in the form of CSV files that relate unique identifiers of processes with descriptions of different aspects of process activity. The contexts are:

- PE: which relates a process with different events (classes of system calls) such as read, write, etc.
- PX: which relates a process with the name(s) of its executable.
- PP: which relates a process with the name(s) of its parent's executable.
- PN: which relates a process with IP addresses and ports accessed.
- PA: the result of joining all of the above contexts together, using the process unique IDs as keys.

We have ingested all of these datasets into a Neo4j database and extracted the above contexts (in each case, this is possible using a simple Cypher query). The ingestion process for the larger, second scenario took over a week.

Both scenarios also come with high-level, human-readable descriptions of the attacks performed. As part of previous work, machine-readable ground truth annotations (i.e. lists of unique identifiers of processes and other components of attacks) were constructed manually [7] for scenario 1. We constructed similar ground truth annotations for scenario 2.

The raw data for scenario 1 is not publicly available. The data for scenario 2 is available at https://github.com/darpa-i2o/Transparent-Computing. Our contexts and ground truth data for both scenarios are available upon request (we plan to make it publicly available soon).

3.1 An example

We now consider some actual scores and rankings resulting from the experiments reported later in the paper, to help make the overall picture more concrete.

We consider the Windows PE dataset for scenario 1, which contains 8 attack processes. The rankings of these processes by OC3 are as follows:

attack#	PE	PX	PP	PN	PA
a_1	242	187		1	490
a_2	192	5821	107	2	1
a_3	73	119	164	85	50
a_4	74	120	165	91	53
a_5	75	121	166	92	52
a_6	244	36	11	83	513
a_7	27	172		72	510
a_8	160	168		77	491

As we can see, some contexts identify different parts of the attack at position 1 or 2, which is ideal, but others are ranked much lower. It seems that, to identify all of the attack components, we would have to inspect at least the first 80 processes in each basic context. If, instead, we just considered PA we would also have to inspect the first 500 or so processes in order to see all of the attacks, and they are relatively sparsely distributed; one attack is at position 1, but several are ranked around position 500.

Now, if we consider the rankings obtained by the different aggregation approaches considered so far, they are as follows.

attack#	sum	avg	geom	median	min
a_1	1	1352	122	261	2
a_2	2	929	77	145	8
a_3	18	11	42	30	268
a_4	21	13	51	44	270
a_5	22	14	54	48	277
a_6	27	3	3	2	41
a_7	31	1322	231	96	99
a_8	115	1348	540	165	282

These rankings are generally somewhat improved over the basic contexts or even the combined context PA. The best of them, using `sum` aggregation, places almost all attacks in the top 31 and the final one at position 115. The `avg` approach, which averages the rankings (not scores), gives much lower rankings to attacks a_1, a_7 and a_8, perhaps because these attacks have no ranking in the PP context, so they are penalized with a very low ranking.

4 Experimental results

Evaluation metrics We consider two metrics for the performance of anomaly detectors, the *area under ROC curve* (AUC) [1] and the *normalized discounted cumulative gain* (NDCG) [15]. The *receiver–operator characteristic* (ROC) curve plots the percentage of true positives (e.g. attacks) detected vs. the percentage of false positives encountered. The area under this curve is an estimate of the probability that a randomly-chosen anomaly is ranked higher than a randomly-chosen normal record. Thus, higher is better; the AUC score ranges from 0 to 1 (a perfect score).

$$AUC(d_{ij}) = \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \frac{d_{ij}}{N_a N_b}$$

where N_b is the number of benign records, N_a is the number of attacks/anomalies, and d_{ij} is 1 when attack i is before benign element j in the ranking, and 0 otherwise. (This matrix is easily obtained from the ranking r; there are more efficient ways to calculate AUC.)

The AUC score is widespread and standard for measuring the performance of anomaly detectors [2]. However, for large datasets with very sparse anomalies, it does not necessarily conform to intuition. For example, the AUC score of PE is over 0.99 in the example of the previous section, even though its top-ranked attack is at position 27 while PP has instead a top-ranked attack at position 11 but an AUC score of around 0.62, because it misses three attacks entirely. Thus, AUC is, in some sense, a measure of the worst-case behavior: ranking a single attack poorly (or missing attacks) can have a disproportionate impact compared to giving mediocre rankings to all attacks.

The NDCG score [15] is an alternative measure of the degree to which a ranking matches the ideal (where all of the anomalies are ranked at the top). It is frequently used in information retrieval settings, where it is critical that relevant results are found near the top of the ranking: for example, it is well known that users seldom click past the first page of search engine results, and advertisers are willing to pay good money to be listed as the first result. The unnormalized DCG score is defined as follows:

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)}$$

where n is the number of ranked elements and rel_i ("the relevance of item at rank i") is 0 if the item at rank i is benign and 1 if the item is an attack. Like the AUC score, NDCG is between 0 and 1 (with 1 being best), but places greater weight on rankings close to the top, by taking the logarithm of the ranking. The NDCG score is obtained by dividing the DCG of a given ranking with the maximum achievable DCG, i.e. that obtained by placing all of the attacks at the top of the ranking. (In general, DCG and NDCG can also take the "relevance" of different results into account; we consider all attacks to have an equal relevance score of 1.) Among detectors with very high AUC scores, NDCG scores are helpful for comparing the quality of the ranking.

We do not report running time of either the base contexts or aggregation techniques: the former are evaluated in prior work on AVF and OC3 respectively, and the latter are implemented using naive, but reasonably efficient, Python scripts. The performance of rank aggregation can probably be improved, but is not the main issue here (they all run in linear or $O(n \log n)$ time).

Results on basic contexts We ran AVF and OC3 on each of the basic contexts resulting from the two scenarios. The results are shown on the left-hand side of Tables 1a,1b,2a and 2b. Each table shows the AUC score and NDCG score of each context. The best score obtained among basic contexts is shown in boldface.

Inspecting the AUC and NDCG scores, we can observe some trends. AUC scores are typically higher than NDCG scores, but, from manual inspection, we find that NDCG scores of 0.4 or higher often correspond to usable results, e.g. ranking at least one attack process in the top 10. Almost all of the maximum scores in scenario 1 are in this range.

Table 1. AUC and nDCG score results for Scenario 1

(a) AVF

Source	Metric	PE	PX	PP	PN	PA	sum	avg	geom	median	min
Windows	AUC	0.968453	0.951961	0.620603	**0.998662**	0.995632	0.001999	0.938251	**0.998463**	0.998001	0.996307
	nDCG	**0.603553**	0.280136	0.210894	0.582564	0.526925	0.143535	0.291811	**0.665215**	0.589899	0.481892
BSD	AUC	0.8758	**0.882951**	0.794625	0.230754	0.983815	0.747818	0.910939	**0.988885**	0.928895	0.8283
	nDCG	**0.513598**	0.346095	0.309749	0.264745	0.524454	0.221113	0.412501	0.63857	**0.736187**	0.43183
Linux	AUC	0.823249	**0.831733**	0.745457	0.799258	0.926913	0.119294	0.910763	0.981438	0.930732	**0.997099**
	nDCG	0.272106	**0.437784**	0.206896	0.318006	0.297636	0.17425	0.260854	0.429756	**0.470940**	0.383228
Android	AUC	**0.826797**	0.570806	0	0.519608	**0.830065**	0.359477	0.712418	0.75817	0.742919	0.723312
	nDCG	**0.849505**	0.390515	0	0.475447	**0.834072**	0.384828	0.446402	0.56294	0.636634	0.602571

(b) OC3

Source	Metric	PE	PX	PP	PN	PA	sum	avg	geom	median	min
Windows	AUC	0.992053	0.951961	0.620603	**0.997296**	0.997396	**0.998975**	0.964583	0.994927	0.994785	0.99317
	nDCG	0.302183	0.280136	0.210894	**0.709458**	0.643458	**0.756333**	0.443902	0.397767	0.421411	0.569777
BSD	AUC	0.97636	**0.998894**	0.845847	0.230732	0.995862	0.99911	0.949692	0.999638	**0.999657**	0.999203
	nDCG	0.435758	**0.464937**	0.43545	0.341885	0.704075	**0.755445**	0.659702	0.677094	0.606847	0.543351
Linux	AUC	**0.887883**	0.831151	0.811842	0.799654	0.995978	0.997208	0.943724	0.996904	0.978369	**0.998772**
	nDCG	0.385061	0.310158	0.244903	**0.498119**	0.463614	**0.554706**	0.26248	0.405506	0.367585	0.476284
Android	AUC	**0.753813**	0.570806	0.0	0.529412	**0.795207**	**0.795207**	0.712418	0.735294	0.705882	0.569717
	nDCG	**0.740326**	0.390515	0.0	0.665942	0.6823	**0.804621**	0.456948	0.574719	0.772502	0.685128

Table 2. AUC and nDCG score results for Scenario 2

(a) AVF

Source	Metric	PE	PX	PP	PN	PA	sum	avg	geom	median	min
Windows	AUC	0.8078	0.908927	**0.940364**	0.443169	DNF	0.454578	0.967885	**0.975070**	0.968712	0.879234
	nDCG	0.23125	0.247556	0.232215	**0.288379**	DNF	0.1834	0.32027	**0.335787**	0.296208	0.302284
BSD	AUC	0.874365	**0.917880**	0.567909	0.89003	DNF	0.235826	0.884352	**0.998099**	0.973508	0.991859
	nDCG	0.197063	0.176319	0.173573	**0.340189**	DNF	0.13224	0.384053	**0.464222**	0.428045	0.294173
Linux	AUC	0.796777	**0.950376**	0.808792	0.780768	DNF	0.138888	0.935786	0.991597	0.938141	**0.993588**
	nDCG	0.296173	0.361843	0.253723	**0.428993**	DNF	0.214308	0.384563	0.390217	0.306841	**0.460792**
Android	AUC	0.901339	**0.978568**	0.0	0.765571	0.922308	0.410756	0.950501	**0.978853**	0.964843	0.567684
	nDCG	0.308236	**0.388954**	0.0	0.326172	0.356021	0.363142	0.361115	**0.440106**	0.388108	0.271071

(b) OC3

Source	Metric	PE	PX	PP	PN	PA	sum	avg	geom	median	min
Windows	AUC	0.856675	0.908927	**0.940364**	0.443468	DNF	0.969838	0.966789	**0.975697**	0.959057	0.949481
	nDCG	0.242088	0.247556	0.232215	**0.365061**	DNF	**0.429694**	0.313282	0.34133	0.282137	0.347435
BSD	AUC	0.935994	**0.999000**	0.66811	0.891853	DNF	0.999573	0.929166	**0.999697**	0.995166	0.999557
	nDCG	0.248715	0.484461	0.243905	**0.514896**	DNF	0.60926	0.419966	**0.653287**	0.583637	0.441898
Linux	AUC	0.873358	**0.944820**	0.933795	0.7814	DNF	**0.999113**	0.958661	0.998994	0.978573	0.996368
	nDCG	0.387535	0.422006	**0.428522**	0.354682	DNF	0.545132	**0.721963**	0.629404	0.674969	0.427674
Android	AUC	0.883986	**0.978650**	0.0	0.687021	0.982278	0.98522	0.951937	**0.990545**	0.987127	0.706331
	nDCG	0.327773	**0.392018**	0.0	0.300675	0.405909	0.397795	**0.606754**	0.451457	0.437549	0.278839

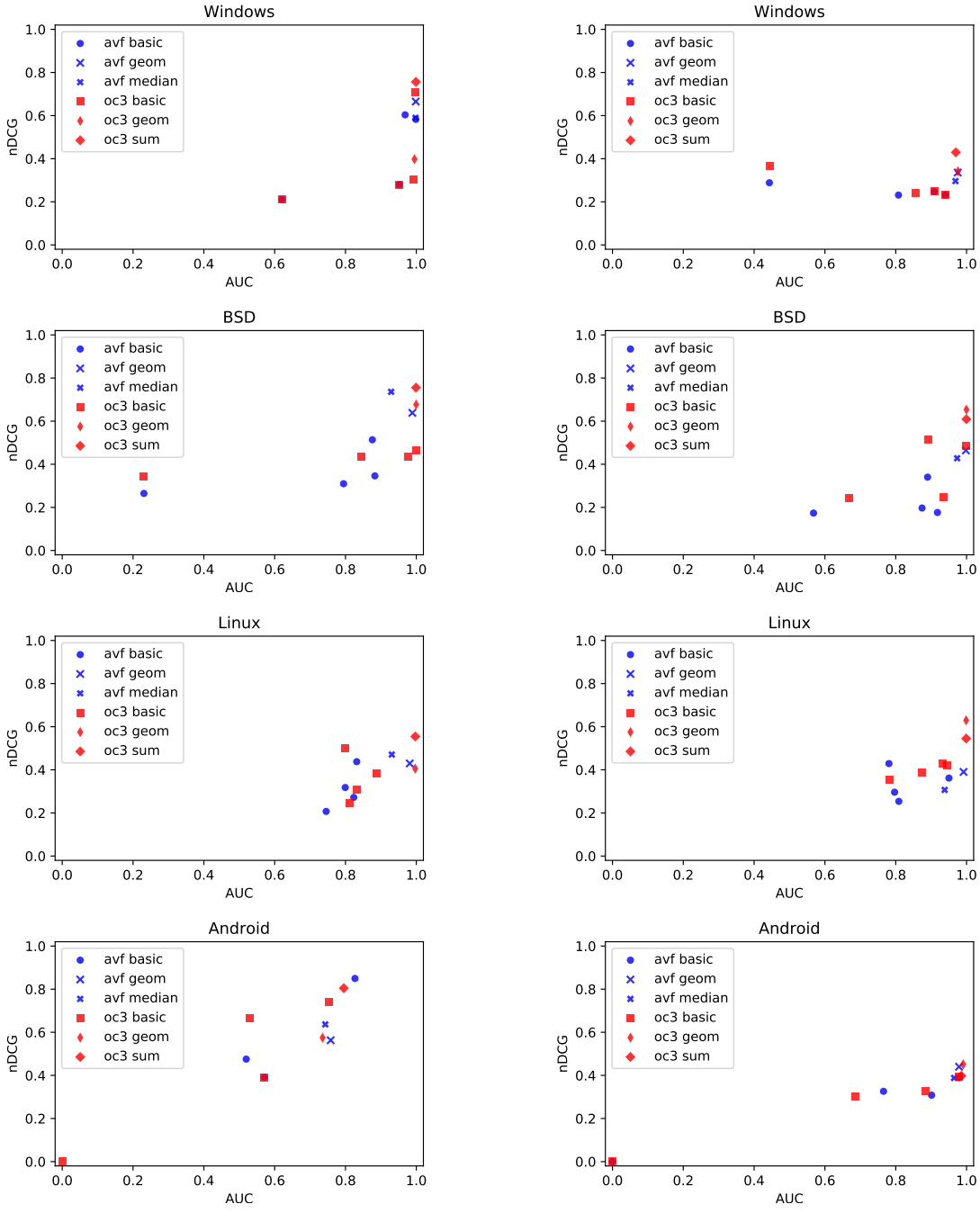
Thus, these results suggest that AVF and OC3 are both able to identify useful information for anomaly detection.

We can also see that there are differences among the scenarios and among the data sources. Scenario 1 seems to be "easier", in that the AUC and NDCG metrics are typically higher than for scenario 2. Likewise, detection performance on the BSD data seems best overall. The detection performance of OC3 also seems somewhat better overall than AVF, which might be expected given that AVF is much simpler.

For scenario 1, the PE context is the most effective overall, but, in eight cases, another context such as PX or PN is better. For scenario 2, the picture is different: while PE is often

Figure 1. AUC vs. NDCG scores for base contexts and selected aggregates

(a) Scenario 1 **(b)** Scenario 2



among the most effective contexts for anomaly detection, it was never the winner; the overall best context was PX, with PP and PN being best 3–5 times each. These results confirm our motivating hypothesis that it is difficult to predict in advance which of several contexts will be most valuable for anomaly detection.

Results of aggregation Next, we ran all of the aggregation algorithms discussed in Section 2 on the results obtained from the basic contexts. We also show the result of PA for all scenario 1 contexts, considering it as an aggregation techniques; however, running AVF or OC3 on PA in scenario 2 is prohibitively expensive for all but the smallest (Android)

settings. In each case, we calculate AUC and NDCG scores in the same way as before. The results are shown on the right-hand sides of Tables 1a,1b,2a and 2b. Again, the best score obtained from aggregation is shown in boldface.

The results seem to exhibit general trends consistent with the results on base contexts: for example, scenario 2 still seems more challenging than scenario 1, OC3 still seems to outperform AVF, etc. Considering all of the results, the best overall results appear to be obtained by `geom` and `median` (for AVF) and by `geom` and `sum` (for OC3). As we expected, `sum` does poorly when combining AVF-based scores, since adding the scores does not have a natural interpretation.

The results for PA are of some interest. Generally, PA usually does a good job of improving on the basic contexts: while it is not always better than all of them, it is usually competitive. However, it is seldom the best choice among the aggregation techniques. For scenario 1, PA does consistently worse than the other aggregations except for the Android data, where it performs best. For scenario 2, we were only able to score PA for the Android data, but its performance was not competitive with other aggregation techniques there.

To get better insight into how these aggregation techniques compare to the detection performance over base contexts (and to each other), we have plotted the AUC scores and NDCG scores in Figure 1. In each plot, the x axis corresponds to the AUC score and the y axis to the NDCG score. The "avf basic" and "oc3 basic" series show the results of the different basic contexts, and we plot `geom` for both detectors, `median` for AVF, and `sum` for OC3. Since ideal performance corresponds to the point (1,1), the best performance corresponds to the upper-right corner of each plot.

In most cases, we can see that the best aggregation performance for each detector significantly improves upon the best results from basic contexts: at least one aggregate point and usually several are near the upper right corner. Indeed, in scenario 1, in the BSD and Linux results, all four of the aggregates are clustered in the upper-right corner. The results are more mixed for the Windows and Android data, however. Moreover, the best overall performance is obtained with `sum` and OC3. AVF aggregated using `geom` or `median` also give good results but neither dominates the other.

For scenario 2, the results differ somewhat, perhaps because this scenario is larger and more realistic. The maximum NDCG scores tend to be lower, indicating that anomaly detection may be more challenging in this scenario. Interestingly, `geom` seems to be more robust for both AVF and OC3 in this scenario: for OC3, it is the best overall in three of four scenarios, while, for AVF, `median` is always worse than `geom`.

AUC versus nDCG In Tables 1 and 2, several configurations (OS, algorithm, context/aggregator) have high AUC but low nDCG scores. This is easily explained by the fact that while AUC simply measures "classification" performance i.e. attack detection performance, nDCG adds an additional constraint, which is whether the attacks detected are close or not to the top of the rankings, and penalizes the detected attacks proportionally to their position in the rankings (i.e. according to how useless to the user they are).

5 Related work

Anomaly detection AVF and OC3 are just two among many algorithms; we have focused on them due to their accessibility (AVF is trivial to implement and source code for Krimp and OC3 is publicly available) and good baseline performance. We have conducted informal experiments with other approaches such as Frequent Pattern Outlier Factor (FPOF) [13], Outlier Degree (OD) [20], and CompreX [4]. FPOF and OD both require parameter tuning and their detection performance on our data is significantly worse than AVF or OC3. CompreX is, like OC3, based on compression and the MDL principle, and compared favorably with OC3 in reported experiments. We obtained good detection performance using CompreX on scenario 1 PE data [7]. However, CompreX is computationally expensive, and the running time grows quickly as the number of attributes grows. Nevertheless, it is worthwhile to see whether aggregation approaches are effective for these or other suitable anomaly detection techniques.

Attack and anomaly detection over provenance has been studied recently. StreamSpot [19] and Sleuth [14] highlight suspicious subgraphs but rely on training data or domain knowledge. Winnower [25] applies graph grammars to summarize provenance graphs, and is fast and effective but assumes a scenario in which there are many similar "clean" provenance graphs derived from machines with similar workloads in a data center. In contrast, our approach requires no parameter tuning, ground truth, or clean data, but so far highlights only suspicious process nodes, not subgraphs, making a direct comparison nontrivial. (We do use ground truth annotations to evaluate our approach, but not to train it.)

Aggregation and feature selection We have considered several simple, parameter-free, and easy-to-implement, approaches to rank aggregation, drawing on Lin's survey [17]. There are a number of other approaches that could be considered instead, such as a Markov chain-based algorithm [9] and Cross-Entropy Monte Carlo [18]. However, these approaches are more computationally intensive, and good performance relies on parameters that may be difficult to determine in advance. Feature selection may also be applicable, but there is little work on unsupervised feature selection for anomaly detection; one recent proposal is [21].

6 Conclusions and future work

In this paper, building on [7], we investigated the results of combining existing techniques for anomaly detection and

rank aggregation on relatively new datasets that attempt to capture realistic APT-style attack scenarios and workloads.

Our experimental results support several conclusions. First, the baseline results further confirm that off-the-shelf algorithms such as AVF or OC3 can be used to detect attacks based on contexts that reflect a small subset of the available provenance data, rather than by an expensive investigation of graph structure. Compared to our previous work [7], which only considers scenario 1, we consider the much larger and more realistic scenario 2. Second, the baseline results also indicate that it may be difficult to predict in advance which contexts are most useful for anomaly detection. Instead, however, simple aggregation techniques such as summing scores, or taking the geometric mean or median of rankings, are quite effective, often producing improvements in both AUC and NDCG scores simultaneously over the results from the base contexts.

Although it is sometimes possible to obtain good overall results by joining all of the contexts together, this can be prohibitively expensive. Moreover, our results show that this usually does not produce the best detection results compared to other forms of aggregation. The main exception to this rule is that for smaller datasets, such as the Android data in scenario 1, the PA approach was the best. The overall success of other aggregation techniques over PA is intriguing because it suggests that our manual partitioning of the available attributes into contexts may yield a detection boost similar to ensemble learning techniques; thus, it may be worthwhile to try out outlier ensemble techniques, such as bagging, as well [2]. (Boosting techniques [10] typically assume a supervised setting making them less useful for unsupervised anomaly detection.)

Like most work in this area, we cannot conclude that our results are guaranteed to work in other, unknown scenarios generalizability of this approach could be further investigated subject to availability of more data or by considering other anomaly detection algorithms. Despite this, overall our results contribute to improved understanding of how to detect attacks and anomalies in provenance using unsupervised anomaly detectors and aggregation techniques to combine contexts. These results provide a foundation for future work on unsupervised identification of suspicious subgraphs for human investigation, complementing existing methods [14, 19, 25].

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under contract FA8650-15-C-7557.

References

[1] C. C. Aggarwal. *Outlier Analysis*. Springer, 2017.
[2] C. C. Aggarwal and S. Sathe. *Outlier Ensembles*. Springer, 2017.
[3] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data Min. Knowl. Discov.*, 29(3):626–688, 2015.
[4] L. Akoglu, H. Tong, J. Vreeken, and C. Faloutsos. CompreX: Fast and reliable anomaly detection in categorical data. In *CIKM*, pages 415–424, 2013.
[5] M. Auty. Anatomy of an advanced persistent threat. *Network Security*, 2015(4):13–16, 2015.
[6] A. M. Bates, D. Tian, K. R. B. Butler, and T. Moyer. Trustworthy whole-system provenance for the Linux kernel. In *USENIX Security 2015*, pages 319–334, 2015.
[7] G. Berrada, S. Benabderrahmane, J. Cheney, W. Maxwell, H. Mookherjee, A. Theriault, and R. Wright. Unsupervised, streaming advanced persistent threat detection in system-level provenance. Draft available at http://homepages.inf.ed.ac.uk/jcheney/publications/drafts/avf-stream.pdf.
[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.
[9] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622. ACM, 2001.
[10] Y. Freund, R. D. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.
[11] A. Gehani and D. Tariq. SPADE: support for provenance auditing in distributed environments. In *Middleware*, pages 101–120, 2012.
[12] P. Grünwald. *The Minimum description length principle*. MIT Press, 2007.
[13] Z. He, X. Xu, J. Z. Huang, and S. Deng. A frequent pattern discovery method for outlier detection. In *WAIM*, pages 726–732. Springer, 2004.
[14] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. N. Venkatakrishnan. SLEUTH: real-time attack scenario reconstruction from COTS audit data. In *USENIX Security 2017*, pages 487–504, 2017.
[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.
[16] A. Koufakou, E. G. Ortiz, M. Georgiopoulos, G. C. Anagnostopoulos, and K. M. Reynolds. A scalable and efficient outlier detection strategy for categorical data. In *ICTAI 2007*, pages 210–217, 2007.
[17] S. Lin. Rank aggregation methods. *WIREs Computational Statistics*, 2(5):555–570, 2010.
[18] S. Lin and J. Ding. Integration of ranked lists via cross entropy monte carlo with applications to mrna and microrna studies. *Biometrics*, 65:9–18, 2009.
[19] E. A. Manzoor, S. M. Milajerdi, and L. Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *KDD*, pages 1035–1044, 2016.
[20] K. Narita and H. Kitagawa. Outlier detection for transaction databases using association rules. In *WAIM*, pages 373–380, 2008.
[21] G. Pang, L. Cao, L. Chen, and H. Liu. Unsupervised feature selection for outlier detection by modelling hierarchical value-feature couplings. In *ICDM*, pages 410–419, 2016.
[22] T. F. J. Pasquier, X. Han, M. Goldstein, T. Moyer, D. M. Eyers, M. Seltzer, and J. Bacon. Practical whole-system provenance capture. In *SoCC 2017*, 2017.
[23] K. Smets and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *SDM 2011*, pages 804–815, 2011.
[24] D. Smith. Life's certainties: death, taxes and APTs. *Network Security*, 2013(2):19–20, 2013.
[25] W. Ul Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *NDSS*, 2018.

A Contexts

Contexts are Boolean-valued datasets/matrices extracted from provenance graphs and represent an aspect of process behavior. Table 3 is an example of such a context. Its rows (also called objects) are process identifiers and its columns (called attributes) are types of system events. For each row, a value '1' means that the process (represented by the row) has performed at least one event of the type corresponding to the column and '0' otherwise; the exact number of such events is ignored. For example, in the example shown in Table 1, the process with identifier d273e9c1-372b-3e7c-8338-59bc2be6b01c has performed the following events: EVENT_FORK, EVENT_CLOSE and EVENT_EXECUTE. Other commonly used contexts are described in Section 3. Their characteristics are summarized in Tables 4a and 4b.

Table 3. Example of context: process identifiers vs type of system events (extracted from Linux provenance graph)

Object_ID	EVENT_TRUNCATE	EVENT_MODIFY_FILE_ATTRIBUTES	EVENT_UNLINK	EVENT_ACCEPT	EVENT_OPEN	EVENT_CREATE_OBJECT	EVENT_MPROTECT	EVENT_UPDATE	EVENT_FORK	EVENT_CLONE	EVENT_MMAP	EVENT_EXECUTE	EVENT_LOADLIBRARY	EVENT_CHANGE_PRINCIPAL	EVENT_CONNECT	EVENT_LINK	EVENT_EXIT	EVENT_UNIT	EVENT_WRITE	EVENT_SENDMSG	EVENT_RENAME	EVENT_RECVMSG	EVENT_READ	EVENT_CLOSE
9336bbbb-23b2-367e-b768-be33e5f130c8	0	0	0	0	1	1	1	0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	1	1
986128dc-ad93-3d7c-99e5-83fa71c766dd	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
d273e9c1-372b-3e7c-8338-59bc2be6b01c	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
a871728c-a30b-307b-ae70-ea67d8f8de8b	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	1	1
e492b0e0-eec3-31ee-87c8-255e71ff5d86	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Table 4. Summary of datasets. In each context column, the upper element of a cell shows the number of rows (records) and the lower element the number of columns (attributes).

(a) Scenario 1

	Size	PE	PX	PP	PN	attacks
Windows	743 MB	17569 22	17552 215	14007 77	92 13963	8
BSD	288 MB	76903 29	76698 107	76455 24	31 136	13
Linux	2.86 GB	247160 24	186726 154	173211 40	3125 81	25
Android	2.69 GB	102 21	102 42	0 0	8 17	9

(b) Scenario 2

	Size	PE	PX	PP	PN	attacks
Windows	9.53 GB	11151 30	11077 388	10992 84	329 125	8
BSD	1.27 GB	224624 31	224146 135	223780 37	42888 62	11
Linux	25.9 GB	282087 25	271088 140	263730 45	6580 6225	46
Android	10.9 GB	12106 27	12106 44	24 11	4550 213	13