Learning to Detect Network Intrusion from a Few Labeled Events and Background Traffic

Gustav Šourek, Ondřej Kuželka, Filip Železný

Learning to detect network intrusion from a few labeled events and background traffic

Gustav Šourek[1], Ondřej Kuželka[2], and Filip Železný[1]

[1] CTU Prague, Czech Republic
{souregus,zelezny}@fel.cvut.cz
[2] Cardiff University, UK
kuzelkao@cardiff.ac.uk

Abstract. Intrusion detection systems (IDS) analyse network traffic data with the goal to reveal malicious activities and incidents. A general problem with learning within this domain is a lack of relevant ground truth data, i.e. real attacks, capturing malicious behaviors in their full variety. Most of existing solutions thus, up to a certain level, rely on rules designed by network domain experts. Although there are advantages to the use of rules, they lack the basic ability of adapting to traffic data. As a result, we propose an ensemble tree bagging classifier, capable of learning from an extremely small number of true attack representatives, and demonstrate that, incorporating a general background traffic, we are able to generalize from those few representatives to achieve competitive results to the expert designed rules used in existing IDS Camnep.

Keywords: Intrusion Detection, Random Forests, NetFlow, Camnep

1 Introduction

Intrusion detection systems analyse network traffic data with the goal to reveal malicious activities and incidents. In this paper we refer to an existing solution for intrusion detection, a multistage collective network behavior analysis system called Camnep [22]. The strategy of Camnep is to monitor high volume traffic networks for incidents, based on statistical information aggregated from publicly accessible parts of connections, i.e network flows or *NetFlows* (Section 3), utilizing variety of techniques from rules to statistical modeling. To assess maliciousness of incidents the system needs to extract higher-level information from lower-level data by constructing so called *events* from the individual NetFlows. Aggregating NetFlows to meaningful entities is an open problem and existing solutions rely mostly on standards and handmade rules designed by domain experts.

The goal of this paper is to provide adaptive machine learning model, capable to generalize from an extremely small number of available true attack representatives, with accuracy close to the expert designed process presented in Camnep. To that aim, we first introduce a fast scalable heuristic procedure for the extraction of generic events from NetFlow traffic (Section 4). Second, we propose an

enhanced Random-Forest-based learning model (Section 5) utilizing the small number of available ground truth samples of particular incident types, with the help of a large number of samples generated from background traffic by the heuristic procedure. The performance of the learned model to identify intrusions is evaluated against Camnep on the same traffic data, and a correspondence of the two methods is analyzed (Section 6).

2 Related work

The amount of network generated data and progress in the area of machine learning suggest for development of automatic and adaptive solutions to address the intrusion detection problem, and there has been a wide variety of machine learning approaches proposed to tackle IDS issues [28]. Related works utilize decision trees to learn explicit knowledge [15], and unsupervised methods using clustering as an inherent part of network traffic analysis [16]. In some latest works, even distributed, robust approaches utilizing strategies from trust modeling and game theory were introduced [2].

The most relevant works, assuming the nature of our dataset, include one-class anomaly detection methods for IDS [20], and semi-supervised approaches for traffic classification [9]. From the learning point of view, the most related works include utilizing Random Forests algorithm to build intrusion patterns [8] or to detect anomalies through the random forest outlier measure [29].

However, in real world practice, there are numerous issues affiliated with an online use of machine learning models on actual networks, and majority of the IDS currently in use are still either rule, or expert system based [18], providing the advantage of interpretability. A somehow hybrid approach is presented in Camnep [22], which combines expert rules for information and feature extraction on the lower stages, with data mining, classification, agent techniques and trust modeling in the final stages [23].

A number of the related works refer to a popular dataset created in 1999 named KDD'99 [19], published by Defence Advanced Research Projects Agency (DARPA). Although it was a step forward in comparing and evaluating IDS approaches, this work has received a lot of critique for including flaws in many statistical respects [27], limitations that are inherited from the DARPA datasets [17], and is widely considered outdated. While problems with the KDD'99 dataset are well-known, in this paper we refer to a real-use IDS Camnep [22], processing recent university traffic, instead of KDD'99, in a widely used NetFlow format to guide and evaluate ourselves, so that we can work with actual incidents and provide feedback relevant to the system.

In contrast with most previous works on detection we take a rather different approach than just classifying preprocessed standalone connections with features, that are not always present in practice, as we address the problem on an event layer, built on top of the NetFlow records. The main contribution of this work is then the joint approach to the detection problem, aggregating flows into events and leveraging the potentially unlimited source of useful background

samples (Section 4) for the respectively customized tree ensemble learning model (Section 5.1). We finally evaluate our model against real-use IDS [22], while utilizing the NetFlow standard format, collected from in-line university network probes, all in actual development with Cisco research.

3 Traffic data

The traffic data we work with were collected from local university (CTU) network during a period of one week. In its raw form the data consist of elementary information aggregated from network packets, which we refer to in this paper as NetFlows. It is a unidirectional component of TCP (UDP, ICMP equivalent) identified by shared source and destination addresses and ports, together with the protocol, captured within the frame of activity defined by a timeout mechanism. The NetFlow format was introduced on Cisco routers to give the ability to collect IP network traffic as it enters or exits an interface, and it is a tuple of:

Start, Duration, Protocol, src-IP, src-Port, dest-IP, dest-Port, Flags, #Packets, #Bytes

This format is widely adopted for security event logging and a number of affiliated disciplines[12, 24]. As a comprehensible unit of network communication with a number of high level features, many IDS works built their detection capabilities on the NetFlow level [26], but generally the incidents and attacks may consist of a multitude of NetFlows. Such sets of flows are commonly called *events*. In addition to information carried by individual flows, events display more complex aggregated properties that cannot be perceived on the individual flow level [2].

Example 1. For a simplistic instance consider a sequence of flows aiming at a particular endpoint port 22. If each of the flows is to be analyzed separately, it can easily be considered as a regular ssh communication request. Yet when we group those flows, according to their properties, such as the destination endpoint, we can explore potential malicious ssh-cracking behavior from the distributed plurality of small similar flows checking the same ssh endpoint during a short time-scope, possibly transferring a considerable amount of data back afterwards.

In this paper, we work with two sources of available data: *ground truth events* and *background traffic flows*. Ground truth events represent attacks evaluated and confirmed by a domain expert and constitute our only source of positive training examples. They are quite scarce as there are only dozens of ground truth events collected. The particular samples of attacks in our ground-truth events include various types of port-scan behavior and samples of ssh-cracking. We are aware that the particularities of these two types of attacks introduce bias into the model, making it possibly hard to generalize over different types of attacks, especially those with more complex network behavior. Nevertheless, for the sake of clarity we refer to these samples in the rest of the paper as generally the *malicious events* or *attacks*.

Background traffic flows is then a large collection of all flows from a snapshot of the local university network traffic. Events occurring within this collection are not available and yet need to be determined (Section 4). Also the true nature of the underlying background traffic is unknown, but believed to be generally legit. Most of the events probably correspond to harmless activities but some of them may also correspond to attacks. Since, in the end, we will need to evaluate how accurately our learned models are able to detect attacks and, more specifically, how close they can get to the performance of particular Camnep rules, we will proceed as follows. In the training phase, we will always work only with the ground truth events and with the unlabeled background traffic but in the testing phase, we will label the background traffic using Camnep and evaluate the accuracy of the learned models using its output labeled test-set events. In the rest of the paper, we will refer to the background traffic events labeled as malicious by Camnep as the *system samples*.

Naturally, the use of labels obtained using Camnep as a proxy for the true labeling has its drawbacks, particularly that a model which could detect the same set of malicious events as Camnep perfectly and on top of that also some missed malicious events would be considered worse than a model which would just perfectly mimic Camnep. However, there is no simple remedy for this problem except a laborious confirmation of the detected discrepancies by a domain expert, which is costly and time-consuming and which we therefore did not perform for this initial study (but which we plan to do in the future).

4 Event extraction

As we mentioned, the extraction of events from background flows is a crucial step in the detection of malicious behavior. Unfortunately, there is no general prescription of how an event should be formed based on the properties of the underlying flow set, and common approaches (including Camnep) thus rely on some form of flow clustering [5]. Whatever the underlying events look like, we need to miss as few malicious samples as possible, and thus we generally want the event extraction procedure not to be picky about what constitutes an event. For this reason we restrain from using standard clustering algorithms, employing pair-wise similarity metric defined on the flow level, as they would introduce a bias, while assigning each flow into a single cluster, possibly causing some of the attacks to be missed. In other words, in this stage we want to generally maximize recall of events and leave the burden of selecting the proper malicious ones on classifiers trained to do so based on past data. On the other hand, we also cannot extract all subsets of background flows as events because there would be an astronomical number of them. Therefore, the way the background event samples are constructed comes from generic rules, capturing only very general constraints on what is and what is not an event. Most of these rules can be understood as a parallel to the basic NetFlow/IPFIX aggregation features [7], but they operate on top of the Netflow level, creating more complex aggregates with considerable amount of redundancy for the sake of proper event recall

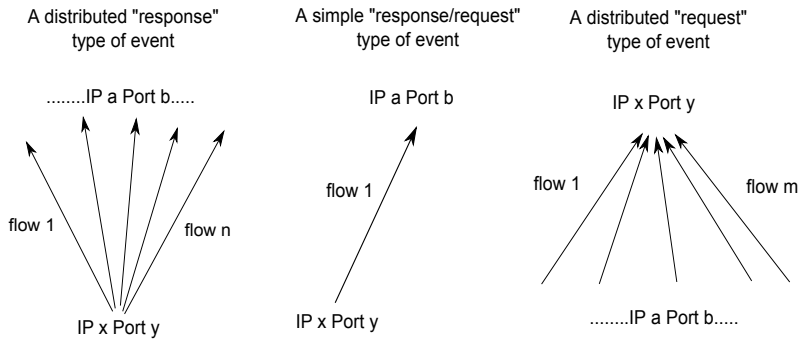

Fig. 1. Illustration of event structure type behavior variations as induced by the use of background event extraction rules.

maximization (while surely extracting a number of false events, too). We design the extraction rules as follows.

endpoint rules

1. all flows share the same source IP and source port
2. all flows share the same source IP and destination port
3. all flows share the same destination IP and source port
4. all flows share the same destination IP and destination port

similarity rules

1. all flows share the same protocol
2. all flow sizes, i.e. number of bytes transferred, are similar
3. all the flows are close enough in time

Each of the subsets of the above set of 4 address rules $\{1..4\}$, combined together with the full set $\{1, 2, 3\}$ of similarity rules, constitute a prescription on how to extract event from the used background traffic netflow data. Applying all of these combinations exhaustively on the data, as described in Algorithm 1, we can extract high diversity of network events.

In the background event extraction process, the incoming NetFlow sequence is simply being split into discrete time intervals to consider the events within, which is sort of a standard in network traffic monitoring and analysis [25]. The sets of flows are then stored in clusters, growing as the new NetFlows are coming from the traffic. The incoming flows are checked against the rules that define a mask on the flow endpoints and other static properties, such as protocol in our simple scenario. Each of the flow clusters also stores a number of properties such as the values of common endpoints, average size of flows, flags, etc. These properties are maintained during the whole process, resulting into events that represent diverse network behavior, including variety of structural configurations of events, such as those depicted in Figure 1, which are similar to *graphlet* patterns explored in other works on flow traffic classification [13].

Algorithm 1 Background event sample extraction

```
1: ruleBase ← the rule combinations for generic events
2: flows ← the incoming sequence of NetFlow
3: winSize ← maximal time spread of flows
4:
5: function DistributeFlows(flows)                          ▷ flow distribution
6:     minTime ← time of the first flow ∈ flows
7:     maxTime ← time of the last flow ∈ flows
8:     intervalCount ← (maxTime − minTime)/winSize
9:     flowWindows ← emptySet
10:    for all flow ∈ flows do
11:        idx = (flow.time − minTime)/winSize
12:        flowWindows_idx = flowWindows_idx ∪ flow
13:    return flowWindows
14:
15: flowWindows ← DistributeFlows(flows)
16:
17: procedure ExtractEvents                                 ▷ flow aggregation
18:    for all flowWindow ∈ flowWindows do
19:        for all rule ∈ ruleBase do
20:            for all flow ∈ flowWindow do
21:                if flow.properties ⊆ rule.properties then
22:                    JoinClusters(flow,rule)
23:
24: events ← emptySet
25:
26: procedure JoinClusters(flow, rule)                      ▷ rule checking
27:    added ← false
28:    for all cluster ∈ events do
29:        endpoints = rule.endpoints ∧ cluster.endpoints
30:        if endpoints ⊆ flow.endpoints then
31:            if flow.size ≈ cluster.averageSize then
32:                cluster = cluster ∪ flow
33:                endpoints = endpoints ∩ flow.endpoints
34:                cluster.endpoints = endpoints
35:                added ← true
36:    if ¬added then
37:        nC = {flow}
38:        nC.endpoints = flow.endpoints ∧ rule.endpoints
39:        events = events ∪ nC
```

We note that the configuration of this exhaustive extraction strategy ensures that every event detected by Camnep from the same traffic data will also occur as some event, or a subpart thereof, in the extracted background-traffic event set, and so the proper event recall is truly maximal. Moreover, various valid parts of such an event might get extracted as well.

Example 2. For instance, in our setting all single flow ssh requests to an endpoint server within a short time period will get extracted as a compact "many-to-one ssh-requests" event, yet some of those requests might get grouped by the source IP only, as well as all of these single requests will again stand as separate ssh-request events in the final background event dataset.

This does not mean that all flow subsets of an event are extracted, but generally rather those maximal sets that exhibit some interesting and common behavior of flows, according to their structural and other properties, as restricted by the generic prescription rules and the nature of the traffic itself.

5 Learning

The goal of learning a classifier from the network traffic data is to automatically generalize knowledge from a small number of ground truth attacks available, with an accuracy comparable to that of a human domain expert. For that we use similar, relatively simple and intelligible representation of an event as a set of 30 features calculated from standard aggregation functions applied over the sets of contained flows and their attributes, such as count of flows, average of #packets, standard deviation of #bytes, etc.

There is a wide variety of different methods for training classifiers from data. However, as it turns out, the characteristics of the network traffic data which we study in this paper make a straightforward application of off-the-shelf machine learning tools a bit problematic. The two main reasons for this are as follows. First, we have labels only for the positive examples (the ground-truth malicious events) and we only know about the other larger part of the data coming from the background traffic that the fraction of positive examples is extremely small. Second, the datasets which we work with are highly class-skewed, with a skew usually of several orders of magnitude (the ratio of $1 : 10^5$ is easily possible in this domain) as total majority of the events in the background traffic correspond to normal activities.

Besides straightforward approaches, such as considering the background traffic globally negative, there are remedies to tackle this problem. In particular, there is variety of anomaly detection approaches, utilizing notably one class classifiers [14], and semi-supervised strategies, exploiting the unlabeled majority of data through the few labeled samples and some a priori assumptions [11]. A similarly motivated approach is to exploit the assumption that malicious events are rare and similar to each other, which would ideally mean that they occupy small neighborhoods of ground-truth malicious events in the feature space. We further explore this intuition through a method based on ensembling techniques.

5.1 Subsampled random forests

One of the best-performing [10] machine learning algorithms these days is the Random Forest [4]. The algorithm creates an ensemble of decision trees by bagging. In each iteration of the Random Forest algorithm, a bag of examples is randomly selected by sampling with replacements the same number of examples as is in the original dataset. Then, one tree is learned upon each such resampled dataset where, in the learning process, the features to be used in a split node are only selected from a randomly sampled subset of features. The output of a learned Random Forest classifier is computed as the average of votes of the individual trees in the ensemble. The classification of a test example is then performed by comparing the output of the random forest classifier to a selected threshold. Thus, by decreasing the threshold, one can usually increase the number of true positives but also, at the same time, the number of false positives. The output of Random Forest can also be thought of as the confidence of classifying an example as positive. It can so be used to rank events from most suspicious to least suspicious.

However, Random Forest itself is not well suited for learning from imbalanced datasets. There have been two main remedies proposed to tackle this problem [6], one of them based on cost sensitive learning, and the other on the use of sampling. The cost sensitive methods performed poorly in our scenario and thus we further explored the sampling methods that are able to target a deeper issue. The issue is that when the number of background-traffic examples becomes really large, it is no longer just a matter of balancing the cost thresholds, as the trees in the ensemble become very correlated in the sense that they share very similar decision boundaries, and therefore the output of the learned forests will be equal for a majority of events, making it impossible to reasonably rank them by their suspiciousness. To solve this problem, we utilize a simple modification of the original majority class subsampling strategy [6], somewhat similar to [3], stated as follows. When sampling the positive examples of which there are only a few instances, we follow the normal strategy and sample with replacements a set of the same size as the positive set, but when sampling the negative examples we sample a smaller set (also with replacements) with the subsampling ratio given as a parameter. Intuitively this simple strategy should increase the variability among the trees.

Example 3. An illustration of the intuition giving a rationale for the above method is shown in Figure 2. Here, the examples are 1-dimensional and can acquire values from 0 to 1. There are 10^5 learning examples sampled from the uniform distribution on the interval $(0, 1)$. The examples to the left from 0.1 threshold are classified as positive and the rest as negative. We train a bagged ensemble of threshold decision rules on this data. The dash-dotted line in Figure 2 corresponds to outputs of an ensemble obtained by conventional bagging, whereas the solid curve corresponds to outputs of an ensemble trained by the method in which negative examples are always subsampled for each bootstrap sample. We can notice that the subsampled case still gives the same output for

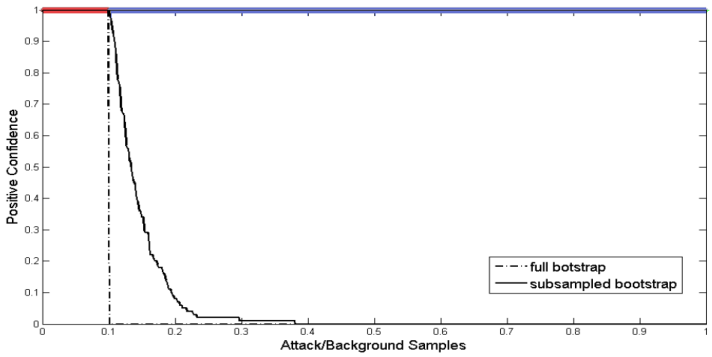

Fig. 2. Illustration of the intuition for the subsampled random forest method.

the examples in the area where positive samples occur, which is a desirable behavior in the case of expert-labeled ground-truth positive events. On the other hand, the output decays more slowly in the area of negative examples and thus allows us to rank them. We expect that a similar effect takes place while subsampling the negative examples in the random forest model. The results of our experiments performed in Section 6 actually suggest that this is indeed the case.

The motivation for the subsampling method employed in this work is different from the motivation for the methods developed in literature for dealing with imbalanced datasets using Random Forests [6, 3]. In our case, what we need to achieve is to have different output values for as many examples as possible, so that we can rank the events by our confidence that they are attacks and not a normal traffic. We also know class labels of only a subset of positive examples and no class labels for negative examples. On the other hand, in existing works, the main motivation is to improve classification performance when all class labels are known.

6 Evaluation

We performed several experiments in order to evaluate the ability of different types of models to detect malicious events in the data. For training, we only used the ground truth and the background traffic data. For testing on separate test-sets, we created labeled events using the Camnep system utilizing the expert rules. As described in Section 3, this means that some of the false positive events detected by some classifiers may in fact correspond to true positives, and so our evaluation is a bit skewed towards pessimism, but there is no other tractable way of obtaining labels for the events in the background traffic part of our dataset. Moreover, as we will see, some classifiers (notably the subsampled random forests) are able to obtain some very good ROC curves, which indirectly confirms accurateness of the labeling provided by Camnep.

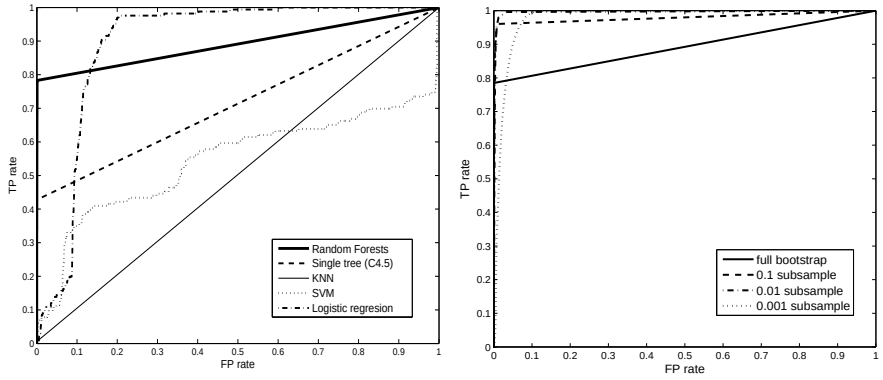

Fig. 3. A comparison of the selected state-of-the-art classifiers with ROC analysis, where Random Forest and a single tree model dominate the beginning of the ROC(left). Enhancement of Random Forest with the introduced subsampling technique boosts the performance further(right), with the subsample ratio varying from a full bootstrap(default Random Forest) to 0.001 subsample.

6.1 State of the art classifiers

The first logical step, after performing initial data exploration experiments to justify the machine learning approach, was to test off-the-shelf, state-of-the-art machine learning algorithms. The algorithms we tested were decision trees, random forests, support vector machines, logistic regression and k-nearest neighbors. Recall that the extreme class-skew of the dataset can render some otherwise very good conventional algorithms unusable for our problem. Although some remedies could be found for those, the instant superior performance of trees and their ensembles, especially in the beginning of ROC, as displayed for comparison in Figure 3 (left), led us to exploit further in their direction.

6.2 Subsampled random forests

The performance of Random Forests, although superior to other standard approaches, was still not satisfactory. Most of the attacks were correctly classified right from the start, i.e. with very low false positive rates, but the rest of the system attacks was not covered until the false positive rates reached unbearable levels (Figure 3, left). On the other hand, we noticed that these system attack samples were possible to be in covered in some way, e.g., by generalized linear regressions. However, we could expect such behavior based on the intuition presented in Section 5.1. There we explained why subsampling the background traffic in the process of learning random forest classifiers should be beneficial in settings like ours, where there are a few positive examples and a large number of unlabeled samples, most of which are probably negative.

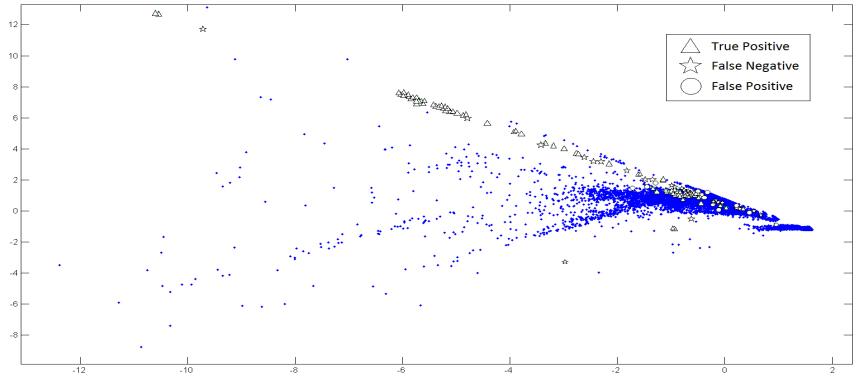

Fig. 4. CLS visualization of the model behavior by the means of ROC characteristics

We have tested various subsample ratios as parameters for the subsampling strategy, and the resulting influence on the ROC performance of the tree ensemble can be seen in Figure 3 (right). Generally, we can note that the full bootstrap performance, equal to the performance of the default Random Forest, is always improved by the introduction of subsampling in terms of the area under the ROC curve. The decreasing sizes of bootstraps are gradually progressing towards higher true positive rates at the expense of increasing the false positives under different paces. Although the results are not conclusive for choosing a generally optimal sub-sample ratio, we further favor the rates from around 0.01 to 0.1, providing a satisfactory compromise between the true and false positives.

6.3 Model analysis

We have shown that an automatic method is able to learn a model with a performance similar to the expert rules designed in Camnep. To further evaluate and interpret the behavior of the learned model, we visualize the results in a similar fashion to ROC characteristics as follows. We mark the system attack samples covered by the model as true positives, those that were not covered as false negatives, and possibly regular traffic covered as false positives. All these characteristics were then recorded for varying classification threshold scores. Using a fast dimension reduction technique called Calibrated Least Squares (CLS) [21], that was best able to visually distinguish these samples, we display the results, for a chosen threshold, in Figure 4. From the projection we identify that the most significant dimensions are mostly correlated with the overall size of events, which is generally reflected in the expert rules, too. To search for some closer correspondence with Camnep rules, we further explore the interpretability of the model by extracting a single tree model from the ensemble, using a learning technique introduced in [1]. For the interpretation, the extracted tree had to be rapidly pruned, which renders its node decisions partially inaccurate. Nevertheless, from the splits in the nodes of the tree, we were able to identify a

number of patterns with an interpretation similar to the original Camnep rules. Some examples of the extracted decisions from the paths commonly leading to malicious-behavior-signed leaves in the tree, are as follows.

- If the percentage of unique sizes of flows is lower than 40%
- If the average number of packets transferred is lower than 6
- If the shared destination port of flows equals 22
- If the number of unique source ports is greater than 5

We are aware that the generality and accuracy of these decisions are disputable, and some bias towards the attack samples used is exhibited (e.g., the 22 port rule for ssh-cracking). Nevertheless, we present the interpretation as an interesting option to validate design of expert rules, by which we indirectly confirm the validity of both the model and the original rules used in Camnep.

7 Conclusions

In this paper we presented and analyzed adaptive means to learn an intrusion detection model, from an extremely small number of ground truth attack representatives, with a performance competitive to the expert rule driven process provided in a part of Camnep IDS system [22].

To capture more complex aggregation properties of the malicious behavior in scope, we addressed the problem on the event level, built on top of the NetFlow layer, for which we introduced a fast, generic, heuristic event extraction procedure. Recognizing the lack of ground truth event samples in the domain of IDS, we utilized this procedure to extract variety of presumably normal event behavior from a given background university NetFlow traffic. With the new extended training sample set, consisting of a few original ground truth attacks and essentially unlimited number of background events, we presented machine learning methods to compare against the output of Camnep on the same traffic.

To surpass the results of the state-of-the-art algorithms tested, we introduced a novel random-forest-based subsampling method for tuning an ensemble tree classifier, with a semi-supervised motivation, to account for the imbalanced nature of the data. This method is based on bootstrap subsampling, suppressing the correlation of individual trees, w.r.t. class frequencies, and by its means we increased the ensemble performance beyond the scope of regularly used tuning, such as sample or misclassification weighting. The results of the experiments performed suggested for the benefits of our bootstrap subsampling method, the overall ability to learn to detect the attacks from the traffic in scope, and also indirectly confirmed validity of the approach and particular rules from Camnep.

Acknowledgements. This work was supported by Cisco sponsored research project "Modelling Network Traffic with Relational Features", and Czech Technical University internal grant SGS14/079/OHK3/1T/13. Part of this work was done while the second author was with KU Leuven where he was supported by Jan Ramon's ERC Starting Grant 240186 "MiGraNT: Mining Graphs and Networks, a Theory-based approach".

References

1. Van Assche, Anneleen, and Hendrik Blockeel. Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In *Machine Learning: ECML 2007*, pages 418–429. Springer, 2007.
2. Karel Bartos and Martin Rehak. Trust-based solution for robust self-configuration of distributed intrusion detection systems. pages 121–126, 2012.
3. Jerzy Błaszczyński, Jerzy Stefanowski, and Łukasz Idkowiak. Extending bagging for imbalanced data. pages 269–278, 2013.
4. Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
5. Umang Kamalakar Chaudhary, Ioannis Papapanagiotou, and Michael Devetsikiotis. Flow classification using clustering and association rule mining. In *Computer Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD), 2010 15th IEEE International Workshop on*, pages 76–80. IEEE, 2010.
6. Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 2004.
7. Benoit Claise. Cisco systems netflow services export version 9. 2004.
8. Reda M Elbasiony, Elsayed A Sallam, Tarek E Eltobely, and Mahmoud M Fahmy. A hybrid network intrusion detection framework based on random forests and weighted k-means. *Ain Shams Engineering Journal*, 4(4):753–762, 2013.
9. Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9):1194–1213, 2007.
10. Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014.
11. Te Ming Huang and Vojislav Kecman. Semi-supervised learning from unbalanced labeled data–an improvement. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 802–808. Springer, 2004.
12. Hongbo Jiang, Andrew W Moore, Zihui Ge, Shudong Jin, and Jia Wang. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 299–304. ACM, 2007.
13. Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.
14. Shehroz S Khan and Michael G Madden. A survey of recent trends in one class classification. In *Artificial Intelligence and Cognitive Science*, pages 188–197. Springer, 2010.
15. Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning intrusion detection: supervised or unsupervised? pages 50–57, 2005.
16. Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. pages 333–342, 2005.
17. John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM transactions on Information and system Security*, 3(4):262–294, 2000.
18. Masayoshi Mizutani, Keiji Takeda, and Jun Murai. Behavior rule based intrusion detection. pages 57–58, 2009.
19. Adetunmbi A Olusola, Adeola S Oladele, and Daramola O Abosede. Analysis of kdd99 intrusion detection dataset for selection of relevance features. In *Proceedings*

of the World Congress on Engineering and Computer Science, volume 1, pages 20–22, 2010.

20. Roberto Perdisci, Guofei Gu, and Wenke Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 488–498. IEEE, 2006.
21. Tomáš Pevný and Andrew D Ker. The challenges of rich features in universal steganalysis. 2013.
22. Martin Rehak, Michal Pechoucek, Pavel Celeda, Jiri Novotny, and Pavel Minarik. Camnep: agent-based network intrusion detection system. pages 133–136, 2008.
23. Martin Rehak, Michal Pechoucek, Martin Grill, Jan Stiborek, Karel Bartoš, and Pavel Celeda. Adaptive multiagent system for network traffic monitoring. *IEEE Intelligent Systems*, (3):16–25, 2009.
24. Dario Rossi and Silvio Valenti. Fine-grained traffic classification with netflow data. pages 479–483, 2010.
25. Chakchai So-In. A survey of network traffic monitoring and analysis tools. *Cse 576m computer system analysis project, Washington University in St. Louis*, 2009.
26. A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of ip flow-based intrusion detection. *Communications Surveys Tutorials, IEEE*, 12(3):343–356, Third 2010.
27. Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali-A Ghorbani. A detailed analysis of the kdd cup 99 data set. 2009.
28. Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. Intrusion detection by machine learning: A review. *Expert Systems with Applications*, 36(10):11994–12000, 2009.
29. Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. Random-forests-based network intrusion detection systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(5):649–659, 2008.