Featureless discovery of correlated and false intrusion alerts

Kidmose, Egon; Stevanovic, Matija; Brandbyge, Søren; Pedersen, Jens Myrup

Published in:
IEEE Access

DOI (link to publication from Publisher):
10.1109/ACCESS.2020.3001374

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EMERGING APPROACHES TO CYBER SECURITY

Received April 30, 2020, accepted May 30, 2020, date of publication June 10, 2020, date of current version June 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3001374

Featureless Discovery of Correlated and False Intrusion Alerts

EGON KIDMOSE[1,2], MATIJA STEVANOVIC[1,2], SØREN BRANDBYGE[2], AND JENS M. PEDERSEN[1]

[1]Department of Electronic Systems, Aalborg University, 9220 Aalborg Øst, Denmark
[2]LEGO System A/S, 7190 Billund, Denmark

Corresponding author: Egon Kidmose (egk@es.aau.dk)

This work was supported by the Innovation Fund Denmark, Industrial Ph.D. Programme, under Grant 5016-00018.

ABSTRACT Malware and cyber-attacks cause substantial damage to corporations. A common countermeasure is Intrusion Detection Systems (IDSs). Unfortunately, IDSs typically raise many alerts on a single incident, with redundant information, and false alerts that are only noise to analysts. For out-of-the-box performance, the impact is so large that alerts are of limited practical use. Existing solutions rely heavily on domain expertise, in feature engineering procedures and explicit algorithms. This has substantial negative impact on the costs of development, deployment, and maintenance. Using feature engineering as part of a method boosts classification metrics, but requires substantial investment, of data science and security expertise, for each deployment. We find that reliance on domain expertise and feature engineering severely inhibits the feasibility of applying existing correlation and filtering methods in practice. To address this, we propose a novel approach for correlating and filtering, with the constraints that methods must be without feature engineering and methods must consume alerts as text strings. Two implementations are presented and evaluated on a partly private and on a public data set. Our implementations are unable to compete with existing methods on common detection metrics, suggesting that investing feature engineering pays of towards those. Measured on practical metrics for filtering and correlating, our implementations are promising, while at the same time cutting the cost of deployment, according to the constraints. Consequently, we find it of practical relevance to consider methods, like ours, that are much easier and cheaper to deploy, compared to the existing ones.

INDEX TERMS Alert correlation, alert filtering, clustering, intrusion detection system, latent semantic analysis, malware detection, recurrent neural network.

I. INTRODUCTION

Over the past decades we have seen tremendous advances in information technology, coupled with a widespread adoption. While bringing many benefits, this has also made society and corporations vulnerable to a multitude of attacks. Criminals have identified a large range of illicit but financially rewarding schemes based on this, of which many involve infecting victims with malware or otherwise intruding on networks. Such schemes include harvesting credentials from victims, and abusing resources for sending spam e-mails, or launching Distributed Denial of Service (DDoS) attacks, and often rely on running malware on victim hosts. Malware often implements a Command and Control (CnC) channel back to infrastructure controlled by the criminals, and the victim host is then referred to as a bot, which is part of a botnet. A single botnet has been measured to encompass 180, 000 victims [1], while estimates on their size range in millions [2], [3]. An estimate of 500 million computers are infected and enrolled in botnets every year, causing a global loss of 110 billion US dollars/year [3]. The cost of cybercrime in general have been estimated as high as 445 to 608 billion USD [4].

A prerequisite for efficiently applying countermeasures is the ability to detect intrusions. Detection capabilities can be provided by Intrusion Detection Systems (IDSs), which in essence observe activity and raises alerts on malicious activity. We discern between Host-based IDS (HIDS) and Network-based IDS (NIDS), with the former having access to more detailed information internal to the host, and the

The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba.

latter being less intrusive, and with a potential to cover multiple hosts. Prior work suggests that the information available in the network is sufficient, and thus NIDSs are preferred [5]–[8]. Examples of NIDSs that have had commercial breakthroughs are Snort [9], Bro IDS [10], and Suricata [11]. However, the usefulness of IDSs is limited by poor quality of alerts. From a theoretical point of view this is unsurprising, as perfect malware detection is impossible [12]. In practice, this problem has shown to be substantial, [13] reporting that three instances of snort deployed in large financial institution on average produce 411, 947.18 alerts per day. It is obviously unfeasible to process so many alerts manually, and the vast number of low-quality alerts causes alert fatigue as well as waste of resources.

In order to address the problem of poor alert quality, we observe that the problem is compounded by two underlying problems. A substantial share of alerts are false alerts, where no malicious activity occurred, but imperfections in the IDS still lead to an alert being raised. As an example, nine in ten alerts are found to be irrelevant in [13]. To security analysts this is pure noise, and false alerts should be filtered out prior to any manual investigation. We refer to the problem of false alerts being present in IDS output as the **filtering problem**. The other problem is that alerts with correlated information are raised as separate alerts. This leads to waste, when multiple investigations are initiated independently, and also when the analyst has to search for correlated alerts in order to gain understanding of a multistage intrusion. Numbers reported by [13] tells that they received roughly 40, 000 relevant alerts per day. Even with a conservative (high) estimate of hundreds of incidents per day, the number of alerts per incident is in the hundreds. Ideally, links between correlated alerts should be identified for the analyst, and only one alert should be raised per incident. We refer to the problem of correlated alerts being present in IDS output as the **correlation problem**.

A naïve approach to address the problems is to assume that IDSs are overly sensitive and tune them accordingly. This conflicts with the requirement that IDSs should not miss any incident, and the need for detailed information in post-detection analysis. Furthermore, it leads to expert time being wasted on tuning at setup and during daily operations. It seems a more prudent approach to let IDSs be overly sensitive, and then address the filtering and correlation problems in a pre-processing step. This is supported by such approaches achieving good results [6]–[8], [13]–[20], and by existence of commercial products such as Cisco Security MARS and FireEye.

Existing methods for correlation and filtering that use Machine Learning (ML) all rely on feature engineering, which is the art of (semi-)manually designing transformations for available data, before applying ML algorithms. It is a generally proven method for improving performance, but it comes with some drawbacks, which needs to be considered. Severyn and Moschitti states that feature engineering is a tedious task, and it reduces adaptability by requiring substantial re-engineering [21]. Anderson *et al.* refer to it as a pain point in building trained systems, which require, yet challenge, computer scientists with PhD-level training, and note that it requires dramatically more iteration and adjustment than what is immediately apparent [22]. Khurana *et al.* is a third example of similar position, stating that feature engineering is largely manual, a complex exercise, iterative, based on trial and error, and often the most time-consuming step in a data science workflow [23]. Gauging the exact cost of feature engineering is hard, and likely impossible to do in general, but there are examples where the effort or cost related to ML and feature engineering has been reported:

The Netflix Prize was an open competition to improve performance on a movie recommendation problem, based on a fixed data set [24]. It shows that a 10% incremental improvement over a working system was worth at least a million dollars, and that a lead time of years was acceptable. The effort made by contestants during the first third of the competition (11 months) is indicated by numbers reported in [24]: 20.000 teams registered, 2.000 teams submitted results, and 13.000 different results were submitted.[1] We judge that a large effort must have been made, also considering that the contest continued for a total of 33 months. The IBM DeepQA project [25] had the goal of creating a system named Watson, capable beating human grand masters at the Jeopardy quiz show. The project had a core team of 20 scientist and engineers and lasted 3 years. Watson applies *more than 100 different techniques* and a principle of *many experts* to find answers, indicating that feature engineering is a key aspect. 5.500 experiments were conducted, consuming a staggering 30.117 CPU-years, which supports the claim that many iterations are required. The man-hours and compute resources spent must have been substantial.

Like feature engineering, expertise from the networking and security domains can aid in correlating and filtering alerts, but again the drawbacks prompt for some attention. Applying heuristics and domain expertise can potentially improve performance, but adaptability and maintainability might see significant negative impact. Consider as an example a correlation and filtering method that relies on comparing source and destination IP addresses. This reflects the domain knowledge that network intrusions have a source and destination, which might boost performance under certain conditions, and appears to be sensible. Consider then an attack where the source can be spoofed, such that it can be chosen freely by the adversary. This would prompt for redesigning the method, redoing feature engineering, and changing the implementation, as the assumption is broken. Similar arguments can be made about swapping NIDS with HIDS, changing or updating IDS, different attackers having different methods of operation, etc. The essence of the issue is that domain experts make assumptions about the conditions,

[1] We use the

so the inevitable changes in conditions can invalidate the assumption, the method and, the implementation.

We believe that feature engineering and domain expertise can provide a performance improvement, but the price to pay is substantial resources and loss of adaptability. The loss of adaptability means that the returns of invested resources is limited, which can make methods unfeasible and explain limited adoption by practitioners. Or in other words, it is not worth it for corporations to invest in developing a method and doing feature engineering when adaptability inhibits reuse. We propose a shift from seeking excellent performance on classification metrics with methods that are unfeasible in practice, towards sufficient practical performance with methods that are feasible due to significantly decrease implementation costs and improved adaptability.

Our contributions are 1) to question if feature engineering and embedded domain expertise is the right approach to the filtering and correlation problem, 2) to present a novel, general approach that is free from feature engineering and embedded domain expertise, 3) to present two implementations of the general approach, and 4) to evaluate the methods on two different data sets.

The paper is structured as follows: We survey related work in Section II. Our general approach is presented in Section III, along with our own method based on Neural Networks and our adaption of Latent Semantic Analysis (LSA) to the present problem. In Section IV the two data sets and relevant procedures are described, along with a proposal of metrics to capture performance for practical purposes. Results of applying the two methods to the two data sets are presented in Section V, and discussed in Section VI, before we conclude on this work in Section VII.

II. RELATED WORK

In this section we survey existing work on solving the correlation and filtering problems. First, we outline the idea of submitting IDS alerts to a pre-processing step, highlighting relevant prior art. Then we survey approaches and techniques that have been used, and we pay particular attention to feature engineering. As data for evaluation poses some interesting challenges, we summarise the options, reasoning, and choices found in related work. As this work builds on the claim that there is a mismatch between practical application and commonly used metrics, we finally highlight prior work supporting this.

A. PRE-PROCESSING

Leaving out the details of correlating and filtering for a start, we introduce the notion of pre-processing. As discussed in Section I, and as evident from the extent of existing work in Table 1, IDSs raise too many alerts for manual processing. One solution is to introduce a pre-processing step prior to manual processing. A pre-processing step takes alerts as input, and outputs **hyper alerts**, which represents one or more lower level alerts. Fig. 1 illustrates this, with individual alerts on the left, and the result after pre-processing on the right;

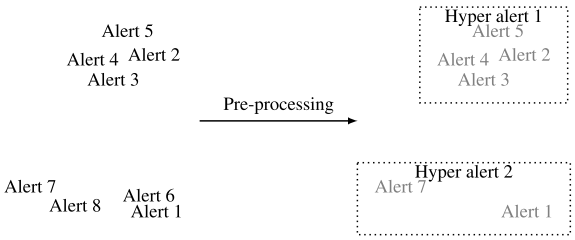

FIGURE 1. Pre-processing alerts into hyper alerts.

Alert 6 and 8 are filtered out and the remaining alerts form two hyper alerts. If done correctly, pre-processing solves the filtering and correlation problem. In the related work, hyper alerts are also referred to as meta alerts or reports.

A simple pre-processing method is to fuse alerts by applying the *atom model*, where incoming alerts are compared to existing hyper alerts; Alerts are fused with the hyper alerts that they most probably belong to, unless the highest probability is below a given threshold; then the alert forms a new hyper alert [16]. In [16] the hyper alert is simply represented by the most recent alert, while [14] applies a complex structure that enables heterogeneous information sources. Fusing multiple correlated alerts into fewer hyper alerts, addresses the correlation problem, and brings down the number of items requiring manual investigation. Fusing alerts into graphs has also been proposed, providing some insights to how attacks develop [15].

The filtering problem can be addressed when fusing, by filtering out alerts that fail to meet some criteria for being fused. This relies on the assumptions that false alerts fail to fuse, while true alerts are fused. An evaluation of this approach, using 10 incidents in a fully controlled lab network, was found to produce hyper alerts on all incidents [6]. Filtering can also be done at the hyper alert level, i.e. after fusing, as suggested by [16]. The assumption behind this is that false alerts are fused together, and that false hyper alerts are easier to filter than individual false alerts. Approaches for filtering only have also been proposed, and could be combined with subsequent correlating [13], [31].

Having clarified the ideas of pre-processing, correlating, and filtering, we now move on to a study of the approaches found in prior work.

B. EXISTING APPROACHES

A commonly used technique in existing methods is to implement a function for estimating similarity of alerts, or alternatively estimating the probability that two alerts are correlated. Naïve implementations, relying on human heuristics to estimate correlating probability, have been demonstrated by [14]–[16]. Applying human heuristics, both [16] and [14] constructed functions to estimate correlation probability. Formal parameter optimisation was applied by [16]. Probability of correlation was estimated with two supervised ML algorithms in [15]. Rather than labelling real samples for training, this work relies on manually crafted training samples, and is therefore fully dependent on human heuristics.

TABLE 1. Overview of related work.

Paper	[26]	[14]	[16]	[27]	[17]	[15]	[6]	[18]	[13]	[28]	[29]	[30]	[31]
Correlating	x	x	x	x	x	x	x	x		x	x	x	
Filtering			x	x	x		x	x	x	x		x	x
Similarity		x	x			x				x	x	x	
Rules	x			x	x		x						
Graph-based				x		x						x	
Machine Learning			x			x	[a]	x	x		x		x
Prototype available	x						x						
Qualitative	x	x		x		x	x	x		x			
Quantitative			x	x				x	x			x	x
New metrics				x							x		x
Real world traffic		x					x	x	[b]	[b]	[b]		x
Controlled attack		x	2000 [32]	2000 [33]		2000 [33]		1999 [34]	2010	2013		2012	
Controlled Malware							x				[29]		

Columns are publications, sorted by increasing year of publication. The first two rows, (Correlating and Filtering), show which problems are addressed. The following four rows, (Similarity, Rule, Graph-based, Machine Learning), indicate the techniques used. The next four rows, (Prototype available, Qualitative, Quantitative, New metrics), pertain to prototype availability and evaluation method. The last three rows, (Real world data, Controlled attack, Controlled malware), describe the data used for evaluation, where a reference indicates that the data is available, and the year indicate when the data was created or recorded. [a]Machine Learning is not used for filtering or correlating (But in the two IDS plug-ins SLADE and SCADE). [b]Background traffic only.

Another approach to handling the correlation problem, is to rely on rules that describe relations between different attacks steps and assume that alerts represent distinct steps. Generally, correlating is implemented by associating necessary prerequisites, and potential consequences to alert classes. The rules can be manually defined, as suggested by [17], or mined from data as in [15], [27]. The latter two also encompasses varying degrees of correlation, as opposed to binary, and [15] involves continuously adjusting probabilities when in operation. Hyper alerts are formed according to the given rules, with associativity enabling more complex scenarios to be captured. The procedure of [6] also relies on predefined rules, describing that certain types of alerts must have been raised if a hyper alert is to be produced. With rules, and in particular with a notion of varying degrees of correlation between alerts, hyper alerts are naturally expressed as graphs, and some work specifically apply graph-based methods or modelling [15], [27], [30].

While some of the earlier discussed methods include ML, they also rely heavily on heuristics based on domain expertise. However, correlating and filtering can be left to ML to a large extent. Clustering corresponds well with the correlation problem, and classification with filtering. An example of how ML can learn to solve the correlation and filtering problem from data, without embedding domain expertise, is presented by [18]. The method is composed of Self-Organising Map (SOM) and k-means clustering. SOM is an unsupervised application of Neural Network (NN) that learns to map samples into a space of lower dimensionality, so that similar samples are close, and dissimilar samples are disparate in the output space. The k-means clustering algorithm is used to obtain well defined clusters in the output space. Each of two stages applies SOM and then k-means, first to group alerts by correlation, and second to filter out groups of false alerts.

C. FEATURE ENGINEERING

Some feature engineering methods applies to ML problems in general, while others embed domain expertise and are thus limited to the given domain. Feature selection and transformations are examples of general feature engineering. Selection is widely used, but we highlight [13], who select features that only applies to network traffic, and a feature that does not exist for anomaly-based IDSs, leading to a method that only applies to rule-based NIDS. Mean/variance normalisation is an example of a general feature transformation applied by [18], which mitigates that some ML methods have bias towards features with high variance or mean. The use of domain expertise for feature engineering appears to be most impacted by the challenges outlined in the introduction, yet all found examples of correlation and filtering with ML make use of this. A common example is a feature transformation that produces a distance between two Internet Protocol (IP) addresses. The distance is computed as the number of most significant bits that the two addresses have in common [14]–[16]. This method only applies when source and destination is known, meaning that it fails for HIDS alerts on activity not involving the IP layer, or if a source IP cannot be obtained reliably. A more specific example is that [16] recognised that source, and destination IPs were swapped for a certain class of alerts. Part of their feature engineering was to compensate for this IDS error, by swapping source and destination IPs back, when an alert was of the given class. The implications of this is that time has been wasted on debugging the IDS, and the method is tailored to the specific situation. BClus, the ML method proposed by [29], relies on complex aggregations over time and low-level network features. Finally, all existing methods rely on fixed attributes to be extracted from alerts, which again create strong ties to specific IDSs and their output format.

D. DATA SETS AND EVALUATION

Evaluation of filtering and correlation methods is commonly carried out on obsolete or private data sets. As attacks, malicious traffic, and the noise from benign traffic, evolves over the years, evaluations with the DARPA 1999 and 2000 data sets [33], [34], or the DEFCON 8 CTF data set [32], becomes obsolete. The malicious activity in these data sets is bluntly obvious by current standards. The malicious activity includes well known attacks launched over plain text protocols, DDoS attacks and attacks during a CTF game, where the actors have limited motivation to act stealthy. Current attacks in the real world are expected to be significantly stealthier and more difficult to detect. One example is that malware is now often trying to hide among legitimate user activity, such as web browsing and e-mailing, with both malicious and benign traffic being encrypted. While the data set of [32]–[34] are obsolete, they have the benefit of being publicly available, which allows for verifying and comparing results. At the time when [16] and [14] was published, their used data sets where likely still relevant, but noting that [15], [18] and [28] used 6-, 11- and 13-year-old data sets, suggests a challenge with data availability within our field. This is supported by the observation that all surveyed evaluations with real world traffic, include no references to the data sets, suggesting that it is not available to the public. We suspect that these highly valuable data sets are kept private to avoid privacy issues. The impact of this is evident from work such as [6]. The used data sets appear substantial and realistic, but others are hindered in assessing the data, reproducing the results, or reusing the data for comparison, as the data is not available. Furthermore, part of the evaluation ignores false positives, as it is not feasible to thoroughly inspect the data in order to label it. The return of investing in labelling data increases with reuse, so sharing data could potentially make it feasible.

Only one example of recent, publicly available, and substantial data on real malware is known to the authors [29]. This particular data set contains traffic traces of real malware executed in a controlled environment. The omission of real benign user activity eliminates privacy concerns in relation to publishing the data. However, in reality benign activity will be present, and an evaluation should reflect this to provide a reasonable representation of real conditions.

An approach to generate current data sets that can be shared without compromising privacy is presented in [35]. The idea is to model benign activity, so that it can be emulated in a lab, along with controlled attacks, or execution of malware, such as that of [29]. The reasoning is that model descriptions and data generated from models are not problematic with regards to privacy, hence data generated in the lab can be shared freely. It is unclear exactly how the method can guarantee that sensitive information is not captured by the model and made evident in the generated data, and the method appears to be susceptible to bias through design choices, as is the case for other methods relying on synthesising data. A recent data set has been produced according to the approach, including raw traffic capture and details about the attacks that enable labelling [36].

Another solution is to record data on real activity and then remove the sensitive information before sharing. This can be implemented by e.g. removing network traffic payloads, pseudonymisation by randomising IP addresses, and by truncating traffic to flows. We are not aware of any substantiated claim that this can be done without losing information that can be of use to some methods, and indeed this would seem counter-intuitive. It remains unclear, what notion of privacy is guaranteed, and data labelling remains an extensive task.

Yet another solution is to evaluate existing methods on a private data set, but this requires implementation to be available or to be re-implemented. We are only aware of two examples of work where publicly available implementations are referenced [6], [26], and re-implementing requires substantial effort, without much assurance that it will match the original implementation sufficiently well. This would still not enable reproducing results, as performance likely depends on the data.

E. PERFORMANCE EVALUATION

As evident from Table 1 there are examples of both quantitative and qualitative evaluation in the related work. Qualitative evaluations provide some interesting details of results, but without quantitative metrics it is challenging to compare performance of different methods or variation thereof. For quantitative evaluations, it is common to apply metrics for detection and classification, such as accuracy, precision, recall, F1-score, etc. A gap between these established metrics and the application domain is noted in [29], which also includes a proposal for a set of new metrics to address this. The new metrics include timeliness of detection and correct detection of infections, rather than correct detection of samples. Also focusing on real-world applicability, [31] takes the view that an organisation has a finite bandwidth, or *daily investigation budget*, and measures performances in terms of bandwidth versus recall. This suggests that the classification/detection metrics are mostly of academic interest, and less relevant for evaluating the feasibility and relevance of a method in practice. None of the related work reports on the effort invested in feature engineering, the dependence of domain expertise, or the adaptability.

In this section, we have surveyed existing methods for correlating and filtering. One important finding is that feature engineering and domain expertise are fundamental to all the existing methods. Another important point is that there is support for rethinking how performance is measured, as the classification and detection metrics are somewhat disconnected from the application domain.

III. METHOD

In this section we present a novel approach to how correlating and filtering of IDS alerts can be automated. The novelty of the approach is that there is no use of feature engineering and no use of domain expertise. The approach relies on data

for learning how textual alerts can be ''read'', i.e. encoded in meaningful way. The section consists of four main parts;

- First, we present a general approach that captures the concept. In essence, alerts are considered text, and ML is used to learn a mapping function for encoding alerts.
- Second, we present our implementation of the concept using Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN).
- Third, we present how we have used LSA, a well-known information retrieval method, as another implementation of the general approach.
- Finally, we introduce DBSCAN clustering, another existing method, which we extend to provide labelling of found hyper alerts.

A. GENERAL APPROACH

Having established in the Introduction that using feature engineering and embedding domain expertise can yield methods that are not feasible in practice, we strictly avoid using such. Feature engineering and embedding of domain expertise are examples of making assumptions about the problem, which are translated into optimisation in implementations. Problems arise when these assumptions are broken. We allow only two assumptions about the problem and build our general approach on that. First, we assume that alerts carry sufficient information to determine which are false and which are correlated. We expect no loss of generality from this, as it follows naturally from the assumption that the filtering and correlation problems for a set of alerts can be mitigated. As for the second assumption, we observe that alerts must be consumed by a machine so a general machine-readable representation for alerts is required. Applying a schema implies feature selection and is out of the question. Clearly, it would degrade adaptability to only be compatible with IDSs producing alerts that can fill the schema and by requiring a parser to be maintained. We note that all IDSs familiar to us are capable of presenting alerts as human readable text strings. Also, all representations of alerts that we know of are a subset of all text strings. Consequently, the second assumption is that alerts can be represented as human readable text strings.

To only build upon these two assumptions, we propose an approach, consisting of three phases, as outlined in Fig. 2. In the first phase, a **mapping function** is learned from alerts. The mapping function must be capable of mapping alerts, represented as text strings, into a vector space, which we refer as the **auto-encoded space**. In the seconds phase, the mapping function is used to map alerts into the auto-encoded space. Finally, in the third phase, clustering is applied in the auto-encoded space to obtain **hyper alerts** that represent incidents. This approach requires that a method is capable of learning a discriminative mapping function in the first phase, such that applying it in the second phase yields a representation where alerts can be discriminated by incident, such that the clustering in the third phase is meaningful. If a discriminative mapping function can be obtained with ML, it will be able

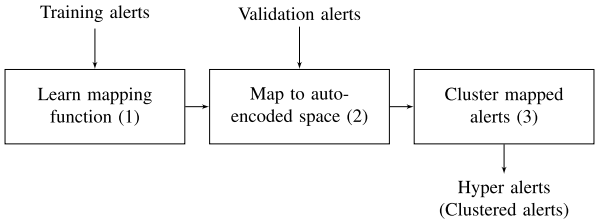

FIGURE 2. Overview of the general approach; First a mapping function is learned from training alerts (1), then it is applied to validation alerts (2), and finally the mapped alerts can be clustered (3).

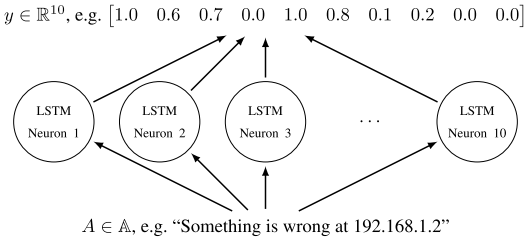

FIGURE 3. A Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) implementation of the function ($\mathcal{M}$), mapping Intrusion Detection System (IDS) alerts into an auto-encoded space of size $n = 10$. [37, Fig. 1].

to replace manual feature engineering and domain expertise as known from existing work. If possible, this eliminates the need for investing in feature engineering, as there is none, and eliminates the issues of adaptability, because the same implementation can be reused as-is in other settings, requiring no manual adaptions, but only data and machine resources for learning the parameters of the mapping function.

Having described the overall proposal, two examples of how a mapping function can be obtained will follow: One using LSTM RNN and one using LSA.

B. LSTM RNN: MAPPING FUNCTION

LSTM RNN is a well-regarded method in the area of Natural Language Processing, and therefore a good candidate for being able to produce a discriminative mapping function. As a neural network-based method, it is computationally efficient and benefits from recent developments in hardware, including utilisation of GPUs for computation. We propose to implement a mapping function ($\mathcal{M}$) with a single layer of n LSTM RNN neurons, as illustrated with $n = 10$ in Fig. 3. The input is an alert from the set of all alerts that can be represented as text ($A \in \mathbb{A}$) and the output is a real-valued vector ($y \in \mathbb{R}^n$), i.e. the auto-encoded space.

$$\mathcal{M}: \ \mathbb{A} \rightarrow \mathbb{R}^n \tag{1}$$

The NN can be trained efficiently on data consisting of input/output pairs, using the Backpropagation algorithm. However, given that no security domain expertise can be embedded in the method, the output part of training data (the auto-encoded representation) is not available. Taking alerts (Input) and creating corresponding points in the auto-encoded space (Output), would produce the required data,

but would also be a clear violation of the independence from domain expertise and feature engineering. Consequently, the mapping function cannot be trained directly.

C. LSTM RNN: TRAINING A MAPPING FUNCTION

To obtain a ML-based approach, without heuristics or feature engineering, for learning a mapping function from data, we elaborate on the meaning of discriminative. To this end, we introduce an indicator function for correlation ($\mathcal{I}$), which outputs 1 for two alerts $((A_i, A_j) \in (\mathbb{A}, \mathbb{A}))$ if they are raised on the same incident, and 0 otherwise;

$$\mathcal{I}\colon (\mathbb{A}, \mathbb{A}) \to \{0, 1\} \tag{2}$$

and we define a similarity function ($\mathcal{S}$), which is similar to $\mathcal{S}$ in structure, but instead applies to alerts mapped into the auto-encoded space $((\mathbb{R}^n, \mathbb{R}^n))$, and outputting a real value for similarity of the alerts ($\mathbb{R}$);

$$\mathcal{S}\colon (\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R} \tag{3}$$

The mapping function and the corresponding auto-encoded space is said to be discriminative iff. uncorrelated alerts are very dissimilar (Cf. eqnarray (4)), and correlated alerts are very similar (Cf. eqnarray (5)). Equation (4) states that the problem of detecting false alerts is equivalent to finding outliers in the auto-encoded space. Correspondingly, (5) states that the problem of grouping correlated alerts is equivalent to clustering in the auto-encoded space.

$$\mathcal{S}(\mathcal{M}(A_1), \mathcal{M}(A_2)) \ll c \quad \text{iff. } \mathcal{I}(A_1, A_2) = 0 \tag{4}$$
$$\mathcal{S}(\mathcal{M}(A_1), \mathcal{M}(A_2)) \gg c \quad \text{iff. } \mathcal{I}(A_1, A_2) = 1 \tag{5}$$

An NN for calculating the similarity of two alerts, as expressed in the left-hand side of (4) and (5), can be implemented as illustrated in Fig. 4. Assuming that similarity in the auto-encoded space can be approximated by the output of the indicator function, this architecture enables us to learn a mapping function. For a data set of alerts labelled with incident IDs, it is trivial to create pairs of alerts, for which correlation is known. The NN can then be trained, with Backpropagation, using the alert pairs as input, and $\mathcal{I}(A_1, A_2)$ as target output. NN weights make up all the trainable parameters, and they are all within the two instances of $\mathcal{M}$. By tying the parameters of the two instances together during training, the result is a set of parameters that can be read out and reused in a single instance of $\mathcal{M}$. Thus, a mapping function can be learned from data, based on these very general assumptions and the proposed training architecture. The evaluation will show if the learned mapping function is discriminative.

D. LSTM RNN: DETAILS OF THE MAPPING FUNCTION

The mapping function is implemented with a single layer of 10 LSTM neurons ($n = 10$, c.f. Equation (1)) with *tanh* non-linearities. A single hidden layer is used as it is the simplest RNN, while it in theory still is enough to estimate any functional mapping [38, pp. 130-131]. Multiple sources

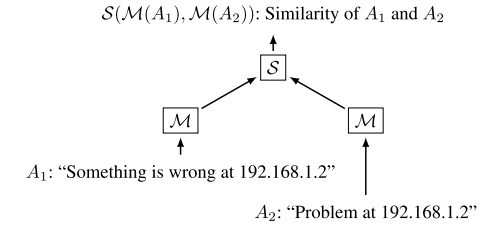

FIGURE 4. LSTM RNN for estimating similarity of two alerts in an auto-encoded space. Two alerts are mapped by two instances of the mapping function ($\mathcal{M}$), to two points in the auto-encoded space. The two points are compared by the similarity function ($\mathcal{S}$) [37, Fig. 2].

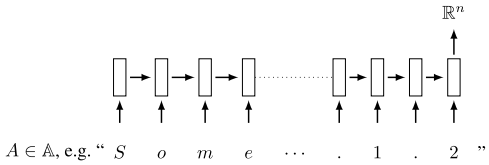

FIGURE 5. Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) reading an alert, letter by letter. Letters at the bottom: Input example. Vertical arrows: Input and output connections. Horizontal arrows: Recurrent connections across elements in the input. Rectangles: Network as seen in Fig. 3, repeated per element in the input, all having the same parameters.

suggest limiting layer size to what yields adequate performance [39, pp. 22.11-22.12], [40, part 3], [41, p. 158]. In previous work we experienced that using 10 neurons performed adequately, while keeping the execution time under a practical limit of 2 days [37]. Consequently, the layer size, and thereby the dimensionality of the auto-encoded space, is set to $n = 10$. Input is One-Hot encoded as encouraged by [39, p. 22.5]. Forward recurrence is used rather than bi-directional, for the improved performance. Finally, cosine similarity is used as the similarity function $\mathcal{S}$.

Alerts are sequences of letters, hence $\mathcal{M}$ is implemented with an RNN. This is illustrated with Fig. 5, which is a more detailed view on Fig. 3. Each letter of the alert (A) is read in sequence, from left to right, and fed to the LSTM network (each rectangle represent the network, consisting of a single layer of n neurons). This connection is depicted with the row of vertical arrows. Each rectangle represents the same network, but each with a new internal state, updated for each step. Recurrence is implemented with the horizontal arrows, signifying that the previous state (along with the current letter) is the inputs to determine the new state. The network output is ignored until the last step, in which it is used as the overall output (vertical arrow, upper right). The output at the last step encodes the entire alert in a vector ($\mathbb{R}^n$). Each recurrence step is a fixed set of matrix multiplications and additions, thus the computation of $\mathcal{M}$ scales well, i.e. linear in the length of the alert. The length is expected to be limited as alerts are intended for human interpretation.

The training procedure used is illustrated with Fig. 6, with the following details and considerations; Pairs of alerts are built by making all possible combinations of the relevant sets of alerts, (Cartesian product in Set Theory or cross-join with

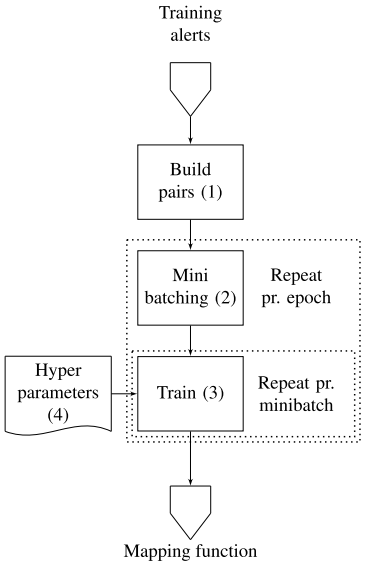

FIGURE 6. Procedure for training the network depicted by Fig. 4. Training alerts, labelled per incident, are used to build pairs of alerts with known correlation(1). Pairs are divided into mini-batches (2), and each mini-batch is used for training by repeatedly applying the Back Propagation algorithm (3), using a set of hyper-parameters (4). Mini-batching (2) and training (3) is repeated for each epoch to produce the mapping function (I.e. the weights for the LSTM RNN).

self on a constant attribute in Relational Algebra). It is noted that if n_{alerts} is the size of the training alerts set, then the set of training pairs will be of size n^2_{alerts}, thus adding training alerts has a big impact on the memory and computation needed for training.

Training is repeated for 10 epochs, each epoch utilising the entire data set, and each incrementally improving performance. For each epoch, training is done with randomly sampled mini-batches of 10000 samples, in order to add noise and to match computations to hardware. Experience is that noise aids in avoiding bad local optima. Training on a mini-batch is done with the Back-Propagation algorithm, meaning Stochastic Gradient Descent (SGD) towards a locally optimal loss (Binary cross entropy), controlled by a learning rate of 0.003. Back-Propagation with SGD is generally not guaranteed to find a global optimum.

E. LSA

LSA is an well-known and widely used Information Retrieval method for learning a transformation from unlabelled data [42].[2] Put shortly, LSA is to count word occurrences per document to obtain a Term Frequency (TF) matrix and apply Singular Value Decomposition (SVD) to it. From the SVD, one can construct a transformation that maps a vector of word counts into a space of lower dimension, where the basis vectors are those that account for the most variance in the TF matrix. The assumption is that this compression preserves and extracts the most useful information, while removing noise. The output of the transformation is typically much smaller size than the input. We are not aware of any previous examples of LSA being applied to IDS alerts, but it appears obvious to consider alerts as documents. Furthermore, the derived transformation corresponds perfectly to our mapping function, and the output space to our auto-encoded space, making LSA appear as a viable implementation of our general method.

The details of our LSA implementation, and corresponding considerations are as follows: Alerts often contain more than words, e.g. IP addresses, timestamps, or rule IDs, hence we count N-grams in place of words. To limit computation and memory usage, only 1-, 2-, and 3-grams are considered. N-grams found in only one or more than half of the alerts are expected to convey little information, so they are discarded. N-grams that are found in few alerts are assumed to be particularly relevant for describing those alerts, and conversely those N-grams found in many alerts are expected to convey little information. To implement this, the Term Frequency weighted by Inverse Document Frequency (TF-IDF) is used in place of plain TF. To limit computation and memory usage, the top 10.000 N-grams by term frequency across the corpus are used, the rest are discarded. Finally, the SVD is truncated to the top 100 largest components, i.e. the most significant topics, and they form the auto-encoded space ($n = 100$). n is chosen such that there can be multiple topics per incident, while not allowing too many, as the less significant topics carry more noise. Truncating also decreases the size of the model.

F. CLUSTERING PROCEDURE

Second and third phases of the general approach (Recall Fig. 2) are captured by Fig. 7. The second phase applies the learned mapping function to alerts to obtain their representation in the auto-encoded space.

The third phase implements the DBSCAN clustering algorithm to the alert representations. In accordance with $\mathcal{S}$ being the cosine similarity, DBSCAN uses cosine as distance.

To gain an understanding of the partitioning of the auto-encoded space in a systematic way, clusters are labelled with incident IDs in the following way: Core points are used to represent clusters, and thereby the partitioning. The most frequent incident ID for alerts in a cluster is taken to be the incident ID of the cluster. To handle varying frequency of alerts from different incidents, the majority vote is weighted by the inverse frequency of alerts from each incident in the whole data set. By imperfections in the learned mapping function, or in the used clustering algorithm, some clusters can be labelled as holding mostly false alerts. This is accepted, as grouping false alerts in a cluster still removes redundancy in the final result, although filtering them out altogether is preferred. Using the partitioning and the concepts of closeness from DBSCAN, it can be predicted to which incident in the training data a new alert belongs to, or if it is a false alert. Validation alerts are mapped into the auto-encoded space and

[2]LSA is also well supported in scientific computation toolkits, cf. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

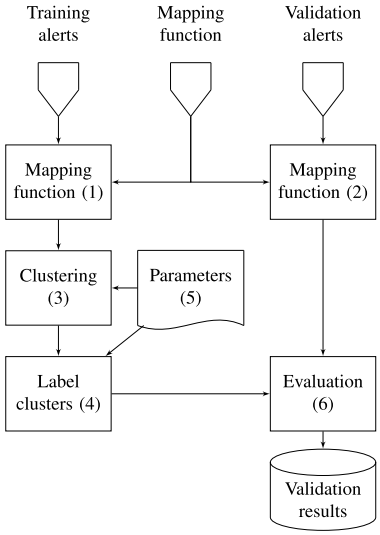

FIGURE 7. Data flow for clustering. The mapping function (C.f. Fig. 6) encodes both training (1) and validation alerts (2). DBSCAN clustering is applied (3), and clusters are labelled as detailed in Section III-F (4), using the same DBSCAN parameters (5). Finally, the labelled clusters, from training alerts, are applied to the encoded validation alerts to label them with incidents (6).

compared to the core points found in the training data set. If a point is within *eps* of a labelled core point, the validation point is classified as belonging to the same incident as the core point. If a point is not within *eps* of a labelled core point, it is an outlier, and the alert is predicted to be a false alert.

G. SECTION SUMMARY

To summarise, we have presented a general approach for extracting useful information from IDS alerts, to automate correlation and filtering without any feature engineering or expert knowledge, and with no ties to particular IDSs. Additionally, we have proposed two methods for learning mapping functions. The first employs a LSTM RNN architecture and learns from labelled alerts. The second is LSA applied to unlabelled alerts. Note that the implementations are available as open source.[3] We now turn to the methodology used for evaluating the two methods.

IV. EVALUATION

In this section, we describe how we evaluate our proposal. Our null-hypothesis is that correlation and filtering methods only can perform adequate for practical application if feature engineering, domain expertise, or both are used in the methods. To test this, the two methods presented earlier are applied to two data sets and scored by metrics capturing practical performance. In this way we seek to demonstrate that our proposed approach and implementations are counterexamples, leading us to reject the null-hypothesis, and conclude that adequate filtering and correlation can be achieved without feature engineering and embedding of domain expertise. To this end, we here introduce the two data sets, a view on practical alert processing, and a set of metrics that capture practical performance.

[3]https://github.com/kidmose/lstm-rnn-correlation.

TABLE 2. Overview of alerts raised on the MCFP bot traffic (Part of Data Set 1).

Bot (I.e. Incident)	Alerts	Alerts (%)
1	100	4.36 %
2	184	8.53 %
3	317	14.69 %
4	328	15.20 %
5	390	18.07 %
6	395	18.30 %
7	444	20.57 %
Total	2158	100.00 %

As already discussed in Section II, data for evaluation poses a challenge when working with correlation and filtering. The training procedure for the LSTM RNN and the need for hard evaluation metrics further constrains the possible data set to those where labels are available. We have identified two usable data sets, one from the Malware Capture Facility Project (MCFP) [29] and the other being the CIC IDS 2017 data set [36].

A. DATA SET 1: MCFP BOT TRAFFIC MERGED WITH BENIGN

Complete traffic traces of bot malware operating in a lab is provided by the MCFP[4] [29]. This data set is interesting because holds data on real bot malware samples, that have been allowed to execute in a lab, to simulate an incident. As traffic traces are provided per execution of individual bot malware sample and without benign traffic, labels can be applied efficiently to each incident before merging. On the other hand, the absence of benign activity and the resulting absence of false alerts pose a challenge to the representativeness of the data. To overcome this, we propose a procedure for obtaining false alerts and for merging all the alerts, as outlined in Fig. 8. The procedure can be reused by others, resulting in a data set that is available to anyone for the parts where privacy allows it, and where the private part can be created according to the following description. This offers a not previously described point in the trade-off between representativeness, privacy, and availability of data.

The per bot traffic traces are processed with the Snort IDS[5] resulting in alerts, which are labelled with incident. Table 2 provides an overview of the data. For further details, see our previous work [37].

False alerts are obtained by monitoring the traffic of tens of office user PCs for 35 days, using a Snort instance configured as above. It must be asserted that the alerts are indeed false, which is done by asserting three independent conditions. The first condition is that the monitored hosts/network is part

[4]Available from:https://mcfp.felk.cvut.cz/publicDatasets/

[5]Using Snort version 2.9.7.6, DAQ version 2.0.6, built in rules and https://snort.org/rules/snortrules-snapshot-2976.tar.gz, accessed October 6th, 2015.

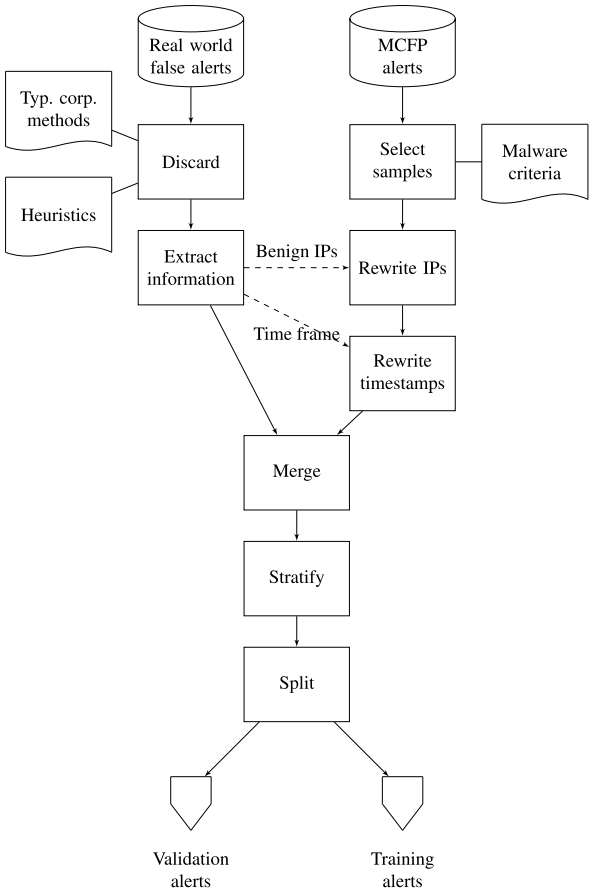

FIGURE 8. Data flow graph for data preparation.

of a well-managed corporate infrastructure, with fundamental information security mechanisms in place, such as user rights management, firewalls, antivirus, patch management, etc. This is believed to decrease the likelihood of a machine being infected. The second condition is that typical corporate methods and procedures, for identifying infected hosts are in place. This involves various solutions in the categories of both HIDS and NIDS, beyond what is part of the data collection setup described herein. The third condition is that heuristics are applied to screen for alerts that are likely to be true. Manual inspection is applied to the heuristically selected subset of alerts, with the goal of determining if the alert is false. Any host identified by an alert that cannot be confirmed to be false, is handled as if it was infected. The first heuristic is based on infections likely causing at least one priority 1 alert, thus all priority 1 alerts are inspected. The second heuristic is that any mention of the keywords `malware`, `malicious`, `blacklist`, `trojan`, and `bot` in the alert is a strong indicator of a true alert, thus alerts matching any of these keywords are inspected. The manual inspection involves the interpretation of alerts, the collection of relevant information, (exploit description, host name look-up, the presence of exploited service and patch level, other alerts on either source or destination host, interview with system owner, blacklists), and finally a conclusion on whether

TABLE 3. False alert data set in numbers, with details of what was discarded.

	Alerts	Prio. 1	Prio. 2	Prio. 3	Hosts	Corp. hosts
Recorded data set	5.548.539	104	4.028.411	1.520.024	10.907	1.552
Discard list 1	251.904	0	251.880	24	-	-
Discard list 2	713.737	37	607.293	106.407	-	-
Used data set	4.582.898	67	3.169.238	1.413.593	9.333	1.236

Discard list 1: Alerts with source or destination IP flagged by typical corporate methods and procedures. **Discard list 2**: Alerts with source or destination IP that was found in a priority 1 alert which could not be rejected as false. Note that alerts can be counted in both discard lists, hence summing across rows will not add up.

there is reason to suspect that any host was infected. In cases where suspicion remains, both source and destination hosts implied by the alert are considered infected. Consequently, alerts involving confirmed or potentially infected hosts are discarded.

During 35 days, 5.548.539 alerts were raised, involving 10.907 different IP addresses, of which 1.552 belong to unique hosts in the corporate domain. The existing corporate methods and processes had detected infections leading to 251.904 of the alerts being excluded (0 priority 1 alerts, 251.880 priority 2 alerts, and 24 priority 3 alerts). No alerts matched the keywords. 104 alerts of priority 1 were raised and inspected manually. Among these, 71 alerts were confirmed to be false alerts. The remaining 33 suspected true alerts lead to 713.737 alerts being excluded, (37 priority 1 alerts, 607.293 priority 2 alerts and 106.407 priority 3 alerts). Combining the above different grounds for discarding alerts, produces the used set of false alerts. Table 3 summarises the discarded alerts and the distribution across the different priorities.

Simply pooling the true and false alerts into a single data set will introduce artefacts, where it is trivial to discern intrusions in time and in IP space. To avoid this, the alerts are rewritten to fulfil the following statements:

1) First alert of each incident is after first false alert.
2) Last alert of each incident is before last false alert.
3) Alerts of the same incident maintain their relative difference in time.
4) IP address of each incident victim is replaced with one appearing in the false alerts.
5) Resulting IP address must be unique for each incident victim.

The ambiguity that remains from the above is handled by randomising, using independent continuous uniform distributions across the possible time span, and independent uniform discrete distributions, across the set of possible IP addresses. To control the training time, ($O(n^2_{alerts})$) the data set is stratified by discarding false alerts, so that they make up 50% of the data.

B. DATA SET 2: CIC IDS 2017

The CIC IDS 2017 data set[6] [36] stands out from other recent data sets because it includes traffic traces, which can be

[6]http://www.unb.ca/cic/datasets/ids-2017.html. Access kindly provided to the authors upon request.

TABLE 4. Alerts raised on the CIC IDS 2017 data set (Data Set 2).

Incident	Alerts Count	Alerts %	Alerts (Stratified) Count	Alerts (Stratified) %
False alerts	172447	39.93	1894	43.88
Bot	2812	0.65	200	4.63
DDoS	23111	5.35	200	4.63
DoS GoldenEye	2200	0.51	200	4.63
DoS Hulk	223760	51.81	200	4.63
DoS Slowhttptest	2172	0.50	200	4.63
DoS slowloris	162	0.04	162	3.75
FTP-Patator	441	0.10	200	4.63
Heartbleed	92	0.02	92	2.13
Infiltration	3312	0.77	200	4.63
PortScan	596	0.14	200	4.63
SSH-Patator	387	0.09	200	4.63
Web Attack BF	195	0.04	195	4.52
Web Attack SQLi	15	0.00	15	0.35
Web Attack XSS	158	0.04	158	3.66
Total	431860	100.00	4316	100.00

Break-down by labels. The second and third columns are before stratification. The fourth and fifth columns are after.

passed to Snort, and because the accompanying homepage describes the incidents with sufficient details to label alerts by incident. This enables use of the data set for evaluating our filtering and correlation methods. Furthermore, the data is presumable available to anyone, is not subject to privacy constraints, and represents contemporary incidents.

Alerts are again obtained by processing the individual traces with Snort. Labels are applied to alerts by comparing the timestamp of the alert to the time interval of incidents and by comparing the IPs of both. An alert is labelled with a given incident when both of two conditions are met: First, the alert timestamp must match the time interval of the incident. Second, either source or destination IP of the alert must match either the attacker or victim of the incident. A network device performed Network Address Translation (NAT) in the data collection setup, which makes is meaningless to match both source and destination to attacker and victim. Lack of comprehensive details on port usage for incidents and details of the NATing process makes it impossible to reconstruct the translations. This also leads to the exception that both IP addresses of the NATing device are ignored when matching. There are no cases where multiple labels match an alert. If an alert fails to match on either or both of the time and IP conditions, it is labelled as a false alert.

The number of total alerts amounts to 431860, which compares poorly with the 4361 alerts in the final Data Set 1. A stratified down sampling of Data Set 2 to approximately one hundredth of the previous size will mean that some incidents are going to have only one or zero alerts, which is problematic. Instead, only incidents with more than 200 alerts are down-sampled to 200 alerts, and then false alerts are down-sampled to match the total size of Data Set 1. A summary of the alerts and their labels is shown in Table 4.

C. METRICS

A set of commonly used metrics for detection and classification problems, are used for evaluating performance. We use accuracy, precision, recall, and F1 score as defined by (6)-(9):

$$\text{accuracy} = \frac{TP + TN}{FN + FP} \tag{6}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{7}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{8}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{9}$$

True Positive count (TP) refers to the count of correct positive outcomes/class assignments, False Negative count (FN) to the count of incorrect negative outcomes/assignments to other classes, False Positive count (FP) to the count of incorrect positive outcomes/class assignments and, True Negative count (TN) to the count of correct negative outcomes/assignments to other classes.

The use of these metrics enables comparison with other works within the field, and they are well suited to describe performance on detection and classification problems in general. However, they fail to capture the actual value gained from applying correlation and filtering methods, as also mentioned in Section II. In the following, we propose a set of metrics to address this, motivated by a model of operation and by general costs associated with a Security Operations Center (SOC).

Any corporation or other large organisation is expected to have some unit handling (hyper) alerts and incidents, which we refer to as a SOC. Automatically reacting to every (hyper) alert, including the false, will clearly cause too many disruptions, hence it is assumed that (hyper) alerts are processed manually, before any action is taken. The processing of an (hyper) alert, by an analyst, is assumed to have a unit cost. In reality, the required time and level of expertise varies, and thereby so does cost, but with large volumes this simplification is reasonable. Given a hyper alert, we assume that one alert is picked at random from the hyper alert, and used to determine what has happened, and what action is to be taken. The analyst is assumed to be perfect, meaning that when analysing an alert, the analyst correctly identifies the victim, the incident, and whether it is a false alert. Assuming that the alerts of a hyper alert are sufficiently homogeneous, i.e. hyper alerts mostly hold alerts of the same incident, this can be expected to work well. Given this model of a SOC, the ratio of alerts to hyper alerts is also the ratio between cost of analyst workload, with and without filtering and correlating. In the following, this will be formalised as the Alert Reduction Factor (ARF).

The other side of operating a SOC, is the risk of missing incidents altogether. Setting the cost of this is extremely hard, as incidents might be stopped from causing harm through other mechanisms than those initiated by the SOC, or in worst case they can potentially be the end of the corporation. Acknowledging that the cost of an incident is impossible to describe in general, we propose to consider the rate at which incidents are missed. This provides insights as to the

probability that a random incident will be missed and can be used as an input to risk management, where uncertainties already need to be handled. A metric named Incident Miss Rate (IMR) is defined to capture this.

For use in the formal definitions, the following sets of alerts, hyper alerts and incidents are defined:

$$\mathbb{A}_{in} : \text{Set of all alerts, before pre-process} \tag{10}$$
$$\mathbb{A}_{out} : \text{Set of all alerts, after filtering by pre-process} \tag{11}$$
$$\mathbb{I}_{in} : \text{Set of all incidents, before pre-process} \tag{12}$$
$$\mathbb{I}_{out} : \text{Set of hyper alerts, produced by pre-process} \tag{13}$$
$$|\mathbb{X}| : \text{Number of elements in set } \mathbb{X}$$

As already stated, ARF describes ratio from all input alerts to the number of hyper alerts:

$$\begin{aligned} ARF: \ & \mathbb{A}_{in}, \mathbb{I}_{out} \to \mathbb{R} \\ & \mathbb{A}_{in}, \mathbb{I}_{out} \mapsto \frac{|\mathbb{A}_{in}|}{|\mathbb{I}_{out}|} \\ ARF \leq |\mathbb{A}_{in}| \qquad & ARF = |\mathbb{A}_{in}| \iff |\mathbb{I}_{out}| = 1 \\ ARF \geq 1 \qquad & ARF = 1 \iff |\mathbb{A}_{in}| = |\mathbb{I}_{out}| \end{aligned} \tag{14}$$

IMR captures how likely it is that an incident will be missed altogether. By comparing how many incidents are in the input data, against how many are represented in the output, it can be measured how often the method makes an error leading to an incident being missed altogether. Ideally this metric is zero, as that means no incidents are missed, while a value of one, means all incidents are missed.

$$IMR: \ \mathbb{I}_{in}, \mathbb{I}_{out} \to \mathbb{R} \tag{15}$$
$$\mathbb{I}_{in}, \mathbb{I}_{out} \mapsto \frac{|\mathbb{I}_{in} \setminus \mathbb{I}_{out}|}{|\mathbb{I}_{in}|} \tag{16}$$
$$IMR \leq 1 \qquad IMR = 1 \iff |\mathbb{I}_{in} \setminus \mathbb{I}_{out}| = |\mathbb{I}_{in}| \tag{17}$$
$$IMR \geq 0 \quad IMR = 0 \iff |\mathbb{I}_{in} \setminus \mathbb{I}_{out}| = 0 \tag{18}$$

The above definition of IMR in (15) provides for the understanding of how the metric is relevant, but the notion of an incident not being represented in the output hyper alerts ($\mathbb{I}_{in} \setminus \mathbb{I}_{out}$) is impractical. Instead, we recall that the analyst picks an alert at random, correctly identifies the incident of that alert, and associates the hyper alert with that incident. The probability that the ID of a given incident, i, comes out as the ID of hyper alert o, is equal to the count of alerts in o that have the given ID, divided by the number of alerts in the hyper alert (C.f (19)). The likelihood that a certain incident ID is not present in any of the produced hyper alert, is the product of the probabilities of not being in each of the individual hyper alerts (Cf. (20)). Based on this, IMR is calculated as the expected rate at which an incident is not being found by the described "oracle analyst" (Cf. (21)).

$$P(ID|o) = \sum_{a \in o} \mathbb{1}(a \in i)/|o| \tag{19}$$
$$P(\neg ID|\mathbb{I}_{out}) = \prod_{o \in \mathbb{I}_{out}} 1 - P(ID|o) \tag{20}$$

TABLE 5. Classification metrics.

Implementation	Data set	Acc.	Prec.	Rec.	F1
LSTM RNN	1	0.737	0.731	0.737	0.720
LSTM RNN	2	0.403	0.401	0.403	0.381
LSA	1	0.826	0.865	0.826	0.793
LSA	2	0.694	0.748	0.694	0.649

Accuracy, Precision, Recall and F1-score for predicting incidents of validation alerts based on clustering of training alerts. Broken down by implementation applied and by data set. The reported numbers are the mean of 10 executions with non-overlapping validation cuts of the data. Higher is better.

$$\begin{aligned} IMR &= \sum_{i \in \mathbb{I}_{in}} P(\neg ID|\mathbb{I}_{out})/|\mathbb{I}_{in}| \\ &= \sum_{i \in \mathbb{I}_{in}} \prod_{o \in \mathbb{I}_{out}} \left(1 - \sum_{a \in o} 1(a \in i)/|o|\right) /|\mathbb{I}_{in}| \end{aligned} \tag{21}$$

V. RESULTS

Both of the two implementations (LSTM RNN and LSA) has been applied to both of the two data sets (Data Set 1, MCFP merged with false alerts, and Data Set 2, CIC IDS 2017). Each of the four combinations of implementation and data set was executed ten times, each time learning a mapping function and finding clusters from training cuts that consists of $9/10^{th}$ of the data. The remaining $1/10^{th}$ made up the non-overlapping, left-out validation cut for each execution.

To enable comparison with prior work, classification metrics for using the labelled clusters to predict incidents of are presented in Table 5. Numbers are the mean over the ten executions.

Table 6 holds the domain specific metrics that we have proposed, also for both implementations and both data set. As this is the measuring point that we argue is relevant for practical purposes, worst- and best-case performance is included in addition to the mean. Fig. 9 holds a scatter plot of IMR and ARF for all four combinations and the ten executions of each, i.e. the data aggregated in Table 6. Data set is encoded by the shape using dots for Data Set 1 and squares for Data Set 2. Implementation is encoded by the colour using Blue for LSTM RNN and red for LSA. Performance increases when moving up (Higher ARF) and to the left (Lower IMR).

As a mean to identify and understand systematic errors confusion matrices are included for each combination of implementations and data sets with Figures 10, 11, 12, and 13. Note that the numbers are normalised according to support for each label to avoid suppressing the incidents with few alerts. Ideally, diagonal values are one and off-diagonal values are zero, as this would indicate no confusion among incidents.

VI. DISCUSSION

Two implementations have been presented and evaluated, which calls for a comparison. By the common classification metrics of Table 5, LSA beats LSTM RNN on all metrics (Fixing data sets for a fair comparison). The picture is similar

TABLE 6. Application domain metrics for performance of LSTM RNN and LSA implementations on Data Set 1 and 2.

		IMR			ARF		
Implemen.[a]	DS[b]	min	mean	max	min	mean	max
LSTM RNN	1	$0.00e^{0}$	$6.18e^{-3}$	$1.79e^{-2}$	$8.31e^{0}$	$9.25e^{0}$	$9.82e^{0}$
LSTM RNN	2	$0.00e^{0}$	$6.12e^{-2}$	$1.09e^{-1}$	$2.44e^{0}$	$3.00e^{0}$	$3.39e^{0}$
LSA	1	$0.00e^{0}$	$7.26e^{-17}$	$1.33e^{-16}$	$2.16e^{1}$	$2.33e^{1}$	$2.54e^{1}$
LSA	2	$8.93e^{-2}$	$9.93e^{-2}$	$1.64e^{-1}$	$4.19e^{0}$	$4.55e^{0}$	$5.07e^{0}$

Aggregation spans 10 non-overlapping validation cuts of the data. Incident Miss Rate (IMR) describes how likely it is that an incident is missed, so lower is better. Alert Reduction Factor (ARF) describes the reduction in manual effort, so higher is better. [a] Implemen. is an abbreviation of "Implementation". [b] DS is an abbreviation of "Data Set".

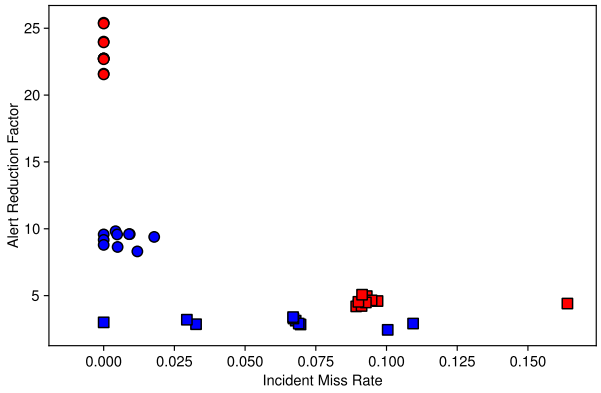

FIGURE 9. Application domain metrics plotted for LSTM RNN (Blue) and LSA (Red) implementations, applied to Data Set 1 (Dots) and Data Set 2 (Squares). Each point represents one of 10 non-overlapping validation cuts of the data (10 cuts for each of the four combinations, 40 points total). Incident Miss Rate (IMR) describes how likely it is that an incident is missed, so left is better. Alert Reduction Factor (ARF) describes the reduction in manual effort, so higher is better.

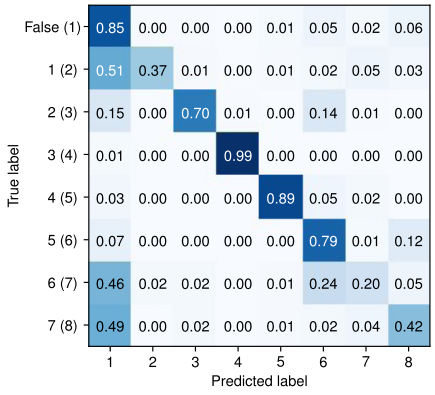

FIGURE 10. Normalised confusion matrix for detection based on mapping function learned with LSTM RNN on Data Set 1. Diagonal values of 1 and off-diagonal values of 0 is ideal.

for domain specific metrics of Table 6 with the exception of IMR on Data Set 2. For IMR on Data Set 2, LSTM RNN beats LSA, but the values are less than an order of magnitude apart. These metrics clearly ranks LSA over LSTM RNN, while the domain specific metrics largely agree, but on one point they disagree or at least turns out inconclusive. One possible

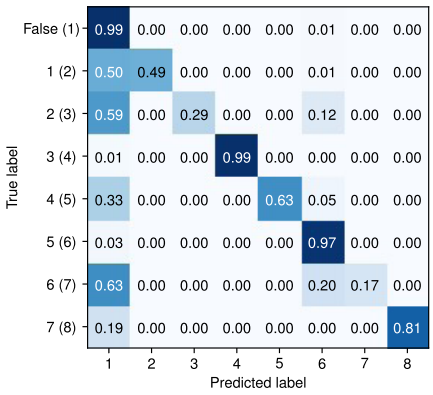

FIGURE 11. Normalised confusion matrix for detection based on mapping function learned with LSA on Data Set 1. Diagonal values of 1 and off-diagonal values of 0 is ideal.

interpretation is that classification metrics are better suited for capturing performance differences, while the domain specific metrics fail to separate the two implementations. Another is that the domain specific metrics show the right picture in the light of practical application value, while the conclusion drawn on classification metrics is wrong due to a disconnect from the application domain. The missing link between classification metrics, the correlation and filtering problems, and the values and risks of the application domain is an interesting topic to explore even further.

To better understand the difference in performance between LSTM RNN and LSA we inspect the per execution performance as plotted in Fig. 9. It is noted that for Data Set 2 (Squares), which unanimously is the most difficult of the two, the implementation appears to behave different. While LSA (Red) consistently outperforms LSTM RNN (Blue) in terms of ARF (High is better), the picture for IMR is more complex (Left is better). LSA is most consistent with a single outlier, while the performance of LSTM RNN spans a larger range in what appears as four different steps. The single outlier for LSA can be explained by inspecting the confusion matrices of each execution, and their aggregate which is presented in Fig. 13. The aggregated and nine of the ten confusion matrices are similar in that they have non-zero diagonal values, except for *DoS Hulk* alerts. This indicates that in general the procedure of learning a mapping function with LSA, applying it, clustering the result, and making prediction using the clusters will capture at least some alerts from all incidents, except *DoS Hulk*. The one exceptional execution, represented by the outlier red square in Fig. 9, differs in that the diagonal value for *SQL Injection* is also empty, indicating a failure to capture this incident. So where nine executions of LSA on Data Set 2 failed completely to capture exactly one specific incident, the tenth execution stands out by failing for an additional incident, which explains the increased IMR of the outlier. We note that 58% of the errors for *DoS Hulk* are due to confusion with another volumetric DoS attack happening two days later. It is unsurprising that the *SQL*

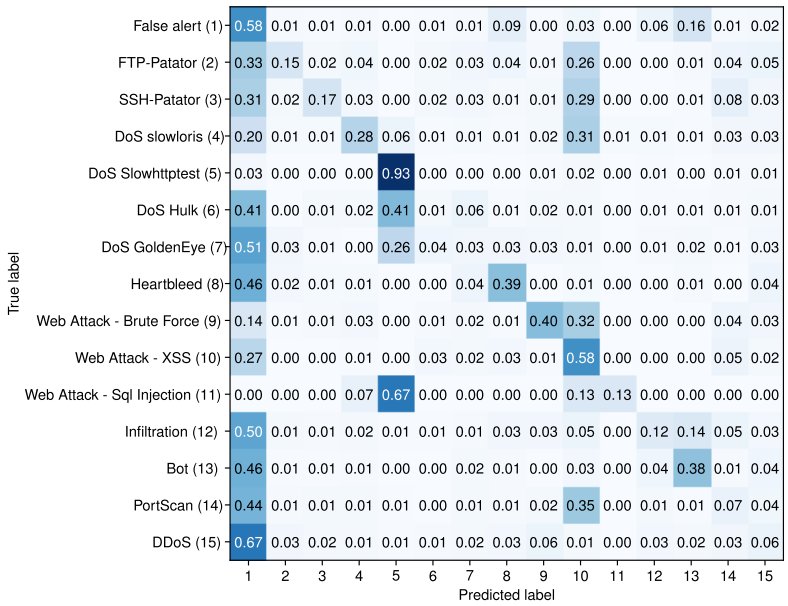

FIGURE 12. Normalised confusion matrix for detection based on mapping function learned with LSTM RNN on Data Set 2. Diagonal values of 1 and off-diagonal values of 0 is ideal.

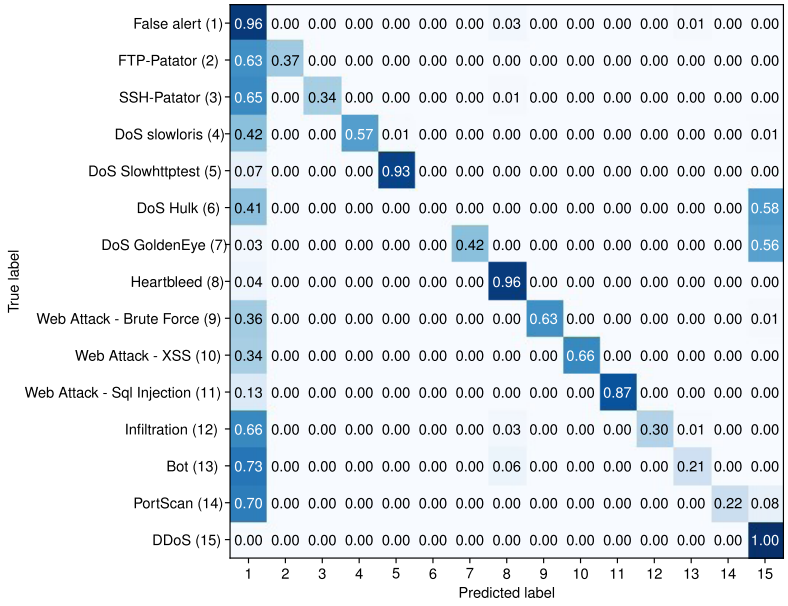

FIGURE 13. Normalised confusion matrix for detection based on mapping function learned with LSA on Data Set 2. Diagonal values of 1 and off-diagonal values of 0 is ideal.

Injection incident triggers the second failure, as it is by far the incident with the lowest number of alerts (15 alerts or 0.35% of Data Set 2, as per Table 4). With 13 or 14 alerts in the training data and only one or two in the validation data we find it somewhat surprising that our approach was able to capture this very imbalanced class in nine of ten executions.

Returning to the topic of classification metrics vs. the proposed domain specific metric, one noteworthy point is that LSTM RNN only achieves 40.3% accuracy on Data Set 2. As per Table 4 43.88% of the alerts are false, so crude "Filter out all alerts" or "Label all alerts by most common label" implementations would beat LSTM, and it is therefore tempting to state that LSTM RNN fails for Data Set 2. However, the domain specific metrics in Table 6 shows that LSTM RNN is still capable of reducing the number of alerts to a third. We claim that any practitioner or manager responsible

for a SOC would find it adequate to cut a large workload down to a third, which is a point that the classification metrics misses completely. The results also indicate that the implementation can be expected to miss 6.12% of incidents. This has to be weighed against the benefits, in particular the limited cost of deployment and maintenance. In settings where it is important to lower the resources invested, and where there is willingness to accept a higher risk, this could very well be acceptable. Where it is crucial to minimise risk, and where large investments are of no concern, this might not be sufficient. In the end, finder the optimal point in this trade-of requires known costs which are not general, and we leave it to practitioners to pass judgement on this. Again, the classification metrics of Table 5 fail to capture the above, while the proposed domain specific metrics describes the gain (ARF) and risk (IMR), which can be used in a practical setting to determine if the implementation is relevant. The domain specific metrics proposed are therefore the best way to capture correlation and filtering performance for practical purposes.

Both domain specific metrics (Table 6) and classification metrics (Table 5), are remarkably better on Data Set 1 compared to Data Set 2. Fig. 9 generally confirms that executions running on Data Set 1 (Dots) exhibit better performance than executions with Data Set 2 (Squares). The one exception is one of the ten executions of LSTM RNN on Data Set 2 which exhibits an IMR of 0, which is on par with the best executions on Data Set 1. This suggests that Data Set 1 is significantly less challenging than Data Set 2. To explore this, one could evaluate more methods on the same data to understand if it is only less challenging for the implementations of our approach or if it is generally easier. Being able to observe this difference in the first place highlights the value of evaluation on multiple data sets, as the common practice of using 10 validation cuts does not identify this issue. Possible reasons that Data Set 2 is harder than Data Set 1 includes: It holds more incidents (14 vs 7), it spans a shorter period of time (5 days vs 35 days), the incidents are more diverse, the incidents are more contemporary, and the incidents are perhaps more random as they are driven by human rather than software (bot malware).

Inspecting the confusion matrices for the two implementations and the two data sets (Fig. s 10, 11, 12, and 13), provides for some interesting observations. We find three to be particularly interesting. First, most of the outcomes are on the diagonal, meaning that ground truth label matches the predicted label. If the output of the mapping function, which is the input for clustering and classification, is random and without relevant information this pattern is extremely unlikely. Therefore, this proves success on learning a mapping function without feature engineering or embedding of domain expertise. If the performance is adequate is discussed below. The second observation is that the first column, which indicates a prediction of ''false alert'', is also substantial. This indicates that a true alert was mistakenly classified as false, i.e. filtered out. Keep in mind that due to the nature of the filtering and correlation problems such an error is not equivalent to missing an incident, but it contributes to the risk of that happening. It does however show that the mapping function, clustering, and classification in combination are less than perfect solutions, which makes it relevant for further exploration. A specific direction could be to explore if it is possible to adjust the aggressiveness for filtering and correlating, in order balance the trade-of between IMR and ARF in practice. It is also possible that some of this is due to bad labelling. False alerts might have occurred on the malicious parts of Data Set 1 due to e.g. background activities of the operating system, and malicious activity in the benign part might have slipped through our procedure described in Section IV-A. For Data Set 2 we did observe some inconsistencies in timestamps between the descriptions and the labelled flow data that is distributed along with the raw traffic. The flows were not used for this work, but it indicates that errors are present, in this particular case perhaps due to human involvement. Thirdly, there are some significant deviations from the above two types of observations. In Fig. 10 there are few errors (70% to 99% are correct) for incidents 2-5, and the errors made are not confusions towards false alerts. This can be due to the LSTM RNN implementation being well suited for capturing the relevant information, as the LSA implementation (Fig. 11) are much more prone to filter out incidents 2 and 4. Both implementations have a tendency to confuse 12% to 14% of the alerts on incident 2 and 20% to 24% of alerts on incident 6 as being incident 5. This might be explained by similarity in the raised alerts, which is possible as it is two examples of bot malware. In Fig. 12 we see many deviations from the diagonal and the first columns. The most significant is 67% of alerts on *Web Attack – Sql Injection* (11) being confused as *DoS Slowhttptest* (5). As both incidents are using the HTTP protocol to deliver application level attacks this confusion is not very surprising, and with only 15 total alerts on (11) it is expected that the bias is towards (5), cf. Table 3. Other significant deviations are other DoS attacks being confused as *DoS Slowhttptest* (5). In Fig. 13 the only substantial deviation from the diagonal and the first column is that 58% and 56% of *DoS Hulk* (6) and *DoS GoldenEye* (7) are confused as being *DDoS* (15). This is can be explained by all three being DoS attacks, and in particular by DoS Hulk also spoofing random sources thereby mimicking the distributed nature of DDoS. Furthermore, both DoS Hulk and DoS GoldenEye use randomly generated data (User-Agent and Referred in HTTP) and similar techniques (HTTP Keep-Alive and no-cache also in HTTP). In summary, many outcomes are on the diagonal, meaning that they are correct, many are in the first row, meaning they are incorrectly filtered out, and the remainder of errors have possible explanations based on deeper understanding of the data.

Having discussed the implementations, the metrics, and the data sets, we now turn to the most important question: Can the proposed general approach be used to filter and correlated alerts with adequate real-world performance without investing time and resources in feature engineering and without depending on domain expertise? Our general

approach, and thereby the LSTM and RNN implementation are done without any feature engineering or domain expertise, so the question can be rephrased to: Do the evaluation results indicate that the implementations perform adequately to be of practical relevance? The factor of eliminated manual analysis effort can be expected to be in the range from 3.00 to 23.27 (Mean ARF, Table 6). As already argued, we are confident that cutting a substantial task to a third is indeed valuable and worth doing. Especially as the proposed approach provide implementations that are easy to apply compared to other methods that demand substantial investments in feature engineering and domain expertise which can make them unfeasible in practice. Incident miss rates are ranging from $7.26e{-}17$ to $9.93e-02$. In the best case, the risk is practically none. In the worst case, missing one in ten incidents might appear concerning but let us consider the alternatives. Manually processing all alerts is practically impossible for large network, and should one choose this approach it will still be subject to risk of missing incidents due to error and alert fatigue Existing methods have poor adaptability and require insurmountable investments for feature engineering and domain expertise, making them unfeasible. Even if the investments should be deemed acceptable existing methods are still not flawless, but for obvious reasons no prior work have reported their IMR for comparison. Taking a step back, the IDSs are not flawless either as is the case for many other applied security mechanisms. This is why practitioners apply the principle of layered security such that multiple mechanism can complement each other. As IDSs together with filtering and correlation methods are intended to fit in such a setting, we believe the worst IMR is acceptable.

In our future work we will explore hyper-parameter optimisation. For the LSTM RNN implementation there is a large space to explore; More layers, other RNN neurons, other activation functions, and in general any variation of the architecture. For LSA, it is also relevant to explore if the used set of N-grams and the number of topics can be modified to improve performance. In both cases, it is important to keep the overall goal of being free from feature engineering and embedded domain expertise in mind, so algorithmic hyper-parameter space exploration is most relevant. We will also focus on exploring other variations under the general approach that we have proposed and evaluated here, seeking to improve the performance(Probabilistic LSA being one option). We also find it highly relevant to apply the two implementations on other IDSs, HIDSs, and other sources that produce human-readable message, even outside the security domain. This will potentially only require data and machine resources, for learning new mapping functions, because our proposed general approach, and thereby the two implementation, are inherently de-coupled from the used IDS, and the security domain in general. As always, more extensive evaluation on more data will contribute to our understanding, thus we will pay attention to such possibilities. Experience shows that the need for raw traffic and the labelled alerts is difficult to fulfil. One possible way to overcome this is to relax the condition that the same IDS must be used for all data sets, effectively requiring raw traffic. The drawback of this is that the IDS then becomes another unknown when comparing performance on different data sets. It would also be very interesting reproduce some of the existing methods and apply them to the same data as ours, but preliminary efforts have proven that this is difficult. Curiously, this is very much in line with our reasoning that feature engineering and embedded domain expertise degrades adaptability.

VII. CONCLUSION

In this work we have described the correlation and filtering problems, which are that Intrusion Detection Systems (IDSs) raise false alerts and alerts that carry redundant information. The implication is that alerts needs to be pre-processed to be of any practical use for detection. Existing solutions apply feature engineering and embed domain expertise. We argue that this approach breaks adaptability and cause a need for substantial investments in expert time for each deployment, resulting in the approaches currently being unfeasible in practice. To overcome this, we propose a novel approach that is free of feature engineering and embedded domain expertise. In our approach a mapping function is learned from data, such that any alert can be mapped into an auto-encoded space of limited dimensions. In this space, alerts can be clustered according to the incident, removing redundant information, and filtering out false alerts. We propose a set of metrics that measure the value of applying a filtering and correlation in practice. The evaluation results show that correlation can be done without feature engineering or domain expertise embedded in the method. The key contribution is an approach that is feasible for widespread practical use and provide adequate performance, as opposed existing methods that require substantial investments for each deployment, making them unfeasible. We conclude that this method is relevant for practical purposes and will pursue further development.

REFERENCES

[1] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 635–647.

[2] J. Finkle. (Feb. 2013). *Exclusive: Microsoft and Symantec Disrupt Cyber Crime Ring*. [Online]. Available: http://www.reuters.com/article/us-cybercrime-raid-idUSBRE91515K20130206

[3] J. Demarest. (Jul. 2014). *Statement Before the Senate Judiciary Committee, Subcommittee on Crime and Terrorism: Taking Down Botnetss*. [Online]. Available: https://www.fbi.gov/news/testimony/taking-down-botnets

[4] J. Lewis. (2017). *Economic Impact of Cybercrime—No Slowing Down*. Accessed: Aug. 31, 2018. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/restricted/rp-economic-impact-cybercrime.pdf

[5] P. O'Kane, S. Sezer, and K. McLaughlin, "Obfuscation: The hidden malware," *IEEE Secur. Privacy*, vol. 9, no. 5, pp. 41–47, Sep. 2011.

[6] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through IDS-driven dialog correlation," in *Proc. 16th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2007, p. 12.

[7] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. USENIX Secur. Symp.*, 2008, pp. 139–154.

[8] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proc. 15th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2008, pp. 1–19.
[9] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proc. LISA*, 1999, pp. 229–238.
[10] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, nos. 23–24, pp. 2435–2463, Dec. 1999.
[11] V. Julien. (2015). *Suricata IDS Open Information Security Foundation*. [Online]. Available: http://suricata-ids.org/download/
[12] D. M. Chess and S. R. White, "Undetectable computer viruses," in *Proc. Virus Bull.*, 2000, pp. 107–115.
[13] R. Vaarandi and K. Podins, "Network IDS alert classification with frequent itemset mining and data clustering," in *Proc. Int. Conf. Netw. Service Manage.*, Oct. 2010, pp. 451–456.
[14] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Recent Advances in Intrusion Detection*. Berlin, Germany: Springer, 2001, pp. 54–68.
[15] B. Zhu and A. A. Ghorbani, "Alert correlation for extracting attack strategies," *J. Netw. Secur.*, vol. 3, no. 3, pp. 244–258, Nov. 2006.
[16] O. Dain and R. K. Cunningham, "Fusing a heterogeneous alert stream into scenarios," in *Proc. ACM Workshop Data Mining Secur. Appl.*, vol. 13, 2001, pp. 1–13.
[17] F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," in *Proc. IEEE Symp. Secur. Privacy*, 2002, pp. 202–215.
[18] G. C. Tjhai, S. M. Furnell, M. Papadaki, and N. L. Clarke, "A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm," *Comput. Secur.*, vol. 29, no. 6, pp. 712–723, Sep. 2010.
[19] P. Ning and D. S. Reeves, "Correlating alerts using prerequisites of intrusions: Towards reducing false alerts and uncovering high level attack strategies," DTIC, Fort Belvoir, VA, USA, Tech. Rep. 43709.13-CI, 2005. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA436839.pdf
[20] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, and R. Muttukrishnan, "OutMet: A new metric for prioritising intrusion alerts using correlation and outlier analysis," in *Proc. 39th Annu. IEEE Conf. Local Comput. Netw.*, Sep. 2014, pp. 322–330.
[21] A. Severyn and A. Moschitti, "Automatic feature engineering for answer selection and extraction," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2013, pp. 458–467.
[22] M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang, "Brainwash: A data system for feature engineering," in *Proc. CIDR*, 2013, pp. 1–4.
[23] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy, "Cognito: Automated feature engineering for supervised learning," in *Proc. IEEE 16th Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2016, pp. 1304–1307.
[24] J. Bennett and S. Lanning, "The Netflix prize," in *Proc. KDD Cup Workshop*, New York, NY, USA, 2007, p. 35.
[25] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty, "Building Watson: An overview of the DeepQA project," *AI Mag.*, vol. 31, no. 3, pp. 59–79, 2010.
[26] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, 2001, pp. 85–103.
[27] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proc. 9th ACM Conf. Comput. Commun. Secur. (CCS)*, 2002, pp. 245–254.
[28] G. P. Spathoulas and S. K. Katsikas, "Enhancing IDS performance through comprehensive alert post-processing," *Comput. Secur.*, vol. 37, pp. 176–196, Sep. 2013.
[29] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Comput. Secur.*, vol. 45, pp. 100–123, Sep. 2014.
[30] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, and M. Rajarajan, "Intrusion alert prioritisation and attack detection using post-correlation analysis," *Comput. Secur.*, vol. 50, pp. 1–15, May 2015.
[31] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, "AI^2: Training a big data machine to defend," in *Proc. IEEE 2nd Int. Conf. Big Data Secur. Cloud (BigDataSecurity), Int. Conf. High Perform. Smart Comput. (HPSC), IEEE Int. Conf. Intell. Data Secur. (IDS)*, Apr. 2016, pp. 49–54.
[32] The Shmoo Group. (2000). *Defcon 8 CTF Data Set*. [Online]. Available: http://web.archive.org/web/20080623064229/http://cctf.shmoo.com:80/data/cctf-defcon8/
[33] Cyber Security and Information Sciences Group, MIT Lincoln Laboratory. (2000). *2000 Darpa Intrusion Detection Evaluation Data Set*. [Online]. Available: https://www.ll.mit.edu/ideval/data/2000data.html
[34] (1999). *1999 Darpa Intrusion Detection Evaluation Data Set*. [Online]. Available: https://www.ll.mit.edu/ideval/data/1999data.html
[35] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012.
[36] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.
[37] E. Kidmose, M. Stevanovic, and J. M. Pedersen, "Correlating intrusion detection alerts on bot malware infections using neural network," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)*, Jun. 2016, pp. 195–211.
[38] C. M. Bishop, *Neural Networks for Pattern Recognition*. London, U.K.: Oxford Univ. Press, 1995.
[39] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural Network Design*, vol. 20. Boston, MA, USA: PWS Publishing Company, 1996.
[40] W. S. Sarle. (1997). *Neural Network FAQ, Periodic Posting to Usenet Newsgroup Compai Neural-Nets*. [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part1/preamble.html.
[41] J. Heaton, *Introduction to Neural Networks With Java*. St. Louis, MO, USA: Heaton Research, 2008.
[42] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse Process.*, vol. 25, nos. 2–3, pp. 259–284, 1998.

EGON KIDMOSE received the B.Sc.Eng. and M.Sc.Eng. degrees from Aalborg University (AAU), in 2012 and 2014, respectively, and the Ph.D. degree for work on network-based detection from AAU, in 2019. He is currently a Postdoctoral Researcher with AAU, researching detection using machine learning on domain names and DNS. During and after his Ph.D., he has experience as a Security Engineer (LEGO System A/S, during and after his Ph.D.). In addition to participating in international conferences, he also prioritizes his engagement in various relevant, local and national communities, ranging from talks at local schools to knowledge sharing with companies and authorities. His research interests include network security, incident detection, machine learning, and application of big data methods.

MATIJA STEVANOVIC received the M.Sc. degree in electrical engineering from the Faculty of Electrical Engineering, Belgrade University, in 2011, and the Ph.D. degree in electrical engineering from Aalborg University, Denmark, in 2016. He was a Postdoctoral Researcher with Aalborg University, from 2015 to 2017, and an Information Security Officer with Siemens Gamesa, Brande, Denmark, from 2017 to 2018. He is currently an IT Security Architect with LEGO System A/S, Billund, Denmark. His research interests include network security, traffic anomaly detection, and malware detection based on network traffic analysis.

SØREN BRANDBYGE received the M.Sc. degree in food science and technology from the Department of Process Technology, Royal Vetenarian High School, Copenhagen, specialized in mathematical modeling and simulation, in 1988, and the M.Sc. degree in information technology from the Center for Interactive Medias, Syddansk University, and the Department of Information and Media Studies, Aarhus University, in 2007. From 2002 to 2019, he held various security and network roles as an Infrastructure Engineer at LEGO System A/S, Billund, Denmark. He is currently an IT Security Specialist with Velux A/S, Denmark. His interests include asynchronous process communication and network security and planning with focus on cryptography and anomaly-detection in highly heterogeneous information networks.

JENS M. PEDERSEN received the M.Sc. degree in mathematics and computer science and the Ph.D. degree in Electrical Engineering from Aalborg University, Denmark, in 2002 and 2005, respectively. He is currently an Associate Professor with the Wireless Communication Section, Department of Electronic Systems, Aalborg University. He is the author/coauthor of more than 120 publications in international conferences and journals, and has participated in Danish, Nordic and European funded research projects. His research interests include network planning, traffic monitoring, and network security. He is also a board member of a number of companies within technology and innovation.