DEEPCASE: Semi-Supervised Contextual Analysis of Security Events

Thijs van Ede*†, Hojjat Aghakhani†, Noah Spahn†, Riccardo Bortolameotti‡, Marco Cova§, Andrea Continella*,
Maarten van Steen*, Andreas Peter*, Christopher Kruegel†, Giovanni Vigna†
*University of Twente, †University of California, Santa Barbara, ‡ReaQta, §VMware, Inc.
{t.s.vanede, a.continella, m.r.vansteen, a.peter}@utwente.nl, {hojjat, ncs, chris, vigna}@cs.ucsb.edu,
r.bortolameotti@reaqta.com, covam@vmware.com

***Abstract*—Security monitoring systems detect potentially malicious activities in IT infrastructures, by either looking for known signatures or for anomalous behaviors. Security operators investigate these events to determine whether they pose a threat to their organization. In many cases, a single event may be insufficient to determine whether certain activity is indeed malicious. Therefore, a security operator frequently needs to correlate multiple events to identify if they pose a real threat. Unfortunately, the vast number of events that need to be correlated often overload security operators, forcing them to ignore some events and, thereby, potentially miss attacks. This work studies how to automatically correlate security events and, thus, automate parts of the security operator workload. We design and evaluate DEEPCASE, a system that leverages the context around events to determine which events require further inspection. This approach reduces the number of events that need to be inspected. In addition, the context provides valuable insights into why certain events are classified as malicious. We show that our approach automatically filters 86.72% of the events and reduces the manual workload of security operators by 90.53%, while underestimating the risk of potential threats in less than 0.001% of cases.**

I. INTRODUCTION

Modern IT infrastructures face constant attacks and are, therefore, continuously monitored. Activities from devices and strategic points within the network are collected and processed by various systems such as Network Security Monitors (NSM, e.g., Zeek [33]) or Intrusion Detection Systems (IDS, e.g., Suricata[1] or Snort[2]). These systems contain *security event detectors* that collect information about security events (e.g., a new contacted host, the use of self-signed certificates, or ports being scanned), which they send to a central Security Operations Center (SOC). In a SOC, events are subsequently triaged by a combination of lower tier security operators and automated rules that combine events into alerts [45]. High priority alerts are then escalated to senior security operators who investigate each alert, and, depending on the threat and impact, take necessary actions [28]. Despite these filtering steps, security operators still have to manually deal with a large number of events and alerts on a daily basis.

The workload of security operators is determined by the quality of the security event detectors. These security systems are expected to flag any suspicious activity to ensure that most malicious activity is detected. However, not all suspicious events are malicious, leading to an overwhelming number of unnecessary alerts for security operators to investigate. To put this into perspective, a 2019 survey by Cisco [11] reported that 41% of 3,540 organizations examined receive over 10,000 alerts per day. Of those alerts, only 50.7% were investigated due to the limited capacity of the security operators, and only 24.1% of investigated alerts were considered an actual attack. A similar 2018 report by Demisto found that companies deal with an average of 174,000 alerts per week, of which only 12,000 were investigated [13]. Moreover, state-of-the-art academic work by Symantec Research Labs uses real-world datasets with an average of 170 security events *per device per day* [36], which shows that even for a few hundred machines, the number of security events easily becomes overwhelming. This illustrates the vast number of alerts that security operators have to deal with. This high workload leads to a condition called *alert fatigue*, where security operators fail to respond to alerts because of the sheer volume they receive each day [22].

In the literature, several works have been proposed to tackle alert fatigue. These works either focus on 1) reducing the number of generated security events by improving individual detectors [9], [24] or 2) prioritizing alerts, a method called *alert triaging* [2], [3], [22]. While reducing the number of security events per detector is useful, it might result in missing a significant portion of the alerts for malicious events. In addition, such a solution needs to be optimized for each detector. As organizations typically use different detectors from various vendors, optimizing at a detector level becomes infeasible.

In contrast, alert triaging shows promising results in reducing the workload of security operators for a more broad range of detectors. Unfortunately, existing triaging approaches still exhibit several limitations. Some works focus on prioritizing individual alerts based on threat indicators associated with each alert [2], [3]. In this case, complex attacks containing many relatively innocent security events remain undetected, while single high impact events are emphasized, despite often being produced by benign processes. More fundamentally, these approaches fail to analyze alerts in combination with other (suspicious) activities in the infrastructure. After all, threat detectors are specialized in finding suspicious behaviors, such as large file uploads or policy violations. However, we argue that suspicious behaviors can, sometimes, be legitimate when viewed together with other activity. To illustrate, consider a scenario where an attacker sends a phishing email containing a link to a website that downloads an executable infecting the machine with a packed botnet malware sample. Security detectors may raise security events for 1) a link to a website with a recently registered domain [43]; 2) a data download using a self-signed certificate [41]; 3) a packed executable [1]; and 4) beaconing activity [26]. When these events occur together, they raise suspicion; however, each individual security event could be benign. After all, these events may be caused by 1) a startup launching its new website; 2) a development webserver where the TLS certificate has not yet been properly initialized; 3) software packed for compression or to protect intellectual property [1]; 4) applications periodically checking servers for updates. It is only when we look at this

[1]https://suricata-ids.org
[2]https://www.snort.org

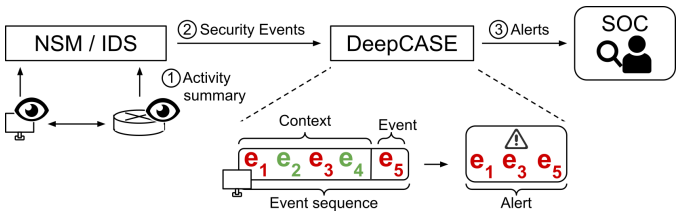

Fig. 1. **DEEPCASE setup.** 1. Agents or probes summarize activity from monitored devices and send it to a network security monitor (NSM) or intrusion detection system (IDS). 2. These NSMs or IDSs contain several detectors that identify security-relevant events (e.g., `Repeated SSH login attempt` or `Observed signature for malware X`). Normally, a security operator would investigate and deal with these events. Our work proposes DEEPCASE as an intermediate step. Here, each event is analyzed in a context, defined by the preceding events from the same device. Together they form an event sequence. Our goal is to correlate relevant events within each sequence and present them to the security operator as one alert.

sequence of events as a whole that we can be confident of an attack. Therefore, techniques prioritizing individual alerts still leave the contextual analysis of alerts up to security operators.

We propose to reduce the workload of security operators by automating the process of analyzing security events from NSM and IDS systems in combination with other triggered events. Figure 1 gives an overview of this approach. The core idea is that besides examining the security event itself, we also analyze the security events preceding it. We call these preceding security events the *context* in which an event is triggered. We refer to an event in combination with its context as an event *sequence*. Analyzing the context helps us understand what activity triggered the events. This allows us to better distinguish between an event that is benign (e.g., a self-signed TLS certificate presented by a development server) and a very similar, yet malicious, event in the context of an attack. Moreover, it allows us to present fewer and more detailed alerts to security operators, which drastically reduces their workload.

Using additional information to assess security events is not entirely new. SOC teams frequently use SIEM (security information and event management) products that include hand-crafted expert rules to combine events into alerts for known attack strategies [12]. Other systems, such as Zeek [33], even offer programmable interfaces for security operators to create these rules themselves. However, these approaches require expert knowledge and cover only event sequences that have been manually defined. Academic work, such as NoDoze [22] and later works [21], [23], attempts to automate this process by leveraging process-level data to reconstruct the activity that triggered an alert. However, this approach relies on process-level information only available in host-based detection systems. In contrast, our work focuses on placing events into context by analyzing only preceding security events. This allows our approach to have wider application, as many detectors (e.g., network-level detectors) or even entire environments (e.g., bring-your-own-device settings) do not have access to process-level information.

Contextual analysis of security events is practical only if it is able to handle 1) *complex relations* within sequences of events, triggered by 2) *evolving* threats, while 3) remaining *explainable* to security operators. Therefore, this work specifically addresses these three challenges, which make contextual analysis non-trivial.

Complex relations. Malicious activity often involves several steps that can lead to a sequence of events [31]. Conversely, benign applications may accidentally trigger security event detectors leading to false alerts. Since modern devices often run multiple benign applications simultaneously, the actual malicious behavior can easily get lost within this sequence of events. Thus, contextual analysis must be able to identify relevant context events from the complex event sequences received at the SOC.

Evolving. Both the attacks and benign applications that trigger security events evolve over time: benign applications get updated, and adversaries develop novel malware and attack techniques. Additionally, companies introduce new detectors or replace old ones to keep up with new threats or gain more insight into the suspicious activity in their IT infrastructure [15]. Ideally, a contextual analysis mechanism should be able to learn new event sequences; and adapt its detection capabilities for new attacks with minimal input from a security operator.

Explainable. Finally, reducing the workload of security operators requires filtering alerts and their corresponding events. When filtered incorrectly, we may miss attacks. Therefore, contextual analysis should provide concrete information as to why certain event sequences are discarded whilst others are marked for further investigation.

To reduce the workload of security operators while tackling all three challenges, we introduce DEEPCASE. Our approach leverages a deep learning model designed to expose complex relations between both new and previously observed event sequences. These sequences are subsequently grouped based on a similarity function, providing concrete information about why various event sequences are treated the same. Now, the operators need to inspect only a few security event sequences within each group to determine if action should be taken, which significantly reduces their workload. In summary, we make the following contributions:

- We introduce a semi-supervised deep learning approach for explaining and classifying security events in relation to their context.
- We implement this approach in a prototype called DEEPCASE, which is a deep learning system for interpreting events within their context.
- We show that DEEPCASE is able to reduce the workload of security operators by 90.53% on a real-world dataset containing all security events of 20 international organizations collected over a period of 5 months.

We make both our prototype of DEEPCASE and implementations of state-of-the-art work used in our evaluation available at https://github.com/Thijsvanede/DeepCASE.

II. SECURITY MODEL

We consider a set of machines and network endpoints that are monitored by a network security monitor (NSM), such as Zeek [33], or an intrusion detection system (IDS), such as Suricata or SnortIDS. First, these systems collect and summarize activities observed within the IT infrastructure (e.g., `HTTP Request from <SRC> to <DST> (URI: /index.html)`). When summarizing activities, these systems often already handle application-layer protocols such as SSH, HTTP, and FTP and are able to reconstruct transported files. However, these activities have no notion of being benign, malicious or suspicious, as they simply describe what happens

within the infrastructure. On top of these activities, monitoring solutions provide several detectors that detect suspicious *events*, which are sent to a central SOC where a security operator can investigate the event. These detectors may be based on signatures, policies, anomaly detection, or even customizable rules. Examples include `Malicious file download` (signature); `Unusual JA3 fingerprint` (anomaly); or `Self-signed TLS certificate on high port` (policy violation). Figure 1 gives an overview of this setup.

Our goal is to analyze all events sent to the SOC and select which events are part of an attack and should be shown to operators. Conversely, events unrelated to an attack should be filtered. After all, a security event may be caused by an adversary attacking the system or may be accidentally triggered by a benign application running on the host. We assume that detectors include information about each host involved in an event, either by leveraging an installed agent (host-based detection), or by deriving the host from the IP address and the logs of a DHCP server (network-based detection), a common assumption in enterprise networks [17].

DEEPCASE determines whether an event e_i is part of an attack given the security *context* $[e_0,...e_{i-1}]$ for this event. The context for event e_i are the n most recent events that occur on the same host as e_i, at most t seconds before e_i. If there are fewer than n events, the context is simply a shorter sequence.

III. APPROACH

We propose DEEPCASE to reduce the workload of security operators. Intuitively, our approach searches for correlations within event sequences generated for a specific device. More precisely, we are looking for correlations between events in the context of an event e_i and e_i itself that indicate whether e_i was produced by malicious activity. Once found, we cluster similar event sequences and present them to a security operator who determines whether this combination of events poses a threat to the IT infrastructure. DEEPCASE then learns this decision and automatically applies it to similar event sequences found in future event sequences. This semi-automatic approach automatically handles known correlations such that security operators can focus on new threats. Figure 2 shows the overview of DEEPCASE. First, we take sequences of security events gathered from all detectors, grouped per monitored device in chronological order. Second, for each security event the CONTEXT BUILDER searches for correlations within its context, and captures those relations in what we call an *attention vector*. We note that events from a device may be triggered by different processes or interactions, therefore, a naïve analysis of the context may not find relevant correlations. The CONTEXT BUILDER uses a deep learning model along with an attention mechanism to identify the correlation between events and their context to generate this attention vector. Subsequently, from the attention vector the CONTEXT BUILDER computes the total attention for each contextual event. Third, the INTERPRETER groups the sequences into *clusters* based on the total attention for each distinct contextual event in a sequence. These clusters can be inspected by a security operator. In manual mode, the security operator classifies each cluster by sampling and inspecting the underlying decision factors (which are provided by the attention mechanism). If the cluster is classified as malicious, the operator can take necessary action for all security sequences within the cluster and, at the same time, the system learns this new cluster. This saves time, as a security operator can assess groups of event sequences rather than individual events. Conversely, when running in semi-automatic mode, DEEPCASE compares attention vectors with previously classified clusters and automatically warns the security operator in case a sequence matches a known cluster. By filtering events from the well-known clusters, a large part of the security assessment can be automated.

A. Sequencing events

We first collect all security events generated by detectors in the monitored IT infrastructure. These are then passed to CONTEXT BUILDER, which analyzes chronological sequences of events to identify relevant contextual events and uses them to build *attention vectors*. To reduce the search space for relevant contextual events, we take each event and create a sliding window of n preceding events (in our case 10) from the same device, which form the context. To remove uncorrelated events, we limit the time difference between the event and events in its context to $t = 86{,}400$ seconds (1 day).

B. The CONTEXT BUILDER

The CONTEXT BUILDER identifies relevant contextual events to build an attention vector. Here, relevance means that our approach should identify events triggered by an attack and discriminate them from events accidentally triggered by benign applications or benign user behavior. To complicate matters, as laid out in our three challenges, the CONTEXT BUILDER should be able to deal with unpredictable, complex relations within event sequences without resorting to a black-box technique. In addition, our approach should be easily updatable to deal with evolving threats and changes in the monitored infrastructure.

To this end, we design a specific kind of recurrent neural network that uses an attention mechanism [4]. Such an attention mechanism is borrowed from the domain of natural language processing (NLP). That domain uses attention to focus a neural network on relevant parts of an input sequence with respect to the desired output [7], [14], [42]. Our work uses this attention mechanism to automatically detect which events in the context $[e_0, ..., e_{i-1}]$ are correlated with the corresponding event e_i in the sequence. Using attention has an advantage over existing state-of-the-art works that use neural networks to analyze sequences of security events [16], [36], [37]: attention can be used to compare the relevance of events in contextual security sequences, which we leverage in our INTERPRETER[3] (see Section III-C).

Figure 3 gives an overview of the network architecture of the CONTEXT BUILDER. Normally, this architecture would make a prediction of the expected event e_i by looking at only the context. However, CONTEXT BUILDER is not designed to predict, as we already know the entire sequence of events which occurred on each device. In fact, we train the CONTEXT BUILDER as if it were to predict the event e_i by looking at only the contextual events. If it is indeed able to correctly predict the event e_i, we can use the attention vector to analyze which parts of the context were relevant for this prediction. Using

[3]For a discussion regarding the use of attention as a means of explaining the relevance of events within a sequence we refer the reader to Section VI-5.

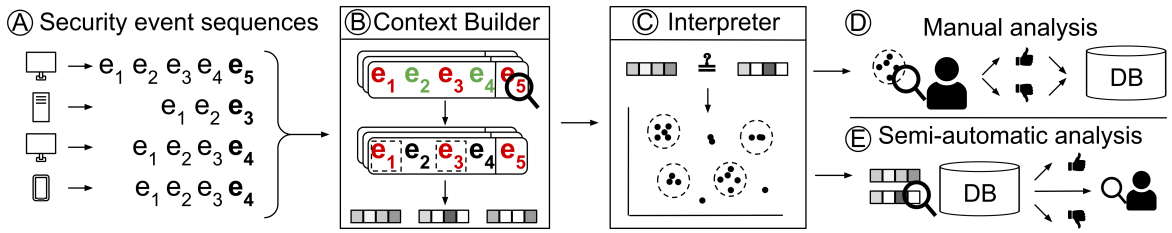

Fig. 2. **DEEPCASE overview of the contextual analysis.** (A) DEEPCASE takes sequences of security events from each device as input. (B) Next, the CONTEXT BUILDER identifies relevant contextual security events (represented by the dashed squares) and builds an *attention vector* for each sequence and uses this vector to compute the correlation of each contextual event. (C) The INTERPRETER compares all correlated contextual events and groups them into similar *clusters*. (D) A security operator analyses and labels the clusters instead of individual events, saving time. (E) Once a security operator labels clusters, similar sequences (based on the combination of contextual events and attention vector) can be automatically classified by comparing them with known clusters.

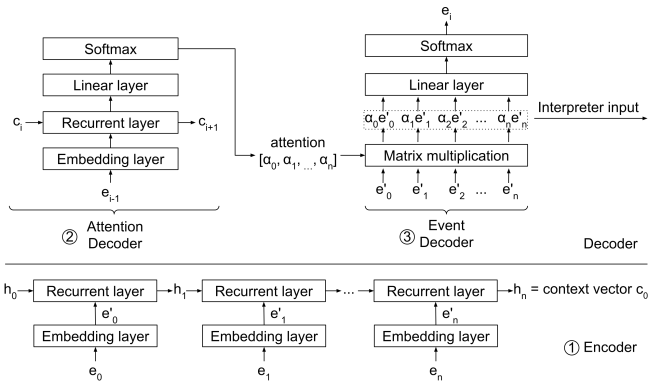

Fig. 3. **Overview of CONTEXT BUILDER's neural network architecture.** (1) The CONTEXT BUILDER embeds contextual events $S = [e_0, e_1, ..., e_n]$ into embedded vectors $[e'_0, e'_1, ..., e'_n]$. These embeddings are used to generate a context vector c_0. (2) The attention decoder then takes this context vector c_0, together with optionally generated previous outputs e_{i-1}, to learn an attention vector. (3) The event decoder distributes this attention over the embedded inputs $[e'_0, e'_1, ..., e'_n]$, which is used as input for the INTERPRETER. The modified context that follows from this is then used to predict a probability distribution of the event e_i.

such an analysis, the CONTEXT BUILDER identifies contextual events correlated with e_i in the form of an attention vector. When the CONTEXT BUILDER is unable to predict e_i, we fall back on the security operator and their existing tools to perform the analysis. This approach uses what is known as an Encoder-Decoder [8] architecture in combination with an attention mechanism [4][4]. At a high level, this means that the encoder network analyzes all contextual events and transforms them into a single vector known as the *context vector*. Next, the attention decoder takes this context vector and transforms it into an attention vector which specifies a weight (i.e., relevance) of each context event. Then, the event decoder multiplies this weight with the encoded context to obtain the total relevance for all contextual events. From here, the decoder computes a probability distribution over all possible next events. Finally, the system checks whether this "predicted" distribution matches the actual event e_i, and passes the total attention for each contextual event to the INTERPRETER. Section III-C1 describes what happens if the this prediction is incorrect. Our analysis is described in more detail below.

1) Encoder: First, the encoder takes contextual events $[e_0, e_1, ..., e_n]$ as input, which in our case is fixed to $n = 10$ and is left-padded in case there are fewer than n context events. We represent each contextual event as a vector using the embedding layer. Our work embeds the input using one-hot encoding, but embeddings may even be learned by the encoder itself [40]. Next, we use a recurrent layer to combine all inputs into an abstract, internal representation called a *context vector*, which represents the entire context as a single fixed-length vector. Note that the intermediate outputs are discarded, as we require only the final context vector for further computation. The CONTEXT BUILDER uses a single layer of Gated Recurrent Units (GRU) [8] to encode the input into a 128-dimensional context vector. Our empirical results showed that a GRU and Long Short-Term Memory (LSTM) [25] have similar performance, where the GRU is slightly faster.

2) Attention Decoder: Now that the input is encoded into the context vector, the CONTEXT BUILDER decodes this vector into an *attention vector* $[\alpha_0, \alpha_k, ..., \alpha_n]$. These α_k values represent the degree to which each corresponding context event e_k contains information regarding the security event e_i in the sequence, normalized to 1. The total attention values for each contextual event are processed by the INTERPRETER to cluster similar event sequences. In our implementation, the attention decoder takes the context vector as input, passes it through a series of linear layers, and applies a softmax function to normalize the output. Such an architecture allows us to create an attention vector for the event e_i.

To describe the context of a particular event e_i, we associate each contextual event with its corresponding attention value. To this end, we multiply the attention vector α with the one-hot encoded context sequence $S' = [e'_0, e'_1, ..., e'_n]$, i.e., the matrix multiplication in Figure 3. This results in an $n \times m$ matrix of events combined with their attention where n is the size of the context sequence and m is the size of an encoded event. To capture the events in a single vector, we sum all n rows to produce an m-dimensional vector describing the context. Recall that the attention values are normalized to 1. Hence, intuitively, this is equivalent to simply summing attention values for each contextual event in case we observe an event multiple times in the context.

3) Event Decoder: The attention decoder outputs the attention vector which, once combined with the contextual events, the INTERPRETER will later interpret and compare with the vectors of other event sequences. However, CONTEXT BUILDER must learn how to decode the context vector into an attention vector. The idea behind this learning process is simple: assuming the attention vector gives relevant context events a higher score than irrelevant events, a neural network must be able to predict the following event e_i given the context events weighed by the attention vector. Therefore, if

[4]We discuss transformers as an attention mechanism in Section VI-8.

we train the neural network of CONTEXT BUILDER to predict event e_i, we will automatically learn how to assign attention to each contextual event. To this end, the event decoder takes the embedded context events and weighs them by performing an element-wise multiplication with the attention vector. Here an attention value of 0 for a context event means that it is ignored in the event decoder, and a value of 1 means that it is the only event that will be considered. Finally, a neural network predicts the probability distribution for the event e_i given this weighted context events. This is done with a linear layer of dimension 128 with a Rectified Linear Unit (ReLU) [20] activation function and softmax function to transform the output into a probability distribution.

Processing event sequences. Before a neural network can be used for prediction tasks, such as predicting the events from their preceding context events, the network should be trained. Hence, we should show the network an event in combination with its context. In our approach, however, it is important to recall that we are not actually interested in predicting the event, as this is already known to us. Instead, we are interested in the attention vector used to make this prediction possible. This means that we can simply train the CONTEXT BUILDER with the known context as input and the known event as a prediction target. After training the CONTEXT BUILDER with multiple epochs of these inputs, we can use the same contextual events used for training to find their corresponding attention vectors. This makes generation of attention from event sequences an unsupervised process.

In order to train the network, we need to compare its generated output e_i to a desired output $\hat{e}_i$ through a loss function $L(e_i, \hat{e}_i)$. Recall that e_i is a probability distribution over all possible events. Therefore, ideally, DEEPCASE should output a probability of 1 for the actual event and 0 for all other events. This can be achieved by using the mean squared error as a loss function, or negative log likelihood when working with log probabilities. However, this incentivizes the network to produce outputs with high probabilities, even when it is not sure that the prediction is correct. To counter this effect, we use label smoothing [39]. Here the desired output $\hat{e}_i$ is a vector where the actual event has a probability value of $1-\delta$ and the remaining probability δ is scattered over the other events using their frequency distribution. In this work we use an empirically determined $\delta = 0.1$ (Section D). As e_i and $\hat{e}_i$ are now modeled as distributions, we use the Kullback-Leibler divergence [29] as a loss function for the backpropagation.

In short, the CONTEXT BUILDER exploits relations (correlations) between an event and its context to generate an attention vector that assigns a weight to each event in the context. Due to its unsupervised nature, it can easily be updated with new sequences of events [15], making it possible to deal with evolving attack patterns and IT infrastructures. Most importantly, generating the attention values for each contextual event provides the INTERPRETER with concrete information about which parts of the context are relevant to the corresponding event.

C. The INTERPRETER

After the CONTEXT BUILDER has computed the attention for each contextual event in the sequence, the INTERPRETER uses this information to compare different sequences. The idea is that event sequences with similar attention values for the same events can be treated the same way by security operators.

1) Attention query: At this point we should recall that interpreting the attention vector makes sense only if the correct event was predicted. After all, if the CONTEXT BUILDER predicts an incorrect event, interpreting attention would lead to the wrong conclusion. This means that for incorrect predictions, we would fall back on manual inspection, limiting the workload reduction for the security operator. To minimize this effect, we introduce a technique called *attention querying*.

Intuitively, this technique does not ask the CONTEXT BUILDER to predict the security event given its context, but instead asks "given the actual security event that occurred, which attention distribution *would* have resulted in the correct prediction?" We achieve this attention query by temporarily freezing the weights of the event decoder and instead making the attention vector variable. Then we use backpropagation to adjust the attention vector such that the event decoder would result in the highest prediction for the observed event. Appendix A provides a detailed description of this process and Appendix B gives a concrete example where the attention showed improved results compared to naive clustering without any use of attention.

It is still possible that even after the attention query, the CONTEXT BUILDER was not able to correctly predict the observed event. We check whether this is the case by comparing the predicted probability of e_i with a confidence threshold $\tau_{\text{confidence}}$ (see parameter selection in Appendix D-C). If the attention query achieved a sufficient confidence level, we take newly found attention as described in Section III-B2. Conversely, if the attention query was not able to pass the confidence threshold, DEEPCASE cannot deal with it and passes the sequence to a security operator for manual inspection.

2) Clusters: Now that we have modeled each sequence by combining the attention vector with their corresponding events, we can compare and group sequences with similar vectors into a cluster. To this end, we define a distance function between the vectors of attention-weighted events. Such a function allows us to search for events that occurred in a similar context. We consider two vectors similar if the distance function $d(x,y)$ is smaller or equal to the threshold $\tau_{\text{similarity}}$. The INTERPRETER defines its distance function in terms of the L_1 distance as given in Equation 1. The L_1 distance is preferable to Euclidean distance because it captures more subtle differences in high-dimensional data. In this work, the dimensionality grows with the number of different possible events m. Hence, the L_1 distance gives better results.

$$d(x,y) = \|x-y\|_1 = \sum_{i=1}^{m} |x_i - y_i| \tag{1}$$

Using this distance function, the INTERPRETER clusters event sequences using DBSCAN [18]. Here we define the maximum distance $d(x,y)$ between points to be considered part of the same cluster as $\epsilon = 0.1$ (Section D-C). Furthermore, we define the minimum required size of each cluster as 5 (Section D-C). This means that clusters containing fewer datapoints are passed directly to the security operator. Next, each cluster is either passed to a security operator for either manual analysis (Section III-D) or processed further by DEEPCASE using semi-automatic analysis (Section III-E).

D. Manual analysis

At this point, each cluster represents a set of sequences of events that share similar contexts. However, we do not yet know whether all sequences within a cluster should be considered benign or malicious, or whether a cluster contains both benign and malicious event sequences. To solve this problem, we present each cluster of similar event sequences to an operator who decides how it should be treated. Appendix F provides some examples of event clusters generated by DEEPCASE.

1) Cluster sampling: In the ideal case, all sequences within a cluster should be so similar that analyzing a single sequence is enough to determine whether all sequences are benign or malicious. However, no system is perfect and even an operator may be uncertain about certain events and their context. Therefore, we propose that security operators sample (without replacement) several event sequences from each cluster and analyze them as if they were normal alerts. Next, the operator classifies each sequence into benign vs malicious, or using different classification systems such as risk levels, e.g., `LOW`, `MEDIUM` or `HIGH`. When all sampled sequences fall into the same category and the sample size is large enough, we can confidently treat all sequences in a cluster the same way. Furthermore this classification can be stored into a database that we can use to semi-automatically classify future event sequences (Section III-E). Conversely, if sampled sequences from the same cluster fall into different categories, we know that this cluster is *ambiguous* and will need to be inspected completely by a human operator. By sampling and analyzing a small number of sequences from large clusters, the workload of security operators is drastically reduced. In Section V-B we evaluate the workload reduction and performance of this sampling process.

2) Outliers: We recall that certain event sequences are passed to a security operator because they cannot be handled by DEEPCASE. This happens in two situations.

In the first case, the CONTEXT BUILDER did not pass the $\tau_{\text{confidence}}$ threshold. Here we were unable to identify relevant contextual events and our approach does not provide additional benefit. Instead, analysis should be performed by the security operator, falling back on existing analysis tools. In case of new and unknown threats, it is only encouraged that security operators manually inspect them to ensure that no malicious activity slips under the radar. Moreover, the CONTEXT BUILDER is constantly updated with these new event sequences. This means that if they occur more regularly, the CONTEXT BUILDER eventually becomes more confident in identifying relevant contextual events in these previously unknown sequences. Therefore, over time, the number of unidentifiable event sequences will likely decrease.

In the second case, the INTERPRETER did not find enough similar sequences and manual inspection is similar to sampling from very small clusters. However, in these cases there are only a few items to sample and the resulting classification cannot be generalized to other clusters. The INTERPRETER may still store these smaller clusters such that when future similar sequences appear, the cluster can still be built incrementally. Moreover, manual analysis can still be facilitated by showing the operator clusters that are outside the similarity threshold $\tau_{\text{similarity}}$, but still have a close similarity to the scrutinized cluster. This provides security operators with more information regarding classification of somewhat-related clusters.

E. Semi-automatic analysis

Once security operators have classified clusters, the INTERPRETER can automatically compare the attention-weighted events of a new sequence (generated by the CONTEXT BUILDER) to these known clusters. If the new sequence matches a known benign cluster, we can automatically discard it without intervention of a human operator. Conversely, when a new sequence matches a known malicious cluster, DEEPCASE informs the security operator to take action[5].

However, as we have seen in the INTERPRETER (Section III-C) and manual analysis (Section III-D), some sequences do not match any cluster. This can either be because 1) CONTEXT BUILDER was unable to achieve a high-enough confidence level for the analyzed event sequence, or 2) because there are not enough other similar sequences to form a cluster. In this case, the semi-automatic analysis notifies the security operator, who evaluates the sequence as described in Section III-D2.

IV. DATASET

For our evaluations we use both a synthetic dataset for the reproduction of our results, as well as a large real-world dataset to evaluate the performance of DEEPCASE in practice.

VMWARE dataset. The real-world VMWARE dataset consists of 20 international organizations that use 395 detectors to monitor 388K devices[6]. This resulted in 10.5M *security events* for 291 unique types of security events[7] collected over a 5-month period. Events include policy violations (e.g., use of deprecated samba versions, remote desktop protocols, and the Tor browser), signature hits (e.g., `Mirai`, `Ursnif`, and `Zeus`) as well as heuristics on suspicious and malicious activity (e.g., beaconing activity, SQL injection, Shellshock Exploit Attempts and various CVEs). Of the 10.5M security events, a triaging system selected 2.7M events that were likely to be part of an attack. Of these 2.7M likely malicious events, 45.1K security events were confirmed to be part of an attack by security operators, and labeled as `ATTACKS`. These attacks include known malware, such as the `XMRig` crypto miner, or remote access Trojans, such as `NanoCore`. Another 46.4K events were classified as a `HIGH` security risk (e.g., successful web attacks and exploitation of known vulnerabilities such as `CVE-2019-19781`); 184.9K events classified as a `MEDIUM` risk (e.g., attempted binary downloads or less exploited vulnerabilities such as `CVE-2020-0601`) and 2.4M events as `LOW` risk (e.g., the use of BitTorrent or Gaming Clients). The remaining 7.8M events were not related to security risks, but were used to give security operators additional information about device activity, and are therefore labeled as `INFO`.

HDFS dataset. We also evaluate DEEPCASE on the HDFS dataset [44] used in the evaluation of the related security log

[5]In some cases it may even be possible to fully automate the response for known malicious clusters, we discuss this use of DEEPCASE as basis for SOAR systems in Section VI-3.

[6]These include devices in a bring-your-own-device setting which were only monitored for a small part of the 5 months. Therefore, the average number of 10.5M/388K = 27.06 events generated per device is significantly lower than the earlier reported 170 events per device per day.

[7]The full list is available at https://github.com/Thijsvanede/DeepCASE

analysis tool DeepLog [16]. This dataset consists of 11.2M system log entries generated by over 200 Amazon EC2 nodes. The dataset was labeled by experts into normal and anomalous events, where 2.9% of events were labeled as anomalous. Unfortunately, this dataset lacks metadata about the risk level of security events and is therefore evaluated in terms of workload reduction, but not in terms of accuracy. Despite containing less information, we use the HDFS dataset to provide a reproducible comparison with state-of-the-art systems.

V. Evaluation

We implemented DeepCASE in Python and compare its performance in workload reduction and performance metrics (e.g., precision, recall, F1-score) with existing workload reduction techniques. Table I gives an overview of the average performance achieved by all compared methods. Additionally, we evaluate how well DeepCASE deals with the three challenges proposed in the introduction, discuss its robustness against evasion strategies, and perform a runtime analysis to show that DeepCASE is able to handle real-world events generated by major organizations.

A. Setup

To evaluate DeepCASE in a realistic scenario, we split our dataset in a part used to perform manual mode analysis and a part for the semi-automatic mode. The manual mode always precedes the semi-automatic mode, and, therefore, we use the first month of data (2M events) in the VMware dataset to evaluate our manual mode and the subsequent months to evaluate the semi-automatic mode. The HDFS dataset was split by the original work into training and test sets, which we use in manual mode and semi-automatic mode, respectively. We run all our experiments using the same parameters, which we obtained during a parameter optimization experiment (Appendix D). Unless otherwise specified, we report the average results of 10 runs for each experiment. We followed the three research guidelines for evaluating machine learning-based security systems as detailed in TESSERACT [34]:

- Our experiments have a *temporal training consistency*, meaning that data for our manual evaluation comes strictly before the data used in semi-automatic mode.
- Data should be collected over a *consistent time window*, i.e., there should be no major gaps between collection of data. Our VMware dataset was collected over a continuous period of 5 months ensuring time consistency.
- There is a *realistic malware-to-goodware ratio in testing*. This ratio follows from the use of a real-world dataset consisting of events collected from 20 organizations.

During the manual mode, the Context Builder learns to produce attention vectors by training the neural network for 100 epochs and extracting the final attention vector (see Section III-D for more details). In semi-automatic mode, this training is only performed when updating the Context Builder (Section V-C2).

B. Workload reduction

In this section we compare the workload reduction of our approach with existing techniques used by real-world SIEM systems. These techniques include alert throttling and expert rules, as well as more naive, automated methods such as n-gram analysis and our own clustering approach without use of the Context Builder. For each technique, we measure the workload reduction in terms of the percentage of events covered by the raised alerts (coverage), the number of produced alerts compared to these covered events (reduction), and the overall reduction in inspected events (alerts + events not covered) compared to the total number of events analyzed. Furthermore, we discuss the performance over covered events in terms of precision, recall, F1-score, accuracy and percentage of events for which the algorithm underestimated the risk level. Table I shows the results of all experiments. The remainder of this section discusses how each result was obtained.

DeepCASE- Manual Mode

When used in practice, DeepCASE starts without any knowledge of event sequences. At this stage, DeepCASE runs in manual analysis mode (Section III-D), where all sequences are processed to produce clusters of event sequences (similar events occuring within a similar context). These clusters are then shown to the security operator, who determines if the sequences in each cluster are benign or malicious. In this setting, the workload is reduced because an operator does not have to investigate each individual event; instead, only a small number of samples from each cluster, and the sequences that could not be handled by DeepCASE. We simulated this *manual mode* scenario using the first month of the VMware events and the training data of the HDFS dataset.

1) Coverage: Table I shows that for the VMware dataset, 94.46% of the 2M training event sequences could be grouped into 1,642 clusters. The remaining 5.54% (110.9K) event sequences could not be turned into clusters, either because DeepCASE was not confident enough (95.70%, 106.1K cases) or because there were fewer than 5 other sequences with a similar context (4.30%, 4.8K cases). These remaining sequences can be manually inspected or filtered through existing triage systems, which are complementary to our approach. We performed the same evaluation on the HDFS dataset (see Appendix E). On this dataset, we found similar results where DeepCASE covers 96.39% of sequences with 393 clusters, leading to an overall reduction of 92.26%.

2) Cluster classification: Each cluster contains security events with a similar context. However, this grouping would be useful for the security operator only if each sequence within a cluster is treated the same way. If a cluster contained both benign and malicious samples, or different risk levels, our approach would have a limited benefit. Thus, we scrutinized all event sequences produced by DeepCASE to evaluate to what extent the sequences in each cluster have the same risk classification. We recall from Section IV that the VMware dataset is labeled into 5 risk categories: `INFO`, `LOW`, `MEDIUM`, `HIGH` and `ATTACK`. Table II gives an overview of the classification of the clusters. Each risk level details the number of clusters that contain only contextual sequences of that single risk level as well as some statistics about the number of sequences per cluster. Not all clusters contain security sequences of a single risk level. Therefore, we also have a `SUSPICIOUS` category which captures the clusters containing multiple risk levels. As we can see from Table II, 1,404 of the 1,642 clusters contain sequences of only a single risk level, corresponding to 67.05% of clustered

TABLE I

WORKLOAD REDUCTION. AVERAGE WORKLOAD REDUCTION OF DEEPCASE COMPARED WITH EXISTING WORKLOAD REDUCTION METHODS. WE HIGHLIGHT THE *Overall* AND *Underest.* COLUMNS. *Overall* SHOWS THE TOTAL WORKLOAD REDUCTION OF SECURITY OPERATORS. *Underest.* SHOWS HOW MANY OF THE COVERED EVENTS ARE ASSIGNED A RISK LEVEL LOWER THAN THEIR TRUE RISK LEVEL, WHICH POTENTIALLY LEADS TO MISSED ATTACKS.

		Workload reduction				Performance over covered events				
	Method	**Alerts**[A]	**Reduction**[B]	**Coverage**[C]	**Overall**[D]	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Underest.**
Manual	DEEPCASE	16,420	99.13%	94.46%	93.64%	N/A	N/A	N/A	N/A	N/A
	Cluster N-gram	35,640	98.12%	94.62%	92.83%	N/A	N/A	N/A	N/A	N/A
	Cluster DEEPCASE	45,400	97.68%	97.96%	**95.69%**	N/A	N/A	N/A	N/A	N/A
Semi-automatic	DEEPCASE	51,800	99.19%	91.27%	**90.53%**	96.39%	91.47%	93.41%	91.47%	**<0.01%**
	fully-automatic part	N/A	100.00%	86.72%	86.72%	96.39%	91.47%	93.41%	91.47%	**<0.01%**
	manual part	51,800	83.83%	34.29%	28.74%	N/A	N/A	N/A	N/A	N/A
	Alert throttling (15 min)	3,532,849	49.77%	100.00%	49.77%	98.08%	98.04%	98.04%	98.04%	0.79%
	Alert throttling (30 min)	2,889,607	58.92%	100.00%	58.92%	97.92%	97.90%	97.90%	97.90%	0.97%
	Alert throttling (60 min)	2,332,467	66.84%	100.00%	66.84%	97.83%	97.83%	97.83%	97.83%	1.11%
	Alert throttling (1 day)	855,798	87.83%	100.00%	87.83%	97.47%	97.49%	97.49%	97.47%	1.34%
	Rules AlienVault[E]	421,693	83.78%	36.97%	30.97%	99.64%	99.63%	99.63%	99.63%	0.16%
	Rules VMWARE [F]	299,246	89.49%	27.02%	24.18%	100.00%[F]	100.00%[F]	100.00%[F]	100.00%[F]	**0.00%**[F]
	Rules Sigma/Zeek[E]	126,147	92.87%	25.14%	23.35%	99.55%	99.51%	99.52%	99.51%	0.17%
	Cluster N-gram	N/A	100.00%	75.70%	75.70%	96.11%	94.00%	94.59%	94.00%	0.01%
	Cluster DEEPCASE	N/A	100.00%	80.59%	80.59%	95.77%	91.25%	92.80%	91.25%	0.01%

A Number of alerts sent to security operator. For DEEPCASE and cluster, this is based on 10 sequences per cluster.

B Computed as the fraction between alerts and covered events (see Formulas).

C Percentage of events covered by alerts (see Formulas).

D Total reduction, alerts + uncovered events compared to total alerts (see Formulas).

E Based on the event translations provided at https://github.com/Thijsvanede/DeepCASE.

F These rules were used in creating the ground truth (selected events were always shown to analysts) and may therefore give an overly optimistic performance.

Formulas

B $1 - \frac{\text{Alerts}}{\text{Covered events}}$

C $\frac{\text{Covered events}}{\text{Total events}}$

D $1 - \frac{\text{Alerts} + \text{Uncovered events}}{\text{Total events}}$

TABLE II

CLUSTERS - MANUAL MODE. CLUSTERS PER RISK LEVEL. SUSPICIOUS CLUSTERS CONTAIN CONTEXT SEQUENCES WITH MULTIPLE RISK LEVELS.

		# Sequences				
Risk level	**Clusters**	**Total**	**Average**	**Min**	**Max**	**σ (SD)**
INFO	1,115	1.216M	1090.3	5	583.9K	19.2K
LOW	221	41.8K	189.4	5	5,557	612.9
MEDIUM	18	568	31.6	5	235	55.5
HIGH	17	1989	117.0	6	1,107	270.6
ATTACK	33	1391	42.2	5	402	77.1
SUSPICIOUS	238	619.8K	2604.4	5	280.1K	20.2K
Total	1,642	1.881M	1145.7	5	583.9K	17.6K

sequences. For clusters containing two adjacent risk levels (e.g., LOW and MEDIUM or HIGH and ATTACK), we find 1,527 of the 1,642 cluster, corresponding to 98.56% of all clustered sequences. Additionally, the maximum cluster sizes and standard deviation values are large. This is because there are many smaller clusters and only a few large ones, i.e., clusters are skewed toward the lower end. Appendix G gives an overview of the cluster size distribution. To measure the extent to which clustered samples belong to the same class - or in our case risk level - we use the conventional homogeneity score [35]. This score measures the decrease in entropy of a sample class when the cluster is known. The homogeneity is 0 if all clusters contain multiple different risk levels and 1 for the ideal case where all clusters contain only samples of the same risk level. Our clusters show a high quality grouping of sequences of the same risk with a homogeneity score of 0.98.

3) Sampling: In the real-world scenario, when a cluster is presented to the security operator, they do not know whether a cluster contains only sequences of a single risk level or multiple risk levels. This is important information as the operator should be able to rely on DEEPCASE to group threats with a similar level of risk in the same cluster. During manual mode, the risk level of each cluster must be determined by the operator. To this end, we suggest that the operator samples several event sequences to determine the cluster's risk level. In the ideal case where each cluster has a one-to-one mapping with the risk levels, a security operator would only need to sample a single event sequence per cluster. However, as we have seen in Section V-B2, this is not the case. Nevertheless, we can measure the number of investigated samples required for an arbitrary confidence in the risk level of a cluster.

A SUSPICIOUS cluster contains contextual sequences of multiple risk levels. From Table II we found that 14.5% of clusters are SUSPICIOUS. Given the sequences within a cluster, we can compute the probability of identifying a cluster as SUSPICIOUS. We define this as the probability of drawing event sequences of at least two different risk levels, i.e., 1 minus the probability of drawing sequences of only a single risk level. Equation 2 gives the probability of detecting a suspicious cluster when sampling k different security sequences. Here N is the total number of event sequences within the cluster, C is the set of event sequences from all risk levels and c specifies the set of sequences for each risk level. Note that we model this as sampling without replacement as a security operator will not choose two of the same event sequences to classify. This probability will always be 0 for non-SUSPICIOUS clusters.

$$P(\texttt{suspicious}|k) = 1 - \sum_{c \in C} \frac{\binom{|c|}{k}}{\binom{N}{k}} \tag{2}$$

To adopt a conservative approach, a security operator can label a cluster by the highest risk level they have identified from sampling. This way, DEEPCASE will miss fewer security threats at the cost of a slightly larger number of event

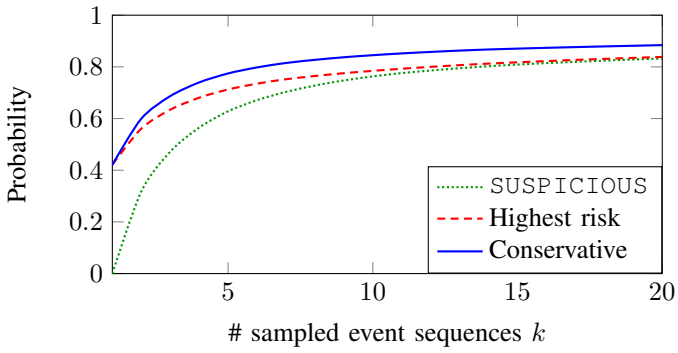

Fig. 4. **Probability of correctly identifying `SUSPICIOUS` clusters.** Shows 1) the average probability of identifying `SUSPICIOUS` clusters (·····); 2) the average probability of finding the highest risk event sequence (- - - -); and 3) the probability of detecting a cluster as `SUSPICIOUS`, or if not `SUSPICIOUS`, labeling it as the highest risk sequence (——), i.e. a conservative clustering approach.

sequences that an operator should investigate. Equation 3 gives the probability of sampling at least one event sequence of the highest risk level given a `SUSPICIOUS` cluster. Here, $|C \setminus h|$ is the number of event sequences from each risk level except the highest risk level h.

$$P(\text{highest risk}|k) = 1 - \frac{\binom{|C\setminus h|}{k}}{\binom{N}{k}} \quad (3)$$

From Figure 4, we find that by sampling 10 samples per cluster, the conservative approach gives a 84.52% confidence for labeling `SUSPICIOUS` clusters. If a higher confidence is required for the conservative approach, we found that inspecting 95 samples gives a confidence of 95% and inspecting 262 samples gives 99% confidence. However, as only 14.5% of clusters are `SUSPICIOUS`, inspecting 10 samples per cluster corresponds to a 97.76% overall confidence rating for all clusters.

4) Workload reduction: In short, when running DEEPCASE on the VMWARE dataset, inspecting 16.4K (10 sequences for 1,642 clusters) event sequences is enough to cover 94.46% (Section V-B1) of all security events with a clustering confidence of 97.76%. To cover 100%, i.e., also cover all outliers not handled by DEEPCASE, an operator should inspect 127.3K (16.4K clustered + 110.8K outliers) out of 2M events. This reduces the total workload of security operators by 93.64%. For the HDFS dataset, this reduction is slightly smaller, reducing the workload by 92.26%.

DEEPCASE- Semi-automatic Mode.

After a security operator labeled clusters in the manual mode, DEEPCASE can be run in semi-automatic mode. During this phase, upcoming sequences are compared against labeled clusters. In case the attention-weighted events match, the sequence of events is automatically labeled according to the matching cluster. Depending on the policy of an organization, event sequences labeled higher than a given risk level are escalated to the security operator who can then remove the threat. If the event sequence is found to be benign, it is filtered and not shown to the security operator. Some attention-weighted events will not match any of the known clusters, and, as a result, they will either form new clusters or are outliers that will be passed to the security operator for manual inspection. In those cases, the security operator will deal with the sequences as if operating in manual mode, as described in Section III-D.

Table I shows the performance of DEEPCASE on the VMWARE dataset running in semi-automatic mode after operating for one month in manual mode. Here, we see that 86.72% of all event sequences match a known cluster, and can be automatically classified. The remaining 13.28% of event sequences is processed in manual mode. From Table I we see that 34.29% of these non-matching sequences formed new clusters. After this manual step, 8.73% of all sequences could not be clustered, either because they did not pass the $\tau_{\text{confidence}}$ threshold (97.91% of cases) or because there were fewer than 5 samples in a cluster (2.09% of cases). In addition, the automatic classification of risk levels gives us a reasonable accuracy and F1-score of 91.47% and 93.41%, respectively. However, we must be careful with such numbers, as misclassifying an event sequence with a lower risk means missing attacks. Conversely, misclassifying event sequences as a higher risk is less problematic, and would only give the security operator more manual work. Despite our goal to reduce the workload of security operators, we rather overestimate the risk level at the cost of a smaller workload reduction than miss attacks. The confusion matrix of Table III shows that the majority of incorrectly labeled cases overestimate the risk level. In fact, DEEPCASE underestimates only 47 sequences, which is less than 0.001% of cases.

To understand why DEEPCASE underestimates 47 of its semi-automatic predictions, we look at some underestimated cases. Of these underestimates, 3 LOW and 35 MEDIUM risks were classified as INFO, 4 HIGH risks were classified as MEDIUM and 8 ATTACK levels as HIGH risks. The sequences misclassified as INFO are the most undesirable, as these will be ignored altogether by analysts. These sequences were part of 4 different clusters: notably the first detection of the `Bladabindi` backdoor without any prior events was misclassified as `INFO` as well as several unsuccessful `web-application attacks`. However, as detections are often only a single step of an attack, we investigated whether all parts of the attack misclassified. Here we found that for *all* `ATTACKS` misclassified as `INFO`, at least one earlier or later step of the same attack was classified as `ATTACK`. Other underestimated predictions were less severe and are still shown to analysts, e.g., when a `HIGH` risk was predicted to be `MEDIUM`. In these cases, the incorrect classification is mostly due to similar event sequences observed for different machines. Here the sequences for clusters analyzed in manual mode occurred on machines that were not of vital importance for business continuity. Section VI-4 explores the adjustment of risk level depending on the importance of machines.

State-of-the-art alert reduction techniques

Instead of raising an alert for each event, many SIEM tools provide options to throttle events as well as options for defining expert rules that aggregate sequences of events into an individual alert. Here, we compare the performance of both methods from state-of-the-art tools with that of DEEPCASE.

1) Alert throttling: With alert throttling, if an event is triggered multiple times over a given period, only a single alert is shown to an operator, usually in an aggregated form. E.g., suppose the same event X is raised 5 times within the

TABLE III
PERFORMANCE - SEMI-AUTOMATIC MODE. THE TOP ROW SHOWS THE CLASSIFICATION PERFORMANCE AND COVERAGE OF SEMI-AUTOMATIC ANALYSIS. THE BOTTOM SHOWS THE CONFUSION MATRIX OF AUTOMATICALLY CLASSIFIED SAMPLES IN THE VMWARE DATASET.

Confusion matrix		Predicted				
		INFO	LOW	MEDIUM	HIGH	ATTACK
Actual	INFO	4896683	281528	90025	132381	165
	LOW	3	663327	303	1	1
	MEDIUM	32	0	3014	14806	788
	HIGH	0	0	4	3419	23
	ATTACK	0	0	0	8	12870

throttling period, the security operator will receive only a single alert after the first event triggered. All subsequent events X within the throttling period are added only as additional information to the first generated alert. After the throttling period passed, a new event will generate a new alert.

We run this throttling mechanism over our VMWARE dataset for various throttling periods ranging from 15 minutes to 1 day. Table I shows the results for this experiment.

The disadvantage of such an approach is that an analyst either has to wait the full throttling period to make a definitive risk assessment of all events within an alert; or she has to assess the risk without having received all events, which potentially misses attacks. The performance metrics for alert throttling show the results after assessing all throttled events with the most common observed risk level.

2) Expert rules: Instead of alert throttling, companies often have expert-crafted rules to combine multiple events into a single alert. Sometimes this functionality is embedded into NSM or IDS software. A notable example is Zeek [33], which offers the Zeek Notice Framework, where analysts can write rules that search through Zeek logs and get notified of matches. Such rules can be based on specific sequences or combinations of events related to known common attacks. Depending on the rules, sequences of events that have only a partial match will still be triggered, but often with a lower reliability level.

In this work, we compare the performance of DEEPCASE using the VMWARE dataset with expert rules from the VMWARE; 292 open source rules from AlienVault's OSSIM[8] that cover 82 different known attacks; and the open source rules from Sigma[9] that cover known attacks from various resources such as the MITRE ATT&CK framework. Sigma includes rules that specifically cover Zeek logs and thereby gives a publically available alternative to Zeek Notices, which are normally specifically written for an organization. As many of the rules from Sigma and AlienVault's OSSIM operate on specific types of detection events, we manually created a bidirectional mapping between all 591 different events used in these rules and the types of events in our COMPANY dataset[10]. This allows us to directly apply the Sigma and OSSIM rules to the events in the COMPANY dataset and compare their achieved alert reduction to the results from DEEPCASE. We have been liberal with this mapping, meaning that any event that could match those provided by AlienVault or Sigma is counted as such. This results in an optimistic coverage of both rulesets and it explains why the coverage of AlienVault is higher than the coverage achieved by the VMWARE ruleset. We note that there are many other solutions that provide rule-based detection such as Azure Sentinel [11] and Splunk [12]. For these solutions, we were unable to obtain publically available rules, and thus could not perform a comparison.

For our evaluation, we count the number of alerts triggered by these expert rules. One single alert consists of all events of a machine (partially) matching one of the available rules. Table I shows the results of the expert rules. Here, we see that while the reduction for the covered events is similar to DEEPCASE, the number of events that are covered is significantly lower. Furthermore, the performance for all events coverd by the rulesets is near perfect, with metrics being over 99.51%. This shows that expert rules are highly effective for detecting threats, but still lack much of the coverage that DEEPCASE provides. DEEPCASE tackles this problem by automatically finding correlations between events, thereby vastly increasing the coverage. Hence, DEEPCASE shows much potential to be used in combination with expert-rules for combining events into alerts.

Naive clustering

DEEPCASE's CONTEXT BUILDER detects correlations between an event and its context. To demonstrate why this component is required, we perform an ablation study, i.e., we compare our full approach with a version of DEEPCASE without the CONTEXT BUILDER. Additionally, we show the performance increase of our clustering approach compared to clustering on exact matches, i.e. N-grams.

1) N-grams: The most straightforward approach for sequence prediction is to treat an event and its context as an N-gram. Here we can store all N-grams in the training data, together with their highest associated risk level and assign the same risk level to N-grams in the test data if there is a match.

Table I shows the result for this experiment, where each alert is equal to the number of stored N-grams. As with DEEPCASE, we assume a security operator will check 10 sequences (N-grams) to determine its risk level. While this approach does not underperform compared to DEEPCASE in the manual use case, the semi-automatic use case shows that N-grams do not generalize as well.

2) Clustering: Instead of using N-grams, we can approach matching better using clustering as proposed by DEEPCASE, without using the CONTEXT BUILDER. This scenario is equivalent to DEEPCASE with CONTEXT BUILDER where the attention value for each contextual event is $\frac{1}{n}$.

Table I shows that the results for using only our clustering approach slightly outperforms DEEPCASE in manual mode. However, in semi-automatic mode, it still produces twice the workload compared to DEEPCASE with the CONTEXT BUILDER. Therefore, we conclude that the CONTEXT BUILDER generalizes significantly more than naive approaches. This results in DEEPCASE roughly halving the workload of a security operator in semi-automatic mode compared to naive clustering.

[8]https://cybersecurity.att.com/products/ossim

[9]https://github.com/SigmaHQ/sigma

[10]Mappings are available at https://github.com/Thijsvanede/DeepCASE

[11]https://azure.microsoft.com/en-us/services/azure-sentinel/

[12]https://www.splunk.com/

TABLE IV

PREDICTION RESULTS. SYSTEMS TRAINED ON FIRST 20% OF DATA AND EVALUATED ON REMAINING 80% OF DATA. TIME SHOWS THE AVERAGE AMOUNT OF TIME FOR 1 EPOCH OF TRAINING. BEST PERFORMANCE IS HIGHLIGHTED IN **BOLD**.

	System	Precision	Recall	F1-score	Accuracy	Train time
HDFS	DeepLog	89.71%	89.34%	89.35%	89.34%	**1.0 s**
	Tiresias	89.70%	87.63%	87.96%	87.63%	15.0 s
	DEEPCASE	**90.41%**	**90.64%**	**90.40%**	**90.64%**	1.3 s
VMWARE	DeepLog	89.65%	90.40%	89.82%	90.40%	**0:06.8 m**
	Tiresias	95.50%	96.21%	95.68%	96.21%	4:51.5 m
	DEEPCASE	**97.90%**	**98.06%**	**97.90%**	**98.06%**	0:13.8 m

C. Challenges

This work addressed three challenges that make reducing the workload of security operators difficult. Here, we evaluate the extent to which our approach solves these challenges.

1) Complex relations: The CONTEXT BUILDER was designed to find correlations between an event and its context. To this end, it analyzes the preceding contextual events and tries to predict what event is most likely to occur next. To understand how well our approach deals with such complex relations, we assess to what extent events are correctly predicted from their context. This prediction based on past security events is not novel, as it was introduced by DeepLog [16] and later extended by Tiresias [36]. However, both of these works focus purely on the prediction aspect, and cannot be extended to perform the contextual analyses proposed in this work. Therefore, to measure how well DEEPCASE is able to deal with complex relations within the data, we compare its prediction performance with the two state-of-the-art systems DeepLog and Tiresias. We implemented both systems as described in their respective papers. While the original source code was not available, upon contacting the authors we received helpful suggestions to re-implement both approaches[13][14].

We evaluate all three approaches on both datasets described in Section IV. We performed 10 training and testing runs for each system, where the first 20% of the datasets were used for training and the remaining 80% was used for testing. Table IV shows the average results of all 10 runs of this evaluation. We see that on both datasets, DEEPCASE performs the best in terms of evaluation metrics. The only downside of DEEPCASE is that the runtime is slightly slower than DeepLog, but as we show in Appendix H, DEEPCASE is easily fast enough for real-world application. In short, DEEPCASE shows improvements over state-of-the-art works that were specifically designed to predict future events, while handling their complex relations.

2) Evolving event patterns: Our second challenge is dealing with evolving threats and event patterns. DEEPCASE should be able to detect new event sequences and group them into new clusters to show to the security operator. In addition, DEEPCASE uses a neural network in the CONTEXT BUILDER to model sequences. We generally expect the results to improve as we show the network an increasingly large number of event sequences. We recall that these sequences are automatically generated, and therefore do not need to be labeled. When monitoring a new IT infrastructure or handling events of new detectors, DEEPCASE should quickly be able to update using these new incoming events.

[13]DeepLog code is available at https://github.com/Thijsvanede/DeepLog

[14]Tiresias code is available at https://github.com/Thijsvanede/Tiresias.

TABLE V

COMPARISON BETWEEN UPDATING STRATEGIES. WE SHOW THE INCREASED NUMBER OF COVERED EVENTS WHEN DEEPCASE IS UPDATED AND ITS PERFORMANCE METRICS. WE COMPARE DEEPCASE OVER TIME WHEN NOT UPDATED WITH DAILY, WEEKLY AND MONTHLY UPDATING.

		Metrics over covered data			
Updates	Coverage	Precision	Recall	F1-score	Accuracy
None	87.38%	96.19%	92.75%	93.67%	92.75%
Monthly	(+411K) 93.29%	95.64%	92.16%	93.16%	92.16%
Weekly	(+450K) 93.93%	94.84%	90.92%	92.30%	90.92%
Daily	(+466K) 94.23%	95.12%	91.39%	92.68%	91.39%

Online updating. We demonstrate DEEPCASE's ability to evolve with new events. Operating in manual mode, our approach already is able to deal with new events and different event patterns. Therefore, we evaluate the increase in performance of DEEPCASE running in semi-automatic mode if periodically updated. In this experiment, we compare the performance between no updates and daily, weekly and monthly updates. In each update, we show new data to the CONTEXT BUILDER and add newly produced and manually labeled clusters to the database of the INTERPRETER to be used for comparing future event sequences.

Table V shows the results for this experiment. We find that regularly updating DEEPCASE improves its coverage between 5.91 and 6.85 percentage points. The performance in terms of accurately predicting the risk level of these newly covered items is only marginally lower than the original detection performance. This is mostly due to having fewer datapoints available for accurate classification. Interestingly, we found that the improvements in coverage came from newly added clusters to the database rather than improved confidence of the CONTEXT BUILDER. In fact, in some cases the CONTEXT BUILDER became less confident, especially when having to learn to classify new events. To illustrate this, consider a context X that is used to predict an event e_{old}. Now consider a newly observed event e_{new} with the same context X. When the CONTEXT BUILDER is updated, it is taught to lower its confidence for X to predict e_{old} as there is now also the option to predict e_{new}. Nevertheless, this confidence reduction allows us to deal with new events. Moreover, the lower confidence is more than made up for by the additional coverage resulting from updating the INTERPRETER.

3) Explainable clusters: In manual mode, DEEPCASE produces clusters that security operators inspect and classify. We have shown in Section V-B3 that it is often enough for operators to sample a few sequences from each cluster to determine its risk level. In this section, we evaluate to what extent our attention query improves the explainability of event sequences. Appendix F highlights some characteristics of individual clusters.

Attention query. One way in which DEEPCASE improves its explanation of security events is through the attention query. This query increases the confidence of the CONTEXT BUILDER in the actual security event that occurred by shifting the attention to the more relevant contextual events. In this experiment, we measured the increase in confidence for the actual event using this attention query. Again, we trained

the CONTEXT BUILDER on the first month of data from the VMWARE dataset as in all other experiments. Next, we predicted security events from their context and measured the confidence level in the actual event that occurred with and without applying the attention query. Without the attention query, the CONTEXT BUILDER reached the confidence threshold of 0.2 in 86.11% of cases. Conversely, applying the attention query resulted in a confidence ≥ 0.2 in 92.21% of cases. This shows that the attention query improves the coverage for explainable events by 6.10%. For the HDFS dataset, this number increased from 94.66% to 99.24%, an improvement of 4.58 percentage points.

D. Robustness

Using DEEPCASE to reduce the workload of security operators may also create an additional attack surface for adversaries. After all, if our approach allows adversaries to maliciously craft attacks such that DEEPCASE discards them as being completely benign, reducing the workload would not serve its purpose. Moreover, our approach itself may be targeted by an attacker to annul the workload reduction for security operators. As our approach relies on events produced by security event detectors, we consider attacks that bypass or alter detector outputs to be outside the scope of this research. Therefore, we discuss and evaluate to what extent an adversary can manipulate DEEPCASE itself.

Denial of service attack

First, an attacker can perform a denial of service (DoS) attack against DEEPCASE. Here, we do not focus on (D)DoS attacks against a monitored device, as this type of attack will simply generate a single cluster. Instead, we discuss DoS attacks with the purpose of letting DEEPCASE generate so many new clusters or sequences that cannot be handled by our approach, overloading security operators. In Appendix H, we show that our approach is able to process more than 10 K sequences per second on a system that has a good graphics card. Therefore, we can safely assume that the bottleneck for DoS attacks is the number of sequences that the human operator has to manually inspect. While DEEPCASE currently does not provide countermeasures for this type of attack, a sudden surge of new sequences to inspect would cause suspicion for a security operator, even if the individual sequences do not seem malicious. In fact, we found that in the VMWARE dataset, for each device monitored in semi-automatic mode, DEEPCASE triggers a new sequence to inspect only once every 2.5 days. A single device producing many more events than the average (in the VMWARE dataset, the worst infected machine produced 607 unknown sequences in a single day) is likely to be thoroughly scrutinized by security operators. We discuss DEEPCASE's limitations with respect to DoS attacks in Section VI-2.

Evasion attack

Second, an attacker may purposely trigger additional security events to change the context of the attack events. Thereby, the attacker either 1) changes the context sequence of an ongoing attack such that it matches a benign cluster instead of a malicious cluster; or 2) attempts to create a new and benign cluster that can later be used to perform an attack.

TABLE VI
PERFORMANCE OF DEEPCASE UNDER EVASION ATTACK. DEEPCASE BECOMES LESS CONFIDENT ABOUT EVENT SEQUENCES AND SENDS THESE OUTLIER SEQUENCES TO SECURITY OPERATORS. THE REMAINING SEQUENCES, FOR WHICH DEEPCASE REMAINS CONFIDENT, PERFORM SIMILAR TO WHEN NO INJECTION TAKES PLACE.

		Metrics on non-outlier data			
Injected	**Outliers**	**Precision**	**Recall**	**F1-score**	**Accuracy**
0/10	19.92%	98.62%	96.43%	96.88%	96.43%
1/10	21.22%	98.88%	97.26%	97.69%	97.26%
2/10	22.86%	99.06%	97.77%	98.17%	97.77%
3/10	25.51%	99.19%	98.14%	98.51%	98.14%
4/10	29.67%	99.29%	98.40%	98.75%	98.40%
5/10	35.55%	99.39%	98.61%	98.94%	98.61%
6/10	42.66%	99.46%	98.78%	99.08%	98.78%
7/10	50.26%	99.52%	98.92%	99.20%	98.92%
8/10	57.88%	99.57%	99.05%	99.30%	99.05%
9/10	68.94%	99.57%	99.16%	99.36%	99.16%
10/10	98.01%	97.92%	96.03%	96.50%	96.03%

1) Detecting evasive attacks: DEEPCASE processes event sequences of an evading attacker in one of three ways: 1) the attack is still classified as an attack; 2) the attack is classified as benign (either by hiding in a benign cluster in manual mode, or by being assigned an `INFO` risk cluster in semi-automatic mode); or 3) the attack is considered an outlier and passed to a security operator. We consider case 2 the only successful outcome for an attacker, as both cases 1 and 3 will be shown to a security operator. In case an attacker tries to create a new benign cluster, such a new cluster will always be passed to a security operator, as non-matching contextual sequences are labeled as outliers. Therefore, in this experiment we evaluate to what extent an attacker can change the context of an event from a malicious to a benign cluster without becoming an outlier, which would be shown to the security operator.

2) Evaluation: We simulated an evasion attack by inserting random security events in the context of events from the VMWARE dataset. We note that an attacker with insider knowledge of DEEPCASE can perform an evasion attack by *selecting* specific security events that perform better than random events. We discuss this scenario in Section VI-2, where we show that having specific knowledge of DEEPCASE's clusters will be better than inserting random events possible for only 6.32% of clusters. For the random security events, we looked only at event sequences with a risk level of `LOW` or higher, i.e., `INFO` risk levels were omitted as they did not contain attacks. In this experiment we inserted random events into the context ranging from 0% to 100% percent of the context size in steps of 10%. Where 0% is the original context and injecting 100% completely altered the context. Next, we measured the number of sequences marked as outliers and the performance metrics over the remaining sequences.

Table VI shows that the performance of DEEPCASE on non-outlier sequences stays roughly the same across different numbers of injected events. However, the number of outlier sequences increase with the number of injected events. For minor pertubations, the CONTEXT BUILDER is still able to detect enough relevant context events to be able to accurately model the sequence. However, when an attacker injects many events, the number of outlier sequences rises quickly. This means that DEEPCASE notices unusual patterns and escalates these sequences to a security operator.

VI. Discussion

We have shown that our approach successfully reduces the number of events presented to security operators by 95.69% in manual mode and 90.53% in semi-automatic mode. Nevertheless, there are some aspects of our approach to be addressed in future work.

1) Bro/Zeek and programmable rules: Modern NSM and IDS systems such as Zeek [33] often include ways to manually define expert rules. As we have shown in the evaluation, such rule based systems work very well for the scenarios that they cover, and even outperform DeepCASE in terms of accuracy for the covered events. However, the main issue is that manually defined rules often have much difficulty covering all generated events. Therefore, we believe that DeepCASE offers a complementing solution to existing rule-based systems, as we ensure many more events are covered, while remaining conservative in our prediction such that less than 0.001% of events are misclassified with a lower risk level.

2) Evasion: For DoS attacks against DeepCASE, the main limitation of our approach lies in the case where the attacker gradually increases the number of sequences that DeepCASE sends to a security operator. While such an attack can significantly impact the workload reduction achieved by our approach, DeepCASE will still show these sequences produced by the DoS attack to security operators. Under such an attack, DeepCASE performs equivalently to the regular setting where DeepCASE is not used.

Furthermore, we note that in our evation experiment from Section V-D we injected random events. In reality, an adversary with knowledge of existing clusters can maliciously craft its injected events to remain undetected. We observe that the only way in which it is *possible* to change the malicious context of a malicious event e, is if there exist both a malicious cluster (malicious context + event e) *and* a benign cluster (benign context + event e) for that given event. We analyzed our clusters and found that only 6.32% of clusters have this property, meaning that for 93.68% of clusters, it is impossible to inject security events that would turn a malicious sequence into a benign sequence. Notably, we observed that the clusters for which switching is possible are often only a single step in a larger attack (e.g., the clusters for `beaconing activity`, where detection of connecting to the command and control server can be circumvented). Other steps of the attack, such as uploading large amounts of data or signature hits are more difficult to bypass without being triggered as outliers. In short, evasion attacks may be possible for 6.32% of the clusters if an attacker exploits knowledge of how clusters are formed, which is a limitation of DeepCASE.

3) SOAR systems: Our work shows that DeepCASE creates clusters of similar event sequences that correspond to similar risk levels for a machine. However, these clusters are used only to filter benign sequences and classify the risk levels of malicious sequences before showing them to security operators. Similar event sequences intuitively signal similar threats infecting a device. This would mean that the response of the security operator is also similar in terms of removing the threat and patching the device. Such automated response systems are known in industry as "security orchestration, automation and response", i.e., SOAR systems. In its current form DeepCASE could help SOAR systems when operators create automated responses per cluster.

4) Relative risk levels: In its current form, DeepCASE does not distinguish between risk levels for different machines or organizations, it merely produces a risk analysis based on previously observed event sequences. However, in practice, devices that are vital for an organization may have a lower tolerance for potential risks than other devices. As our approach analyzes event sequences on a per-device basis, we suggest to forward lower risk event sequences to the security operators for vital devices. If this is done only for a small set of devices, the impact on workload reduction should be minimized. However, further research is required to evaluate the full impact on workload reduction.

5) Attention and explainability: Our approach uses an attention mechanism to select relevant parts of the context of an event. This attention mechanism is a popular approach in the current state-of-the-art research into natural language processing (NLP). In this domain, there is an ongoing discussion whether attention may be used for explaining feature importance [27]. The main critique here is that the attention vector used in state-of-the-art works (e.g., BERT [14]) does not apply attention 1-to-1 to each input, but rather maps attention to a complex combination of different inputs. We mitigate this critique by multiplying the attention directly with the embedding of each context event creating a direct mapping (see the Event Decoder in Section III-B3). Furthermore, the results from our evaluation show that combining the attention with each individual event can be used for accurate matching and filtering of event sequences.

6) Transferability: Our approach models event sequences based directly on the events produced by underlying security detection systems. This means that changes in the detectors will affect the performance of DeepCASE in terms of filtered sequences. We have shown that our approach can automatically update itself with new detectors in Section V-C2. In some scenarios, an operator may want to take a pre-trained model from one organization and apply it to another organization to avoid having to run the manual mode. Further research could show us how well existing models can transfer to other settings, using methods such as transfer learning [32].

7) Context: One limitation of DeepCASE is that it only deals with events in the context of the same machine. While it would be interesting to find cross-host relations between events, we view this as future work. Another limitation of DeepCASE is the limited size of the context it can deal with. Therefore, attacks over long periods of time and contexts filled up with many unrelated events cannot be properly assessed. Our parameter selection (Appendix D) showed that incrementing the size of the context (both in terms of time and number of events) beyond 5 samples and 1 hour has limited effects on the performance.

8) Transformers: Our approach uses an Attention-based Encoder-Decoder model. Recent advances in the field of NLP have improved this type of architecture in the form of Transformers [42], of which notable examples are BERT [14] and GPT-3 [7]. These transformers are based on the same concept of attention as our work, but offer a larger amount of parallelization. For a more detailed description, we refer to the original paper [42]. While this transformer architecture

would also work for DEEPCASE, the increased complexity of such a network would add little helpful insights into our main concept, namely that attention can be used to explain relations between security events.

VII. RELATED WORK

Related works have explored various ways to contextualize and predict security events, automating the operators' tasks.

A. Contextual security events

The default method of analysing security events is provided by expert rules such as those provided by AlienVault's OSSIM, Sigma, and the rules that can be programmed in software such as Zeek [33]. However, as we have shown in the evaluation, these rules often only cover a limited subset (in our evaluation a maximum of 36.97%) of all event sequences that we observe. More automated methods such as NoDoze [22] and UNICORN [21] model the security context of system-level events and organization-wide events, respectively. Both approaches model this context as provenance graphs that track which processes are connected to triggered events. They then automatically assign an anomaly score to this sequence to assist the triaging process. The main drawback of these approaches is that it requires process-level information of monitored hosts in order to construct provenance graphs. Other works, such as OmegaLog [23], go even further in providing security operators with additional provenance information by analyzing the executed binaries. Our work drops the requirement for process-level information altogether by focusing only on the security events themselves, and later identifying correlated events using the CONTEXT BUILDER. This allows us to handle events of less device-intrusive detection mechanisms such as network-level security detectors. Additionally, security operators can monitor more devices, including those operating in bring-your-own-device settings.

B. Event prediction

Other works do not focus on classifying the threat level of attacks, but rather focus on predicting the next attack steps given a sequence of prior security events.

DeepLog [16] employs a recurrent neural network to predict future events. In case the predicted event does not happen, DeepLog raises an alerts for an anomaly in the event sequence. When their approach detects an anomaly, it is considered malicious and passed through a separate workflow extraction system to detect the underlying cause. Like many other systems, DeepLog focuses on prediction of system logs, which means that it is optimized for more detailed events. This can also be observed from the similarity in performance between DEEPCASE and DeepLog on the HDFS dataset, and the difference in the VMWARE dataset. Furthermore, DeepLog focuses only on anomalies, but an anomaly is not necessarily malicious.

Tiresias [36] does not predict anomalies, but instead tries to accurately predict future events. While achieving a decent performance with an F1-score of 95.68% versus 97.90% of DEEPCASE, Tiresias is a complete black-box approach to event prediction. This means that security operators have no way of telling whether a particular prediction is meaningful given a specific context.

In another paper, the same authors propose Attack2vec [37], which detects *changes* in attack patterns based on differences in preceding security events. While this is also a form of contextual analysis, the goal and approach are strictly different. Where Attack2vec only detects changes in attack trends, DEEPCASE is able to also cluster these new attacks and present them to security operators to immediately take action.

One of the earlier works in security event prediction is Nexat [10]. Nexat uses a co-occurrence matrix of security events to predict the most likely next event. However, this approach both assumes straightforward relations between events and is fully supervised. This makes it more difficult to deal with rapidly evolving attack patterns as it constantly needs to be retrained. Conversely, DEEPCASE offers security operators a simple and effective approach to semi-automatically update itself to deal with novel event sequences.

1) Attention-mechanisms: ALEAP [19] uses an attention mechanism for event prediction. Unfortunately, their work does not leverage this mechanism to provide contextual analysis, and, therefore, cannot properly assist security operators in their work. Moreover, their approach only has a prediction performance of only 72.36% precision (no mention of accuracy/recall/F1-score). Besides the lower performance, the complexity of the model is much higher than the one used in our approach, leading to longer training and prediction times.

Brown et al. [6] also use attention mechanisms to enhance prediction of security log messages. However, they predict specific log attributes based on different attributes within the same log message. Hence, they base their prediction of maliciousness on meta-data of individual messages, such as the machine generating an event, or the user that authenticated an action, instead of other activities within the network. This is simply a different approach to what many individual security detection mechanisms already do.

2) Graph-based approaches: Other works capture context through graph-based approaches, which can be built from data [23], [30], [38]. The disadvantage of such approaches is that they rely on predefined rules to model context. Therefore, new patterns will remain unobserved, making it difficult to give a complete overview of the context.

VIII. CONCLUSION

In this work, we proposed DEEPCASE, a novel approach that assists security operators in analyzing security events by inspecting the context of events. Unlike existing approaches, this work does not require system-level information and can therefore be used to analyze security events of any type of security detector. Moreover, we showed that DEEPCASE is able to deal with complex and evolving attacks without resorting to a black-box approach.

Additionally, we showed that DEEPCASE reduces the workload of security operators on real-world data by 95.39%, and semi-automatically handles 90.53% of events with an accuracy of 94.34%. Moreover, DEEPCASE underestimates risk in less than 0.001% of cases, showing that real attacks are rarely missed. These results demonstrate that contextual event analysis is an effective technique for security event analysis and a useful tool for real-world security operations centers.

REFERENCES

[1] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[2] Saleema Amershi, Bongshin Lee, Ashish Kapoor, Ratul Mahajan, and Blaine Christian. Human-guided machine learning for fast and accurate network alarm triage. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI, 2016.

[3] Muhamad Erza Aminanto, Lei Zhu, Tao Ban, Ryoichi Isawa, Takeshi Takahashi, and Daisuke Inoue. Combating threat-alert fatigue with online anomaly detection using isolation forest. In *International Conference on Neural Information Processing*, pages 756–765. Springer, 2019.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems*, pages 1–8, 2018.

[7] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] Tobias Chyssler, Stefan Burschka, Michael Semling, Tomas Lingvall, and Kalle Burbeck. Alarm reduction and correlation in intrusion detection systems. In *Detection of intrusions and malware & vulnerability assessment, GI SIG SIDAR workshop, DIMVA 2004*. Gesellschaft für Informatik eV, 2004.

[10] Casey Cipriano, Ali Zand, Amir Houmansadr, Christopher Kruegel, and Giovanni Vigna. Nexat: A history-based approach to predict attacker actions. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, pages 383–392, 2011.

[11] CISCO. Anticipating the Unknowns - Chief Information Security Officer (CISO) Benchmark Study. Technical report, 2019. Retrieved from https://ebooks.cisco.com/story/anticipating-unknowns/ on 2020-9-16.

[12] Frédéric Cuppens and Alexandre Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings 2002 IEEE symposium on security and privacy*, pages 202–215. IEEE, 2002.

[13] Demisto. The State of SOAR Report. Technical report, 2018. Retrieved from https://start.paloaltonetworks.com/the-state-of-soar-report-2018 on 2020-9-16.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. Lifelong Anomaly Detection Through Unlearning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1283–1297, 2019.

[16] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1285–1298, 2017.

[17] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[18] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 96, pages 226–231, 1996.

[19] Shuhan Fan, Songyun Wu, Zhiliang Wang, Zimu Li, Jiahai Yang, Heng Liu, and Xinran Liu. Aleap: Attention-based lstm with event embedding for attack projection. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2019.

[20] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[21] Xueyuan Han, Thomas F. J.-M. Pasquier, Adam Bates, James Mickens, and Margo I. Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *27th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2020.

[22] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *Network and Distributed Systems Security Symposium (NDSS)*, 2019.

[23] Wajih Ul Hassan, Mohammad A Noureddine, Pubali Datta, and Adam Bates. Omega-log: High-fidelity attack investigation via transparent multi-layer log analysis. In *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[24] Grant Ho, Aashish Sharma, Mobin Javed, Vern Paxson, and David Wagner. Detecting credential spearphishing in enterprise settings. In *26th USENIX Security Symposium*, pages 469–485, 2017.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Xin Hu, Jiyong Jang, Marc Ph Stoecklin, Ting Wang, Douglas L Schales, Dhilung Kirat, and Josyula R Rao. Baywatch: robust beaconing detection to identify infected hosts in large-scale enterprise networks. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 479–490. IEEE, 2016.

[27] Sarthak Jain and Byron C Wallace. Attention is not explanation. *arXiv preprint arXiv:1902.10186*, 2019.

[28] Faris Bugra Kokulu, Ananta Soneji, Tiffany Bao, Yan Shoshitaishvili, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Matched and mismatched socs: A qualitative study on security operations center issues. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1955–1970, 2019.

[29] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[30] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1777–1794, 2019.

[31] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 1137–1152. IEEE, 2019.

[32] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[33] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[34] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium*, pages 729–746, 2019.

[35] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.

[36] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting security events through deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 592–605, 2018.

[37] Yun Shen and Gianluca Stringhini. Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *28th USENIX Security Symposium*, pages 905–921, 2019.

[38] Xiaokui Shu, Frederico Araujo, Douglas L Schales, Marc Ph Stoecklin, Jiyong Jang, Heqing Huang, and Josyula R Rao. Threat intelligence computing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1883–1898, 2018.

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2818–2826, 2016.

[40] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, 2014.

[41] Martin Ukrop, Lydia Kraus, Vashek Matyas, and Heider Ahmad Mutleq Wahsheh. Will you trust this tls certificate? perceptions of people working in it. In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, pages 718–731, 2019.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

[43] Thomas Vissers, Jan Spooren, Pieter Agten, Dirk Jumpertz, Peter Janssen, Marc Van Wesemael, Frank Piessens, Wouter Joosen, and Lieven Desmet. Exploring the ecosystem of malicious domain registrations in the .eu tld. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 472–493. Springer, 2017.

[44] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP)*, pages 117–132, 2009.

[45] Carson Zimmerman. Ten strategies of a world-class cybersecurity operations center. Technical report, 2014. Retrieved from https://www.mitre.org/publications/all/ten-strategies-of-a-world-class-cybersecurity-operations-center on 2020-10-26.

APPENDIX A
ATTENTION QUERY

Traditional attention-based neural networks use attention to focus on specific inputs while predicting an output. In this work, we introduced an attention query[15], which asks the neural network "given the actual event that occured, which attention distribution *would* have resulted in the correct prediction?" This is a powerful technique that allows us to automatically learn complex relations within the context of security events.

The attention query uses backpropagation to optimize the attention vector for a given input and output to the event decoder. Mathematically, the event decoder is represented as

$$y = g(f(W\left(\sum_{i=0}^{n}\alpha_i x'_i\right)+b)) \tag{4}$$

where $g(\cdot)$ is the softmax function, $f(\cdot)$ is the ReLU activation function, $W(\cdot)+b$ is the linear layer and $\sum_{i=0}^{n}\alpha_i x'_i$ is the matrix multiplication. To find the optimal attention for a given output, we compute the derivative of the loss function $L(\hat{y}, y)$ with respect to the variables α, representing the attention vector as in Equation 5. Finally, we use the Adam optimization algorithm with 100 steps to adjust the attention distribution and find the optimal attention distribution for the observed event. Note that this increases the number of contextual event sequences that we can model. However, there can still be cases where the contextual event sequences do not yield any information regarding the observed event. In these situations, the output probability for observed event e_i remains low and we pass the event to the security operator for further evaluation. For a full evaluation we refer to Section V-C3.

$$\frac{\delta L(\hat{y},y)}{\delta \alpha} = \frac{\delta L(\hat{y},y)}{\delta g(\cdot)} \cdot \frac{\delta g(\cdot)}{\delta f(\cdot)} \cdot \frac{\delta f(\cdot)}{\delta \alpha x'} \cdot \frac{\delta \alpha x'}{\delta \alpha} \tag{5}$$

APPENDIX B
ATTENTION EXAMPLE

In Section V-B2, we showed that naive clustering does not generalize as well as DEEPCASE which includes the CONTEXT BUILDER. To give a concrete example of this difference in performance, consider the following sequence of events as listed in Table VII. Here, the attention is heavily focused on the `Beaconing activity`. Therefore, using DEEPCASE, it will be clustered in a `Beaconing activity` cluster, whereas the naive clustering approach would place this in a cluster that covers both `Beaconing activity` and `Cryptocurrency`.

[15]We provide a pytorch implementation of the attention query at https://github.com/Thijsvanede/DeepCASE

TABLE VII
EXAMPLE SEQUENCE OF EVENTS. THIS EXAMPLE SHOWS THAT THE ATTENTION WILL ENSURE THIS SEQUENCE IS CLUSTERED IN A `BEACONING ACTIVITY` CLUSTER.

Event	Attention
`Login to cryptocurrency mining pool`	0.0048
`Login to cryptocurrency mining pool`	0.0048
`Login to cryptocurrency mining pool`	0.0047
`Login to cryptocurrency mining pool`	0.0047
`BitCoinMiner`	0.0043
`BitTorrent`	0.0050
`Beaconing activity`	0.2564
`FlyStudio`	0.0048
`Beaconing activity`	0.3336
`Beaconing activity`	0.3769

APPENDIX C
VMWARE DATASET DETAILS

Table VIII gives a more detailed description of the VMWARE dataset used in our evaluation. This dataset captures security events of 20 international organizations of different sizes.

TABLE VIII
DETAILS OF VMWARE DATASET. AN OVERVIEW OF THE NUMBER OF SECURITY EVENTS PRODUCED PER ORGANIZATION, CATEGORIZED BY RISK LEVEL.

			Risk level				
Org.	Machines	Events	INFO	LOW	MEDIUM	HIGH	ATTACK
1	4184	24422	24422	0	0	0	0
2	13	2879	2706	173	0	0	0
3	50	878	515	341	3	2	17
4	1376	8888	6553	798	1473	6	58
5	386	2599	1459	639	395	83	23
6	229627	7117123	4858475	2191848	19215	10761	36824
7	881	132416	123841	2358	2366	3848	3
8	2185	59595	57680	1430	224	29	232
9	53381	319975	304394	10615	658	27	4281
10	358	3337	2742	501	58	0	36
11	1973	81523	77316	3174	179	807	47
12	1123	14867	13344	1460	12	1	50
13	4607	191545	180352	6855	3642	134	562
14	14	1953	1883	70	0	0	0
15	188	26202	25927	275	0	0	0
16	23	382	260	74	27	3	18
17	18802	2062074	1850477	145995	32273	30584	2745
18	67749	340819	200296	15844	124379	111	189
19	74	2789	2302	483	0	0	4
20	391	6521	6140	373	3	5	0

APPENDIX D
PARAMETER SELECTION

DEEPCASE uses the parameters listed in Table IX. We determined the values of these parameters by performing a 10-fold grid search on the first 1% of data of the VMWARE dataset sorted by time. This first 1% is split 50:50 into training and testing sets, and is used only for the parameter selection, i.e., it is not used in further experiments.

A. Sequencing events

The context is defined by 1) the maximum number of events in the context (length) and 2) the maximum time difference between an event and its context (time). To obtain the optimal

TABLE IX
PARAMETERS. VALUES OF ALL PARAMETERS USED IN DEEPCASE.

Subsystem	Parameter	Value	Section
Sequencing events	Sequence length	10	D-A
	Sequence time	1 day	D-A
CONTEXT BUILDER	hidden dimension	128	D-B
	δ	0.1	D-B
INTERPRETER	$\tau_{\text{confidence}}$	0.2	D-C
	ϵ	0.1	D-C
	minimum sequences	5	D-C

values, we performed a 10-fold grid search over length values 1, 2, 5, 10, 20 and time values of 1 minute, 1 hour, 1 day, 1 week. For all combinations, we trained the CONTEXT BUILDER with a hidden dimension of 128 and a δ of 0.1, and we evaluated whether the threshold for the corresponding event was higher than 0.2 ($\tau_{\text{confidence}}$). We found that for all input sizes $\geq$ 5 and times $\geq$ 1 day, between 95.00% and 95.02% of events were correctly predicted from their given context. Therefore, for both length and time values we chose the middle option of 10 events with a maximum age of 1 day.

B. The CONTEXT BUILDER

Similar to sequencing events, we performed a 10-fold search for the hidden dimension of the CONTEXT BUILDER, and evaluated whether the threshold for a correctly predicted event reached at least 0.2. Here, we searched powers of 2, 2^1,2^2,2^3,...2^{10} and found an optimal value for $2^7 = 128$. The same search over the δ values of 0.0 to 1.0 with steps of 0.1 yielded an optimal value of 0.1. For the δ value, the increased performance was mainly due to correct classification of classes with very few samples. This follows from the idea of the δ value, which increases performance at the cost of a slightly reduced confidence.

C. The INTERPRETER

Finally, we used the values obtained from the parameter selection of event sequences and the CONTEXT BUILDER to select parameters for the INTERPRETER. Here we performed a grid search over the $\tau_{\text{confidence}}$ values and ϵ values, both ranging from 0.1 to 1.0 with steps of 0.1. During this experiment, we measured the overall performance in terms of the F1-score, yielding a $\tau_{\text{confidence}}$ of 0.2 and ϵ of 0.1. For the minimum sequences, i.e., the minimum number of sequences to be considered a cluster, we chose 5 to give the security operator enough samples to provide a confident prediction. We elaborate on this choice further in Section V-B3.

APPENDIX E
WORKLOAD REDUCTION - HDFS DATASET

Table X shows the workload reduction achieved by DEEPCASE on the HDFS dataset. In this overview, the alert throttling is left out because the dataset does not contain any timestamp information for the alerts. Additionally, there is no comparison with rulesets as there are no expert-rules available for this dataset.

TABLE X
WORKLOAD REDUCTION - HDFS. AVERAGE WORKLOAD REDUCTION OF DEEPCASE COMPARED WITH NAIVE CLUSTERING TECHNIQUES*.

		Workload reduction			
	Method	Alerts	Reduction	Coverage	Overall
Manual	DEEPCASE	393	95.71%	96.39%	92.26%
	N-gram	1,204	86.58%	94.33%	81.68%
	Cluster DEEPCASE	446	95.26%	99.01%	94.32%
Semi-Automatic	DEEPCASE	N/A	100.00%	96.43%	96.43%
	N-gram	N/A	100.00%	93.83%	93.83%
	Cluster DEEPCASE	N/A	100.00%	98.82%	98.82%

* As this evaluation is on the HDFS dataset, there are no available rules or timestamps.

APPENDIX F
CLUSTER EXAMPLES

We recall that the INTERPRETER produces clusters by comparing the contextual events in a sequence weighted by the attention vector. These models describe the total attention of each security event type in contextual event sequences. As clusters contain similar sequences, we can describe its characteristics from the attention-weighted events. We describe a cluster by simply averaging the attention values for each event over all sequences.

These cluster descriptions uncover interesting patterns that illustrate how an operator could reason about assigning risk levels. Figure 5 gives examples of these descriptions. Here the relevance of events found in the security context of a cluster is scored according to its average attention value. Consider the event in the VMWARE dataset where a detector observes an `unusual user agent string`. This may be due to a newly installed or otherwise benign program, or it may be triggered by malware that does not imitate common user-agents. This event occurs 4.6K times in the dataset, which our INTERPRETER groups into 19 different clusters. To determine whether an event of type `unusual user agent string` is malicious or not, we have to look at the context weighed by the attention values. The first example in Figure 5 shows a case with 1) `NO CONTEXT` description if the event occurs where fewer than 10 detectors were triggered in the day before observing the event, indicating that there is no other supporting evidence for an attack; and 2) other detectors for the similar patterns such as `unusual JA3 fingerprint`. This cluster does not directly indicate malicious behavior. However, other cluster descriptions in Figure 5 show the `unusual user agent string` event in combination with detectors for the `Tor browser`. Depending on the organization, this may be considered a policy violation and can be classified as such. Finally, there are clusters with more malicious indicators such as observing `unusual user agent string` in combination with a `large data download`, or signature hits for malware (e.g., `Linkury`) or `crypto miners`.

As another example, let us look at the contexts observed for `beaconing activity`, which can be triggered by malware periodically contacting its command and control server, or can be triggered by benign software periodically checking for updates. We observe several clusters for `beaconing activity`. Some clusters contain only events by detectors looking for repetitive network connections, which do not necessarily indicate malicious activity. Other

`unusual user agent string`	(benign)	
`NO CONTEXT`		47.95%
`unusual user agent string`		7.91%
`unusual JA3 fingerprint`		42.70%
`unusual user agent string`	(malicious)	
`NO CONTEXT`		20.84%
`unusual user agent string`		35.09%
`unknown crypto miner`		43.93%
`beaconing activity`	(malicious)	
`NO CONTEXT`		3.68%
`unusual data upload`		32.29%
`active directory trust enumeration`		1.21%
`recently registered domain access`		62.64%

Fig. 5. **Cluster description examples.** Three examples of both benign and malicious clusters. All clusters are described by the average relevance of other contextual events as described by the attention.

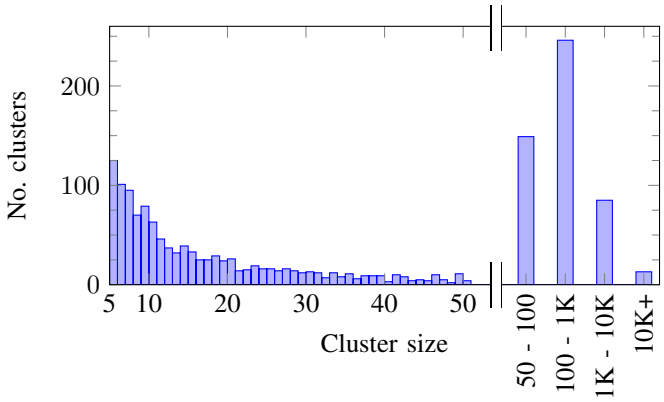

Fig. 6. **Cluster size histogram of clusters.** Clusters are skewed toward the smaller sizes. This behavior is observed for all risk levels. In general, the resulting workload reduction is most beneficial from the largest clusters.

clusters, such as shown in Figure 5, show signs of malware because `beaconing activity` is detected in combination with `recently registered domain access` and `unusual data uploads`.

APPENDIX G
CLUSTER DISTRIBUTION

Clusters produced by the INTERPRETER vary in sizes as some attacks or benign patterns are more frequent than others. Figure 6 gives an overview of the skewed distribution of clusters. We suggest that security operators sample a fixed number of sequences from each cluster. This means that the main body of workload reduction is due to large clusters. Other cluster risk levels show the same skewed distribution.

APPENDIX H
RUNTIME ANALYSIS

We evaluate the average of 10 runs of DEEPCASE for various number of sequences. All experiments ran on a Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz machine running Ubuntu 18.04 LTS. Neural network training and prediction ran on a NVIDIA TITAN RTX 24 GB TU102 graphics card.

Figure 7 shows the result of this analysis. The total runtime is made up of four main computations: 1) training the CONTEXT BUILDER; 2) the attention query; 3) INTERPRETER's clustering in manual mode; and 4) matching known clusters

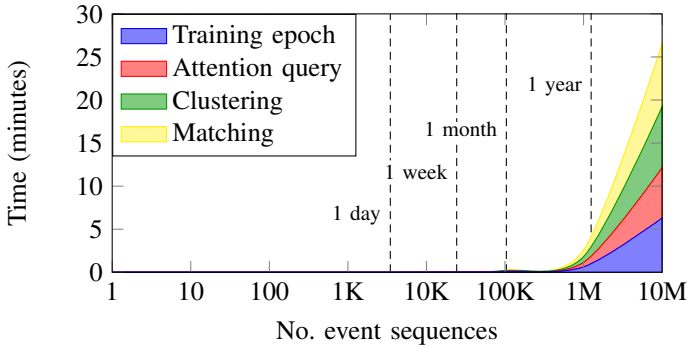

Fig. 7. **Runtime analysis of DEEPCASE.** Average runtime of our approach for different number of contextual sequence inputs. The runtimes of each sub-computation are stacked to show total runtime of DEEPCASE. For reference, we show the number event sequences an average organization in the VMWARE dataset produces during one day, week, month and year.

in semi-automatic mode. We note that Figure 7 shows only a single training epoch of the CONTEXT BUILDER, which in reality is trained with 100 epochs. However, these epochs scale linearly and need to be trained only once for the manual mode, or when updated. From this figure, we find that it takes DEEPCASE less than 5 minutes to process 1 year of data for a single company (roughly 1.2M event sequences, based on the average number of sequences in the VMWARE dataset). Training the neural network of CONTEXT BUILDER consumes the largest part of the total runtime. We note that training epochs together with the attention query can be highly parallelized as they are performed on a GPU. Furthermore, we note that while DBSCAN clustering has a worst case complexity of $O(n^2)$, using KD-trees [5] allows us to reduce that to a complexity of $O(n\log n)$. Overall, DEEPCASE can easily keep up with the number of generated events in large, real-world environments.