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.

{ts.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 [4] or Snort [1]). 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 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 [11]; 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
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.
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 $\{e_0, \ldots, e_{i-1}\}$ 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 naive 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 $\{e_0, \ldots, e_{i-1}\}$ 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 (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

---

3For 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-C.
events. We represent each contextual event as a vector using
incorrect. Our analysis is described in more detail below.
Section III-C1 describes what happens if the this prediction is
next events. Finally, the system checks whether this “pre-
decoder computes a probability distribution over all possible
relevance) of each context event. Then, the event decoder
it into an attention vector which specifies a weight (i.e.,
attention mechanism [4]
perform the analysis. This approach uses what is known as
fall back on the security operator and their existing tools to
When the C
events correlated with
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
performs the analysis. This approach uses what is known as
an Encoder-Decoder [8] architecture in combination with an
attention mechanism [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
relevance values for each contextual event. 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 $\alpha$ with the
one-hot encoded context sequence $S' = [e_{i0}, e_{i1}, ..., e_{in}]$, 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

We discuss transformers as an attention mechanism in Section VI-B.
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 weights 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, DEEP CASE 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 \(\delta\) is scattered over the other events using their frequency distribution. In this way 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 \([13]\), 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, DEEP CASE 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 total attention per 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 DEEP CASE using semi-automatic analysis (Section \([III-I]\)).
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 DEEP CASE.

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-D). 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 DEEP CASE. 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 vectors of total attention per event 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, DEEP CASE informs the security operator to take action.

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 DEEP CASE in practice. LASTLINE dataset. The real-world LASTLINE dataset consists of 20 international organizations that use 395 detectors to monitor 388K devices. This resulted in 10.5M security events for 291 unique types of security events 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 DEEP CASE on the HDFS dataset used in the evaluation of the related security log

[^1]: In some cases it may ever be possible to fully automate the response for known malicious clusters, we discuss this use of DEEP CASE as basis for SOAR systems in Section VII.

[^2]: 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.

[^3]: 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 LASTLINE 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 LASTLINE 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 II 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 occurring 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 LASTLINE events and the training data of the HDFS dataset.

1) Coverage: Table II shows that for the LASTLINE 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 LASTLINE 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 sequences.
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 C 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 6B, 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 I 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.

$$P(\text{suspicious}|k) = 1 - \sum_{c \in C} \left( \frac{\binom{|c|}{k}}{\binom{N}{k}} \right)$$  \hspace{1cm} (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
be passed to the security operator for manual inspection. In they will either form new clusters or are outliers that will vectors not match any of the known clusters, and, as a result, it is filtered and not shown to the security operator. Some remove the threat. If the event sequence is found to be benign, risk level are escalated to the security operator who can then an organization, event sequences labeled higher than a given according to the matching cluster. Depending on the policy of tools provide options to throttle events as well as options for mostly due to similar event sequences observed for different machines that were not of vital importance to the business. Section VI-4 explores the adjustment for business continuity. Section [V-B1] shows the performance of DEEP CASE on the LASTLINE 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 III, 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 $\tau_{threshold}$ 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, DEEP CASE underestimates only 47 sequences, which is less than 0.001% of cases.

To understand why DEEP CASE 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 [V-B1] 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 DEEP CASE.

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
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 LastLine 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 LastLine dataset with expert rules from the LastLine; 292 open source rules from AlienVault’s OSSIM that cover 82 different known attacks; and the open source rules from Sigma [14] that cover known attacks from various resources such as the MITRE ATTCK 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 LastLine dataset [18]. This allows us to directly apply the Sigma and OSSIM rules to the events in the LastLine 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 LastLine ruleset. We note that there are many other solutions that provide rule-based detection such as Azure Sentinel [12] and Splunk [13]. For these solutions, we were unable to obtain publicly 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 II 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 covered 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 II 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 1.

Table II 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.
C. Challenges

This work addressed three challenges that make it difficult to reduce the workload of security operators. 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.\[13\]

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 II, 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.

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 V 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

---

**TABLE IV**

PREDICTION RESULTS. SYSTEMS TRAINE 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.

<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Accuracy</th>
<th>Train time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepLog</td>
<td>89.71%</td>
<td>89.34%</td>
<td>89.35%</td>
<td>89.34%</td>
<td>1.0 s</td>
</tr>
<tr>
<td>Tiresias</td>
<td>90.41%</td>
<td>90.64%</td>
<td>90.40%</td>
<td>90.64%</td>
<td>1.3 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Accuracy</th>
<th>Train time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepLog</td>
<td>89.65%</td>
<td>90.40%</td>
<td>89.82%</td>
<td>90.40%</td>
<td>0:06.8 m</td>
</tr>
<tr>
<td>Tiresias</td>
<td>95.50%</td>
<td>96.21%</td>
<td>95.68%</td>
<td>96.21%</td>
<td>4:51.5 m</td>
</tr>
<tr>
<td>DEEPCASE</td>
<td>97.90%</td>
<td>98.06%</td>
<td>97.90%</td>
<td>98.06%</td>
<td>0:13.8 m</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Updates</th>
<th>Coverage</th>
<th>Metrics over covered data</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>87.38%</td>
<td>Precision</td>
</tr>
<tr>
<td>Monthly</td>
<td>+411K</td>
<td>93.29%</td>
</tr>
<tr>
<td>Weekly</td>
<td>+456K</td>
<td>93.93%</td>
</tr>
<tr>
<td>Daily</td>
<td>+466K</td>
<td>94.23%</td>
</tr>
</tbody>
</table>

---

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

\[14\] Tiresias code is available at https://github.com/Thijsvanede/Tiresias
the CONTEXT BUILDER on the first month of data from the LASTLINE 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 E 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 LASTLINE 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 LASTLINE 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>
<thead>
<tr>
<th>Injected</th>
<th>Outliers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/10</td>
<td>19.92%</td>
<td>96.43%</td>
<td>96.88%</td>
<td>96.43%</td>
<td>96.43%</td>
</tr>
<tr>
<td>1/10</td>
<td>21.22%</td>
<td>97.82%</td>
<td>97.69%</td>
<td>97.26%</td>
<td>97.26%</td>
</tr>
<tr>
<td>2/10</td>
<td>22.86%</td>
<td>98.98%</td>
<td>98.17%</td>
<td>97.77%</td>
<td>97.77%</td>
</tr>
<tr>
<td>3/10</td>
<td>25.51%</td>
<td>98.14%</td>
<td>98.51%</td>
<td>98.14%</td>
<td>98.14%</td>
</tr>
<tr>
<td>4/10</td>
<td>29.67%</td>
<td>98.40%</td>
<td>98.75%</td>
<td>98.40%</td>
<td>98.40%</td>
</tr>
<tr>
<td>5/10</td>
<td>35.55%</td>
<td>98.61%</td>
<td>98.94%</td>
<td>98.61%</td>
<td>98.61%</td>
</tr>
<tr>
<td>6/10</td>
<td>42.66%</td>
<td>98.78%</td>
<td>99.08%</td>
<td>98.78%</td>
<td>98.78%</td>
</tr>
<tr>
<td>7/10</td>
<td>50.26%</td>
<td>99.32%</td>
<td>99.20%</td>
<td>99.32%</td>
<td>99.32%</td>
</tr>
<tr>
<td>8/10</td>
<td>57.88%</td>
<td>99.57%</td>
<td>99.30%</td>
<td>99.57%</td>
<td>99.57%</td>
</tr>
<tr>
<td>9/10</td>
<td>68.94%</td>
<td>99.76%</td>
<td>99.36%</td>
<td>99.76%</td>
<td>99.76%</td>
</tr>
<tr>
<td>10/10</td>
<td>98.01%</td>
<td>99.83%</td>
<td>99.40%</td>
<td>99.83%</td>
<td>99.83%</td>
</tr>
</tbody>
</table>

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

Furthermore, we note that in our evasion 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 DEEP CASE.

3) SOAR systems: Our work shows that DEEP CASE 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 DEEP CASE could help SOAR systems when operators create automated responses per cluster.

4) Relative risk levels: In its current form, DEEP CASE 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 DEEP CASE 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 DEEP CASE 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 DEEP CASE 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 increasing 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 alert 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 LASTLINE 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], which 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.
ACKNOWLEDGEMENTS

We would like to thank our reviewers for their valuable comments, which significantly improved our paper. This work was partially supported by the MechaMan, NSF CNS-1704253 and CHECRS, DARPA/AFRL FA8750-19-C-0003 grants, as well as the Netherlands Organisation for Scientific Research (NWO) in the context of the SeReNity project.

DECLARATION OF INTERESTS

Over the course of this research, Marco Cova, Christopher Kruegel, and Giovanni Vigna were employed by Lastline. In June 2020, after the conclusion of the main research activities, Lastline was acquired by VMware. As part of this transactions, the three authors joined VMware.

REFERENCES

can still be cases where the contextual event sequences do not contextually represent the observed event. Note that this increases the number of examples that we can model. However, there remains a large number of events that we cannot predict. 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.

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 occurred, what 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

\[
\hat{y} = g(f(W \sum_{i=0}^{n} \alpha_i x_i^t) + b)
\]

where \(g(\cdot)\) is the softmax function, \(f(\cdot)\) is the ReLU activation function, \(W(\cdot) + b\) is the linear layer and \(\sum_{i=0}^{n} \alpha_i x_i^t\) 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 \(\alpha\), 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 \(e_t\) remains low and we pass the event to the security operator for further evaluation. For a full evaluation we refer to Section V-C3.

APPENDIX B
ATTENTION EXAMPLE

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

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

<table>
<thead>
<tr>
<th>Org.</th>
<th>Machines</th>
<th>Events</th>
<th>INFO</th>
<th>LOW</th>
<th>MEDIUM</th>
<th>HIGH</th>
<th>ATTACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4184</td>
<td>24422</td>
<td>24422</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>2879</td>
<td>2706</td>
<td>173</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>878</td>
<td>515</td>
<td>341</td>
<td>3</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>1376</td>
<td>8888</td>
<td>6553</td>
<td>798</td>
<td>1473</td>
<td>6</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>386</td>
<td>2599</td>
<td>1459</td>
<td>639</td>
<td>395</td>
<td>83</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>229627</td>
<td>7117123</td>
<td>4858475</td>
<td>2191848</td>
<td>19215</td>
<td>10761</td>
<td>36824</td>
</tr>
<tr>
<td>7</td>
<td>881</td>
<td>32416</td>
<td>123841</td>
<td>2358</td>
<td>2366</td>
<td>3848</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2185</td>
<td>95955</td>
<td>57680</td>
<td>1430</td>
<td>224</td>
<td>29</td>
<td>232</td>
</tr>
<tr>
<td>9</td>
<td>53381</td>
<td>319975</td>
<td>304394</td>
<td>10615</td>
<td>658</td>
<td>27</td>
<td>4281</td>
</tr>
<tr>
<td>10</td>
<td>358</td>
<td>3337</td>
<td>2742</td>
<td>501</td>
<td>58</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>11</td>
<td>1973</td>
<td>81523</td>
<td>77316</td>
<td>3174</td>
<td>179</td>
<td>807</td>
<td>47</td>
</tr>
<tr>
<td>12</td>
<td>1123</td>
<td>14867</td>
<td>13344</td>
<td>1460</td>
<td>12</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>4607</td>
<td>191545</td>
<td>183035</td>
<td>6855</td>
<td>3642</td>
<td>134</td>
<td>562</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>1953</td>
<td>1883</td>
<td>70</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>188</td>
<td>26202</td>
<td>25927</td>
<td>275</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>21</td>
<td>382</td>
<td>260</td>
<td>24</td>
<td>27</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>18802</td>
<td>2062074</td>
<td>1850477</td>
<td>145995</td>
<td>32273</td>
<td>30584</td>
<td>2745</td>
</tr>
<tr>
<td>18</td>
<td>67749</td>
<td>340819</td>
<td>200296</td>
<td>15844</td>
<td>124379</td>
<td>111</td>
<td>189</td>
</tr>
<tr>
<td>19</td>
<td>74</td>
<td>2789</td>
<td>2302</td>
<td>483</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>391</td>
<td>6521</td>
<td>6140</td>
<td>373</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

APPENDIX C
LASTLINE DATASET DETAILS

15We provide a pytorch implementation of the attention query at https://github.com/Thijsvanede/DeepCASE.
We elaborate on this choice further in Section V-B3. Operator enough samples to provide a confident prediction. To be considered a cluster, we chose 5 to give the security the F1-score, yielding a experiment, we measured the overall performance in terms of ranging from selecting event sequences and the C selection, i.e., it is not used in further experiments.

Finally, we used the values obtained from 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 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 $\delta$ 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 $\delta$ values of 0.0 to 1.0 with steps of 0.1 yielded an optimal value of 0.1. For the $\delta$ value, the increased performance was mainly due to correct classification of classes with very few samples. This follows from the idea of the $\delta$ 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 $\epsilon$ 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 $\epsilon$ 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 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 LASTLINE 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.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Parameter</th>
<th>Value</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequencing events</td>
<td>Sequence length</td>
<td>10</td>
<td>D-A</td>
</tr>
<tr>
<td></td>
<td>Sequence time</td>
<td>1 day</td>
<td>D-A</td>
</tr>
<tr>
<td>CONTEXT BUILDER</td>
<td>hidden dimension</td>
<td>128</td>
<td>B-B</td>
</tr>
<tr>
<td></td>
<td>$\delta$</td>
<td>0.1</td>
<td>B-B</td>
</tr>
<tr>
<td>INTERPRETER</td>
<td>$\tau_{\text{confidence}}$</td>
<td>0.2</td>
<td>C-C</td>
</tr>
<tr>
<td></td>
<td>$\epsilon$</td>
<td>0.1</td>
<td>C-C</td>
</tr>
<tr>
<td></td>
<td>minimum sequences</td>
<td>5</td>
<td>C-C</td>
</tr>
</tbody>
</table>

**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 rule sets as there are no expert-rules available for this dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Alerts Reduction</th>
<th>Coverage</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepCASE</td>
<td>393</td>
<td>95.71%</td>
<td>96.39%</td>
</tr>
<tr>
<td>N-gram</td>
<td>1,204</td>
<td>86.58%</td>
<td>94.33%</td>
</tr>
<tr>
<td>Cluster DeepCASE</td>
<td>446</td>
<td>95.26%</td>
<td>99.01%</td>
</tr>
</tbody>
</table>

The HDFS dataset does not contain any timestamps, nor are there any rules available. Therefore, we could not compare with alert throttling or rule based approaches.

**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 vectors detailing the total attention per event. 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 LASTLINE 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
unusual user agent string  (benign)  47.95%
NO CONTEXT  7.91%
unusual user agent string  42.70%

unusual user agent string  (malicious)  20.84%
NO CONTEXT  35.09%
unknown crypto miner  43.93%

unusual JA3 fingerprint  47.95%
NO CONTEXT  7.91%
unusual user agent string  42.70%

beaconing activity  (malicious)  3.68%
NO CONTEXT  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.

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 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 DEEP CASE for various numbers of sequences. All experiments ran on an 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 in semi-automated 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 DEEP CASE 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 LASTLINE 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, DEEP CASE can easily keep up with the number of generated events in large, real-world environments.

APPENDIX H

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