DrSec: Flexible Distributed Representations for Efficient Endpoint Security

Mahmood Sharif*[†], Pubali Datta[‡], Andy Riddle[§], Kim Westfall[§], Adam Bates[§], Vijay Ganti[¶], Matthew Lentz[∥][†], and David Ott[†]

*Tel Aviv University [†]VMware [‡]University of Massachusetts Amherst [§]University of Illinois Urbana-Champaign [¶]Google [∥]Duke University

Email: mahmoods@tauex.tau.ac.il, pdatta@umass.edu, {rriddle2, kw26, batesa}@illinois.edu, ganti.vijay1@gmail.com, mlentz@cs.duke.edu, dott@vmware.com

***Abstract*—The increasing complexity of attacks has given rise to varied security applications tackling profound tasks, ranging from alert triage to attack reconstruction. Yet, security products, such as Endpoint Detection and Response, bring together applications that are developed in isolation, trigger many false positives, miss actual attacks, and produce limited labels useful in supervised learning schemes. To address these challenges, we propose `DrSec`—a system employing self-supervised learning to pre-train foundation language models (LMs) that ingest event-sequence data and emit distributed representations for processes. Once pre-trained, the LMs can be adapted to solve different downstream tasks with limited to no supervision, helping unify the currently fractured application ecosystem. We trained `DrSec` with two LM types on a real-world dataset containing ∼91M processes and ∼2.55B events, and tested it in three application domains. We found that `DrSec` enables accurate, unsupervised process identification; outperforms leading methods on alert triage to reduce alert fatigue (e.g., 75.11% vs. ≤64.31% precision-recall area under curve); and accurately learns expert-developed rules, allowing tuning incident detectors to control false positives and negatives.**

***Index Terms*—Endpoint security, EDR, language models, self-supervision, alert triage, process identification**

1. Introduction

In response to the increasing sophistication of advanced threats, the complexity of the enterprise-security software ecosystem is growing rapidly. As low-level functions such as intrusion detection have proven insufficient to successfully defend organizations, today's products are also expected to effectively prioritize and triage detections (e.g., [57]), construct unified explanations of suspicious incidents (e.g., [15]), enrich system-event activity with context (e.g., [67]), and more. This complexity is also reflected in recent academic research, which recently began to investigate these specific tasks (e.g., attack-story reconstruction [4], [41], [69], alert triage [25], [26], [38]). As is especially evident in the literature, learning-based approaches are quickly supplanting traditional rule- and signature-based techniques as a means of providing necessary functions.

In spite of the diversity of these security functions, each ultimately operates over system-level events collected by endpoint sensors. Consider traditional Endpoint Detection and Response (EDR) deployments, which largely make use of heuristic/rule-based detection models. The endpoint sensors will collect streams of system events that are grouped by process and subsequently matched against pre-defined rules. For example, when a user deletes a file, a rule may be triggered based on the MITRE ATT&CK technique "File Deletion" [39]. Naturally, while this captures some malicious behavior (e.g., ransomware), it is also a source for false positive alerts as many benign processes also delete user-created files (e.g., disk clean-up tools). Therefore, some applications may also rely on automated approaches for specific sub-tasks; for instance, EDR systems often use rule-based alert detection, but may augment this with ways to address the problem of having too many false positives to effectively triage the alerts [1], [10], [56].

However, there are a number of challenges faced by such automated approaches, which often involve learned models. First, as highlighted by the file deletion example, it is important to understand how the sequence of past events (or their causal relationship) affects the interpretation of the current event. Second, specialized models often require significant feature engineering as well as large amounts of labeled training data to adequately train models. These involve substantial human effort (and thus incur high cost). Finally, while all of these applications are enabled through the same telemetry streams (i.e., process events), they are typically implemented in an ad-hoc manner without accounting for any overlapping requirements.

Our insight is that if we pre-train language models (LMs) to represent processes, we could easily fine-tune them for various downstream tasks. Our inspiration comes from advances in the natural language processing (NLP) domain; we explore if a similar approach can be applied to the security domain. The benefits we can derive from this approach can address many of the problems and challenges faced by various security applications. Instead of developing ad-hoc, end-to-end models for each application, we can instead reuse the LM as the core, foundation model [11]. Not only does this avoid (costly) manual feature development, but it also requires less labeled training data when fine-tuning for the specific task while maintaining an equivalent of accuracy.

In this paper, we present `DrSec`, our system for pre-training and fine-tuning LMs for use in a variety of security applications. We develop an expressive, yet concise, vocabulary to map process-level events (and their properties) to sequences of tokens that serve as input to the LM. This vocabulary is based on the set of events collected by a large, popular, commercial EDR system. Using a large dataset of (unlabeled) tokenized sequences for processes, DrSec pre-trains the LM in an unsupervised manner with a self-supervised approach that attempts to predict the masked tokens given their neighbors. Given a specific security task which augments the pre-trained LM, DrSec can fine-tune the LM weights (and train the output layers) using a small amount of labeled training data associated with the specific task; alternatively, DrSec can be used in an unsupervised manner based on the representations it outputs.

In a nutshell, we make the following contributions:

1) **We design and implement DrSec**, a system leveraging self-supervised learning of LMs to create distributed representations summarizing process behavior. The LMs can then serve as a common foundation unifying the development of different security applications. Specifically, the process representations produced by the LMs can be used in different learning schemes, in an unsupervised fashion or with limited labeled data for fine-tuning, to address varied security tasks.
2) **We apply DrSec to three security tasks**: process identification, alert triage, and EDR-rule learning. Using a real-world dataset containing ∼2.55B events pertaining to ∼91M processes, we pre-trained two LM types (Doc2Vec [32] and transformers [65]) to instantiate two DrSec variants. Subsequently, we tested the resulting distributed representations in security tasks. The experimental results demonstrate that DrSec can enable accurate process identification, outperform leading methods on alert triage, and facilitate learning expert-developed rules.

Next, we provide necessary background and describe the threat model (§2–3). Then, we present an overview of DrSec before diving to its technical details and design (§4–5). Afterward, we present our evaluation of DrSec (§6) and discuss the findings and open problems (§7). Finally, we discuss related work (§8) and conclude (§9).

2. Background and Motivation

We begin by considering the state of the art in endpoint security, grounding our discussion in Carbon Black's major commercial EDR product [66]. Carbon Black provides an integrated EDR product offering detection and investigation capabilities, as well as security information and event management (SIEM) support [8]. The software architecture is comprised of distributed endpoint sensors and a cloud-based platform for configuration and analysis.

Like most commercial EDR solutions, this system primarily performs heuristic/rule-based detection. In rule-based detection systems, queries are defined by first- or third-party analysts that encode known malicious behaviors. These detection queries are annotated with additional useful context, such as natural language descriptions of the attack behavior and pointers to associated techniques in the MITRE ATT&CK framework [40]. At the time of this writing, EDR provides over 2,000 possible detection queries to customers that are organized into sets of company- and community-defined lists. Customers are able to configure their EDR deployments to alert on any of these queries, and even define their own. The use of rule-based detection queries is not the primary focus of our work, although we will demonstrate how our system and interoperate and enrich alert streams.

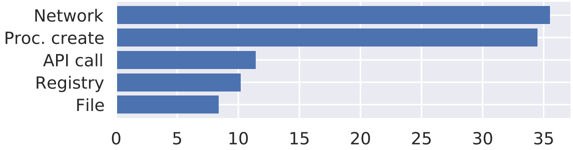


Figure 1: **Event-type frequency.** The percentage of events belonging to each event type in $\mathbf{D}_{\mathbf{Large}}$.

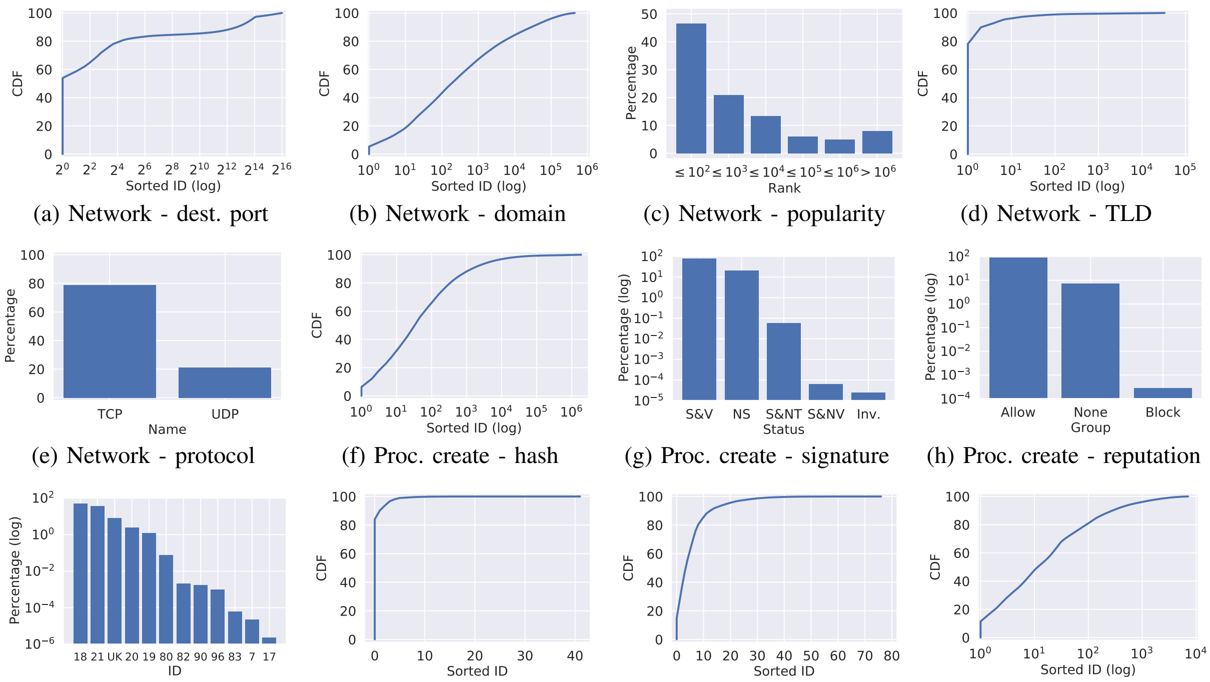
The detection mechanisms in EDRs perform pattern matching over endpoint *telemetry* data; when an alert fires, this telemetry can then be reviewed by analysts to investigate the incident. Endpoint telemetry is primarily comprised by system-event logs that are collected via commodity operating system (OS) audit frameworks–e.g., Event Tracing for Windows or Linux audit–but individual events can also be enriched with additional information such as whether a program's executable was signed by a trusted developer. Lastly, many EDRs have the ability to export telemetry logs for further analysis; we use this feature of Carbon Black's EDR to generate evaluation datasets for this study.

Specifically, our system makes use of the following enriched event-telemetry attributes:

- *Network Event* telemetry carries five attributes describing the destination port, the domain name, the domain's popularity per the Tranco ranking [44], the top-level domain (TLD), and the transport protocol.
- *Process-Creation Event* telemetry contains information about the binary's SHA-256 hash, signature status, and reputation, as well as a process security identifier indicating the user group the process runs under.
- *File-Access Events* telemetry reports a combination of flags detailing whether the file or its attributes were modified, the file type (data or executable), whether the file was encountered by the sensor for the first time, and the file's location (on a fixed, removable, or network drive).
- *API-Call Event* telemetry reports the OS API function called by a process.
- *Registry-Access Event* telemetry reports the registry value accessed.

2.1. Endpoint Telemetry Characterization

To gain a better sense of the relative contributions of these event types, we present a characterization of a large dataset, $\mathbf{D}_{\mathbf{Large}}$, that contains telemetry data collected by a large EDR vendor. This dataset was created sampling the events for ∼42k processes per hour over three months



(a) Network - dest. port
(b) Network - domain
(c) Network - popularity
(d) Network - TLD
(e) Network - protocol
(f) Proc. create - hash
(g) Proc. create - signature
(h) Proc. create - reputation
(i) Proc. create - SID
(j) File access - flag comb.
(k) API call - function name
(l) Registry access - value

Figure 2: **Prevalence of event attributes.** Distributions of attribute values in the telemetry events in $\mathbf{D_{Large}}$.

(04/27/2021 to 07/26/2021) across all customer endpoints running a Windows OS. For each process, we collected all events from within the hour it was sampled from. In total, the dataset describes ∼91M processes and ∼2.55B events. Fig. 1 reports the prevalence of each event type. Network and process creation events are most common in the dataset, comprising 35.50% and 34.49% of all events, respectively. File accesses are least common, comprising 8.39% of events.

Fig. 2 depicts the event attributes and the distributions of their values. Network-event attributes are described in Figs. 2a–2e. The 443 TCP destination port is most commonly encountered (53.92% of events), showcasing the prevalence of HTTPS, but other ports such as 7680 (Windows Update Delivery Optimization), 389 (LDAP), and 80 (HTTP), are also common (≥2.97%). The distribution of domain names is less skewed, with 183 domains and IP addresses receiving roughly half the traffic. As expected, most traffic (46.59%) flows to top-100 domains, and mostly (78.13%) to the .com TLD (out of 33,064 TLDs encountered). Only the TCP and UDP transport protocols are monitored by the telemetry, and TCP is more prevalent (78.69%).

Process-creation attributes are described in Figs. 2f–2i. The process hashes follow a power-law distribution, with 1,399 hashes of the unique 1.84M encountered accounting for >90% of events. Hashes pertaining to common system (e.g., svchost.exe and taskhostw.exe) and user (e.g., the Chrome browser and Microsoft Office) processes are highly prevalent. Most binaries (79.43%) are signed by a trusted certificate authority and successfully verified, while others are not signed (20.51%), signed by an untrusted authority (5.90%), signed but not successfully verified (<0.01%), or have invalid signatures (<0.01%). Most processes created are found on an allow list (92.82%), while some are unlisted (7.18%) or blocked by the company's or the EDR provider's policy (0.03%). Finally, the SIDs indicate that most created processes run with system (51.04% with SID 18) or user (36.89% with SID 21) privileges.

File-access, API-call, and Registry-access attributes are described in Figs. 2j–2l. Analyzing the file access flag combinations, we can see that there are 42 flag combinations encountered, with modifications of files on a fixed drive accounting for 84% of events. We observed 77 different API-calls with the 14 most common ones (e.g., FindFirstFileExW for file search and CreateWindowExW for window creation) accounting for >90% of events. Finally, analysis of registry-access values shows the distribution of values is skewed, with >80% of events containing only one of 92 unique values.

2.2. Challenges in Endpoint Security

Having reviewed EDR features and characterized a representative sample of event telemetry data, we now consider some of the central challenges in endpoint security today.

Interpreting Event Attributes. As Fig. 2 highlights, many security-sensitive event attributes–process hashes, API call types, etc.–can be difficult to interpret. Crucially, the importance of a given attribute highly depends on its value, including attributes that may include tens or hundreds of thousands of different possible values. Ultimately, the high dimensionality of these attributes places great cognitive burden and domain-expertise demands on human analysts, and

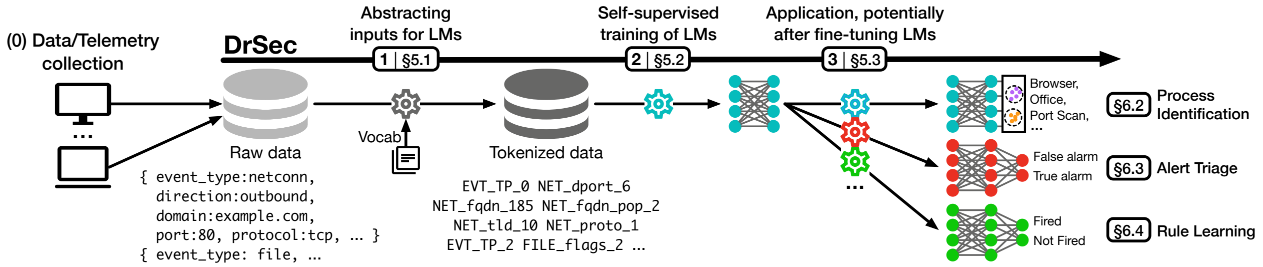


Figure 3: **Overview of DrSec**. (0) Endpoint devices contribute event telemetry data collected by sensors. (1) DrSec pre-processes the data to map events to sequences of tokens as input to the LM. (2) DrSec pre-trains the LM in a self-supervised manner over the token sequences. (3) DrSec specializes the LM for specific security-related application (e.g., alert triage) through fine-tuning over a limited dataset.

may make it difficult for machine-learning (ML) approaches to converge to useful models.

A Fractured Ecosystem of Security Applications. We have referenced a variety of important security applications include threat detection, threat investigation, alert management (i.e., SIEM), and incident response. While all of these applications are fundamentally enabled by the same telemetry streams, they are typically implemented in isolation without regard for their overlapping requirements.

Alert Fatigue (Too Many False Positives). EDR products are prone to generating an unmanageable number of threat alerts [1], [10], [56]. FireEye reports that large enterprises may receive tens of thousands of alerts per week, with the majority being false positives [19]. This causes many alerts to go uninvestigated or underinvestigated, leading to incidents where an attack is detected by the EDR but never responded to by analysts (e.g., the Target data breach [45]). Further, even when alerts are properly investigated, false alerts simply waste analysts' time at reported rates of 10-40 minutes per alert [1], [10].

Missed Detections (Too Many False Negatives). It is also very possible for even state-of-the-art EDRs to fail to detect sophisticated intrusion behaviors. Structurally, rule-based detection systems are only able to identify known attacker behaviors, such that a novel attack vector or zero day attack may be unanticipated by the EDR's detection queries. Along similar lines, it is also possible for an intrusion to be nominally detected, but only by queries associated with lower severity or confidence scores. As a result of threat alert fatigue, these true alerts may blend into the noise of false alarms and go unnoticed.

Limited Labeled Data. Due to the sheer amounts of process and event telemetry collected, only a negligible fraction of data is manually inspected by experts. For example, according to conservative estimates, a cyber analyst can investigate roughly three alarms per hour [1], [10], [26]. By contrast, Carbon Black's EDR collects data from $\gg$42k processes per hour. Thus, security applications ingesting telemetry data (e.g., for alert triage [64]) need to be extremely (labeled) data efficient, or, alternately, operate without labeled data (e.g., in clustering schemes) [55].

In this work, we make progress towards addressing these grand challenges through DrSec. DrSec produces LMs that reduce the dimensionality of event attributes through a concise vocabulary, enabling easier analysis of telemetry. Through the creation of this pre-trained LM, DrSec provides a unified intermediate representation of telemetry data for use in a variety of security applications. We show that DrSec can be used to mitigate the shortcoming of rule-based detection engines, providing applications such as alert triage or process identification that are not captured by EDR queries. To our knowledge, DrSec is the first system tackling all five crucial challenges simultaneously.

3. Threat Model

Our deployment model adopts the same deployment and threat models as commercial EDRs. Sensors that generate telemetry data are deployed on device endpoints. As is the case with commercial products, we assume that adversaries can compromise machines but cannot disable or violate the integrity of endpoint sensors. In practice, tampering with sensors is a risky proposition; any disruption could immediately trigger a high severity alert, and developers follow strict standards to protect sensors from compromise. We assume a cloud-based analytics model in which telemetry is transmitted to central (trusted) servers for analysis.

Our solution introduces ML tasks to detection systems that were previously rule-based, potentially expanding the attack surface of the EDR. The adversary may attempt to poison training data to induce false negatives in downstream applications. Alternately, the attacker may be able to induce false negatives by patterning their own behaviors to match those of known applications (i.e., mimicry attacks [68]). We consider the feasibility of such attacks in §6.5.

4. Overview

Fig. 3 provides a high-level overview of DrSec's workflow. Initially, telemetry data is collected by sensors running on a large number of endpoint devices and, optionally, multiple organizations. The raw telemetry data consists of process-level events such as opening new network connections, creating a child process, or filesystem accesses.

The DrSec workflow begins by (1) pre-processing the telemetry data to transform sequences of events for a specific process into sequences of tokens that can be ingested by the LM. We use a vocabulary developed out-of-band; similarly to the NLP domain [31], [32], we construct the vocabulary carefully by deriving specific tokens for common events and event properties, and mapping uncommon events to generic tokens. The resulting vocabulary should balance being sufficiently descriptive to differentiate between different processes (e.g., malicious and benign), while also constrained enough to ensure the resulting LM is space- and time-efficient. The flexibility in the vocabulary allows DrSec to adapt to arbitrary endpoint sensor technologies.

Next, DrSec (2) pre-trains the LM in an unsupervised manner. Pre-training follows self-supervised schemes that leverage the existing structure within the data to produce pseudo-labels, which are then used to derive meaningful distributed representations for processes and events. For example, in the NLP domain, pseudo-labels could be computed by examining a window of n tokens and attempting to predict the subsequent token. LMs trained via such schemes have been found to learn representations that capture the semantics of sentences and documents [17], [32], [37]. We expect that self-supervised LMs can also capture the semantics of processes by taking advantage of the correlations between events and event properties.

Once trained, DrSec enables (3) the application of the LM to various downstream security applications, such as alert triage and malware detection. This involves fine-tuning by extending the pre-trained model and using limited amounts of labeled data that are specific to the downstream application (e.g., dataset containing processes with benign and malicious processes). Note that fine-tuning is not always necessary if the representations from the LM can be used directly in the application (e.g., for process identification). We use the term fine-tuning broadly to refer to two approaches: *1)* training a smaller model on the LM's distributed representations, and *2)* attaching a classifier on top of the LM and adjusting the weights via end-to-end training with the labeled dataset.

5. Technical Approach

We now detail the three primary stages of DrSec. We begin with how we abstract events and processes for ingestion by DrSec LMs (§5.1). Next, we describe how we train LMs in a self-supervised manner (§5.2). Then, we explain how we use LMs for specific applications, potentially after adaptation using a minimal labeled dataset (§5.3).

5.1. Abstracting Events and Processes

Central to our work is the insight that processes and their event sequences can be viewed as a language. In return, this language can be modeled via LMs, producing flexible distributed representations to address downstream computer-security problems. Consider the example of the Microsoft Word process and its corresponding events presented in Tables 1–2, which will serve as a running example for the remainder of this section. In this example, the winword.exe (i.e., Microsoft Word) program is executed with user privileges, as reflected by the Windows security ID [36]. Furthermore, we can notice that this program is placed on an allow list by the EDR provider, and that it has been (cryptographically) signed by the developer and successfully verified by the OS. The winword.exe process was created by Microsoft's program manager, explorer.exe, that is also signed and placed on an allow list, and runs under user privileges. As shown in Table 2, the winword.exe process exhibits a sequence of three events. First, it calls the SetClipboardViewer API to register to clipboard events. Second, it creates a new file for writing. Third, it connects to the Microsoft-owned domain, via TCP on port 443 (i.e., HTTPS). We can also see that this domain is highly popular, as it pertains to the top 100 most popular domains. The process, parent, and events (and their properties) follow a strict structure. The formal (regular) language presented in this section can capture these details.

	Name	*Sign.*	*Reputation*	*Security ID*
Process	winword.exe	S&V	Allow list	S-1-5-<#>
Parent	explorer.exe	S&V	Allow list	S-1-5-<#>

TABLE 1: **Example**. Available telemetry for a winword.exe process and its parent. S&V stands for signed and verified.

#	*Type*	*Details*
1	API	Function: SetClipboardViewer
2	File	Flags: CREATE
3	Network	DestPort: 443 Domain: nexus.officeapps.live.com Popularity: TOP_100 TLD: .com Protocol: TCP

TABLE 2: **Example**. Events exhibited by winword.exe.

The language we design to represent processes seeks to strike a good balance between two competing properties. On the one hand, it aims to remain sufficiently *descriptive* to distinguish different processes and their behaviors (e.g., benign and malicious) apart. On the other hand, it seeks to be *abstract* enough to make it amenable to modeling via LMs. One way to control the complexity is by limiting the size of the vocabulary: if the vocabulary is too large such that certain tokens appear only a few times during training, the LM would fail to capture their semantics. By contrast, if the vocabulary is too small, then it would become challenging to represent semantically different expressions. Indeed, abstractions are widely used in domains where LMs are often applied. For instance, in NLP, stop words (e.g., words like "the" or "to") are sometimes removed to facilitate learning with little-to-no impact on sentence semantics [48]. In this work, we abstract process and event details to DrSec by following the standard practice of including unique tokens to represent only the most prevalent details in the vocabulary [31], [32]. Specifically, we rely on our data characterization (§2.1) to ensure the most frequent event details are captured by the vocabulary.

$$S \rightarrow S_{proc} \cdot S_{parent} \cdot (E_{net}|E_{cproc}|E_{file}|E_{api}|E_{reg})^*$$
$$S_{proc} \rightarrow A_{phash} \cdot A_{sign} \cdot A_{rep} \cdot A_{secid}$$
$$S_{parent} \rightarrow A_{phash} \cdot A_{sign} \cdot A_{rep} \cdot A_{secid}$$
$$E_{net} \rightarrow \texttt{EVT\_TP\_0} \cdot A_{dport} \cdot A_{fqdn} \cdot A_{pop} \cdot A_{tld} \cdot A_{proto}$$
$$E_{cproc} \rightarrow \texttt{EVT\_TP\_1} \cdot A_{phash} \cdot A_{sign} \cdot A_{rep} \cdot A_{secid}$$
$$E_{file} \rightarrow \texttt{EVT\_TP\_2} \cdot A_{acctype}$$
$$E_{api} \rightarrow \texttt{EVT\_TP\_3} \cdot A_{fname}$$
$$E_{reg} \rightarrow \texttt{EVT\_TP\_4} \cdot A_{regval}$$

Figure 4: **The process abstraction-language defined as a regular grammar** for balancing between the descriptiveness and generalizability of raw telemetry streams.

5.1.1. Overall Abstraction. Given the available process and event details studied in §2, we present a regular grammar to represent processes as an abstract language. Once represented by an expression in this language, it may be impossible to *entirely* recreate the process and event details (i.e., the abstraction is not invertible). However, as demonstrated in experiments (§6), the language is sufficiently detailed to capture key process behavior.

Fig. 4 presents the abstraction language's grammar. Expressions describing processes in this language start with preambles summarizing processes' and their parents' properties (non-terminal symbols S_{proc} and S_{parent}, respectively). These consist of tokens (a.k.a. terminal symbols) describing the process hash (A_{phash}), signature status (A_{sign}), reputation (A_{rep}), and security identifier (A_{secid}). Following the preambles, the expressions contain a sequence of tokens capturing event types (EVT_TP_X) and their corresponding details. For network events (E_{net}), the language includes tokens describing the destination port (A_{dport}), domain name or IP address (A_{fqdn}), the domain popularity (A_{pop}), the top-level domain (A_{tld}), and the network protocol (A_{proto}); for process-creation events (E_{cproc}), details such as process hash, signature status, reputation, and security identifier are captured in the language; for files accesses (E_{file}), the language encodes information about the access type ($A_{acctype}$); for API function calls (E_{api}), the function name (A_{fname}) is included; while for registry accesses, the registry value (A_{regval}) is appended. Note that the size of the vocabulary is limited by the number of unique values that terminal symbols (e.g., A_{fqdn}) admit.

①PROC_HASH_2831 SIGN_STAT_0 PROC_REP_0 PROC_SECID_3 ②PROC_HASH_850 SIGN_STAT_0 PROC_REP_0 PROC_SECID_3 ③EVT_TP_3 API_func_38 ④EVT_TP_2 FILE_flags_2 ⑤EVT_TP_0 NET_dport_6 NET_fqdn_185 NET_fqdn_pop_2 NET_tld_10 NET_proto_1 ...

① Process Preamble ② Parent Preamble ③ Event 1 ④ Event 2 ⑤ Event 3

Figure 5: **Running example**. Abstraction of the Microsoft Word process from Tables 1–2.

Fig. 5 illustrates an abstraction of the Microsoft Word process and its corresponding events from the running example (Tables 1–2) using the proposed language. Notice how the telemetry data (e.g., the domain name from Table 1) are mapped to tokens within the language's vocabulary. Also note how the expression resulting from concatenating the tokens is accepted by the language in Fig. 4. This expression can be ingested by LM for training or producing a distributed representation of the process at inference time.

While we designed this grammar for a particular EDR telemetry-stream, this approach is a general means for describing processes and their behavior as a language. In particular, we emphasize that the language can be extended to encode new event types and event attributes by introducing new symbols. Furthermore, by configuring the vocabulary size (i.e., the unique terminal symbols included in the language), one can control the fidelity in which expressions in the language represent processes.

5.2. Self-supervised LM Training

We consider two LM types, one based on Doc2Vec [32] and another based on transformers [65]. The former represents a family of models that were popular between circa early 2010s. Transformers, on the other hand, constitute an advanced neural network architecture driving some of the most transformative results in ML in recent years, including the GPT models used for language generation [11], and AlphaFold, which has revolutionized protein-structure prediction [30]. By considering both classic and state-of-the-art models, our work enables evaluating whether recent advances in NLP translate to the security domain. Additionally, because Doc2Vec models are typically more computationally efficient than transformers and less dependent on the availability of dedicated hardware (e.g., GPUs), evaluating both model types allows us to explore the possibility of attaining good performance with limited compute resources.

5.2.1. Doc2Vec. Similarly to standard LMs, given a window of n tokens, the training objective of Doc2Vec is to predict the n+1th token [32]. Doc2Vec's so-called distributed memory model, can be seen as a neural network of three layers. The first layer embeds each of the n tokens to a vector space and creates an embedding for the entire sequence (i.e., document). The sequence embedding introduces context useful for predicting the n+1th token. The second layer aggregates the sequence and token embeddings via averaging. Lastly, the third layer feeds the aggregated embeddings to a logistic classifier to predict the next token. At training time, different token sequences and windows are used, and all embeddings are updated via an optimizer to improve the prediction of the next word. During inference, the parameters are frozen except those of the sequence embedding, which are updated by the optimizer. This sequence embedding is then used for downstream tasks.

5.2.2. Transformers. The transformer architecture is widely used to learn distributed representations [65]. Training transformers typically follows the masked-LM self-supervised training task where the goal is to predict the identity of a randomly selected subset of masked tokens [17], [34]. Given an input token sequence, a subset of tokens is first selected for masking at random. Additionally, the beginning and end of the sequence are marked via special tokens. The resulting sequence is then ingested by the transformer model, embedding tokens in a vector space, adding positional information, and passing them through several stages of encoding and decoding. The transformer's output is composed of multiple embeddings, one per input token. As part of the self-supervised training tasks, an optimizer updates the transformer's parameters such that the true masked tokens are more accurately predicted given the mask embeddings. At inference time, the embedding of the special start-of-sentence token is used as the distributed representation for the entire sequence. We use the RoBERTa transformer model [34], a variant of the popular BERT model [17] whose training procedure is more stable.

5.3. Fine-Tuning and Application

After self-supervised training, the distributed representations can be leveraged in downstream tasks. For classification tasks where labeled datasets are available (e.g., alert triage), DrSec trains or fine-tunes an ML classifier in a supervised manner, following standard practices [17], [32], to solve the task. For Doc2Vec, the classifier (e.g., random forest [9] or support vector machine [14]) takes the distributed representations as input, and is trained from scratch to predict the label. For the transformer, we attach a shallow neural network classifier that takes the distributed representation of the start-of-sentence token as input, and fine-tune the model end-to-end with the labeled data. In certain applications (e.g., process identification), labeled data may be unnecessary or unavailable. For those, we simply use DrSec's distributed representations in a completely unsupervised manner, without fine tuning, to solve the tasks.

6. Evaluation

We now present our evaluation of DrSec. After describing our experimental setup (§6.1), we consider the use of DrSec in three complementary security applications in a threat-hunting environment: process identification (§6.2), alert triage (§6.3), and EDR rule learning (§6.4). We close the section with a security analysis of DrSec against attacks threatening its integrity (§6.5).

6.1. Experimental Set-Up

We evaluate two variants of DrSec: one using Doc2Vec ($\text{DrSec}_{\text{Doc2Vec}}$) and another using a RoBERTa transformer ($\text{DrSec}_{\text{RoBERTa}}$). We pre-trained both models on a large-scale dataset, $\mathbf{D_{Large}}$. For both models we used a vocabulary size of 50K, keeping only the most common tokens in $\mathbf{D_{Large}}$. This is in line with vocabulary sizes commonly used in NLP [32]. According to our telemetry characterization (§2.1), the most common 50K tokens account for 98.72% of events and event attributes in $\mathbf{D_{Large}}$, leaving only a small fraction of event attributes (1.28%) to be represented by generic tokens signifying they are rare.

For Doc2Vec, we made use of the Gensim implementation [21]. To select parameter values, we withheld a small portion of the dataset (~73K processes) and conducted a grid search to select the model hyperparameters that attain best next-word prediction accuracy with limited training effort. This led us to select a window size of five, representation dimensionality of 200, and ten pre-training epochs. Other parameters were set to Gensim's defaults. Pre-training occurred on a machine with an 48-core Intel Xeon 8175M CPU and 192GB of RAM. Completing pre-training on this machine took less than two days.

For RoBERTa, we used the Hugging Face implementation [29]. We used the default model parameters of the RoBERTa-base model, producing distributed representations of 768 dimensions. Because transformers have quadratic space and time complexity in sequence length, we trimmed the input sequence length to 384, keeping only prefixes, to make training feasible on our hardware. We pre-trained the RoBERTa model for one epoch on a machine equipped with an eight core Intel Xeon E5-2686 CPU, 64GB of RAM, and an Nvidia Tesla V100 GPU containing 16GB of memory. Pre-training the model took seven days.

6.1.1. Baselines. We used three systems for baseline comparisons. Unlike DrSec, two of the systems (specifically, DeepCase and TTP) are bespoke to alert triage, so they are not used in all experiments. The last system, relying on term frequency, is used in all experiments.

DeepCase Proposed by van Ede et al. [64], DeepCase is a semi-supervised learning system for alert triage. Similarly to DrSec, DeepCase first learns representations of event sequences in a self-supervised manner. DeepCase representations are learned using a recurrent neural network with an attention layer. The network ingests an input sequence, one token at a time, and outputs positive weights summing up to one for each event in the sequence. The weights are then summed up per token type, creating a vector to represent the sequence (dimensionality of vocabulary size). The weights of the model are optimized during training to enable predicting the next token, given its preceding sequence. The resultant vector representation are then used in a supervised-learning scheme for alert triage.

The original DeepCase work used a vocabulary size of 291 for event abstraction and an input-sequence length of 10. We used different abstractions of events (e.g., by encoding or not encoding the popularity of visited websites in network events) and compiled vocabularies of sizes 139, 337, and 354. Additionally, we tested varied input lengths $\in \{10, 20, 30, 40\}$. Using the author's original implementation and model architecture, we pre-trained DeepCase on $\mathbf{D_{Large}}$ using all hyperparameter combinations, finding that the largest vocabulary (354 tokens) and input-sequence length (40) achieved marginally better results than other options. Accordingly, we report DeepCase performance using these best-performing hyperparameters. Note that DeepCase relies on a different LM architecture and event abstractions that are much less descriptive than DrSec's (e.g., vocabulary size of 354 versus 50K in DrSec).

TTP . As a naïve alternative for merging rule- and learning-based detection, we also evaluated DrSec's alert-triage application against a random forest classifier that takes as input a feature vector of Tactics, Techniques, and Procedures (TTPs) rules [71] matched against processes and attempts to predict whether alerts are true or false alarms. The feature vector indicates whether a TTP was exhibited by a process or not (i.e., a feature corresponding to a TTP is set to 1 if the TTP was exhibited by the process, and 0 otherwise). Naturally, each TTP reflects a detection-query rule designed by an expert for reasoning over event sequences. The experts also need to maintain and adapt existing rules and develop new ones as attackers develop new, more sophisticated attacks. Hence, relying on TTP-based detectors is relatively expensive and time consuming. We tested this approach for alert triage only when using a certain dataset ($\mathbf{D_{SOC}}$; see §6.1.2), as TTP data were otherwise unavailable. The dataset contains 85 different TTPs, ranging from OS credential dump (MITRE T1003) to process discovery (MITRE T1057). In other words, the classifier inputs are 85-dimensional binary features.

Token Occurrence and Frequency To further characterize the progress enabled by DrSec and contrast its deep learning-based approach with classical ones, we compared it against versatile baselines relying on term occurrence and frequency. Specifically, we considered models using bag-of-words, term frequency, and term frequency-inverse document frequency (TF*IDF) features, akin to prior work (e.g., [74]). For fair comparison, we produced these features using DrSec's vocabulary (50K tokens). Among these baselines, we found that TF*IDF with log normalization–where tokens' feature values are the product between one plus their log-frequency and the logarithm of their document-frequency's inverse–consistently achieved the best performance. Hence, in the interest of clarity, we only report the TF*IDF results.

6.1.2. Datasets.

We used four datasets in our evaluation.

D_{Large} For a dataset description, please refer to §2.

D_{SOC} We received one alert-triage dataset from the security-operation center (SOC) of the EDR vendor. The $\mathbf{D_{SOC}}$ dataset contains 757 processes triggering alarms that were manually inspected by expert analysts and classified as true and false alarms. The dataset is highly imbalanced with a 1:10 ratio between true and false alarms. Samples in the dataset represent a diverse set of real-world scenarios. Examples of true alarms include a PowerShell process with administrator privileges editing a sensitive part of the file system, a running service continuously attempting to execute malware, and a potentially unwanted program accessing the file system. Examples of false alarms include a system-cleaning utility performing an update and a Microsoft Word process suspected of executing malware. We split the dataset at random into training, validation, and test sets (481, 18, and 258 samples, respectively).

$D_{AtlasV2}$ We also evaluated alert-triage performance on a dataset captured on two Windows 7 endpoints as they are attacked by ten real-world attack scenarios described in ATLAS [4]. The ten attack scenarios involve four single-host attacks and six multi-host attacks which incorporate lateral movement by the attacker through a poisoned web page on the first victim machine. These attacks exploit different vulnerabilities in Adobe Flash Player and Microsoft Word as their initial entry points.

Rule name	*N:P Ratio*	*Description*
Email run	167	Email client invokes another app.
Key log. w/o UI	886	Program monitors user input.
Policy bypass	5,796	Process bypasses the device policy.
Net server	769	App. unexpectedly acts as a network server.
Non EXE run	269	App. invokes another with a non-exe extension.
Office run	451	Microsoft Office process invokes another app.
Sys. config.	1,570	App. attempts to modify a system config.
Util. run	3,767	Unexpected program invokes a system utility.
Virus-like	1,422	App. exhibits a suspicious behavior.
Web run	192	Browser invokes another app.

TABLE 3: **Rules in $\mathbf{D_{Rules}}$**. For each rule, we include its name, the number of negative (N) processes—i.e., ones that have not triggered the rule—per positive (P) samples, and a brief description.

This dataset is a reproduction of the original ATLAS engagement, using the virtual machines and attack scripts that were open-sourced by the authors, so that we could deploy the Carbon Black EDR to collect uniform telemetry data across all datasets. In addition to the attacks, $\mathbf{D_{AtlasV2}}$ contains diverse benign user data manually generated by two authors using the machines as their primary workstations for a four-day benign period. The attacks were then launched on a final fifth day, during which the authors took turns continuing to use the machines as workstations while the other author briefly suspended normal use to trigger the attacks. After data collection, ground truth labels for the telemetry data were obtained based on the known attack entities identified by the ATLAS authors (e.g., the names of malicious files or websites used in the compromise).

D_{Rules} For the EDR-rule learning application, we obtained an additional dataset containing 125,577 processes and 7.31M events from 424 Windows endpoints spanning two weeks (09/01/2022 – 09/14/2022). Unlike $\mathbf{D_{SOC}}$ and $\mathbf{D_{AtlasV2}}$, which only contain suspicious processes that have triggered alerts, $\mathbf{D_{Rules}}$ includes *all* processes executed on the devices, a few of which have triggered alerts (i.e., positive samples) while the majority have not (i.e., negative samples). In particular, $\mathbf{D_{Rules}}$ includes alerts fired by ten different rules included in Carbon Black's product, ranging from ones that alert cyber analysts about key logging to ones that warn about browsers that invoke other applications (indicating potential drive-by-download attacks). Table 3 lists the rules represented in the dataset and their descriptions, accompanied with the number of negative samples for each positive sample in $\mathbf{D_{Rules}}$. Notice how $\mathbf{D_{Rules}}$ is substantially more imbalanced than $\mathbf{D_{SOC}}$ (up to 1:5,796 vs. 1:10 ratio between positive and negative samples). We used a 60-20-20 split to create training, validation, and test sets, respectively.

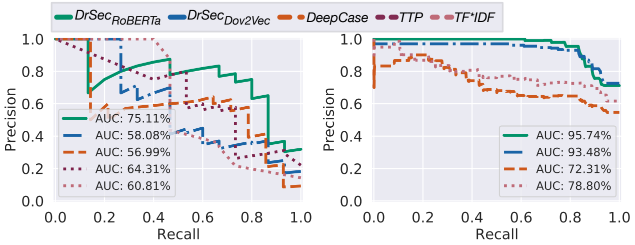
Our use of customer data in $\mathbf{D_{Large}}$, $\mathbf{D_{SOC}}$, $\mathbf{D_{Rules}}$ is consistent with the vendor's data practices, and all customers consent in advance to share data for improving products. Moreover, administrators can opt out selected devices from sharing data. To aid in reproducibility, we release the labcreated $\mathbf{D}_{\mathbf{AtlasV2}}$ and a model pre-trained on it (§6.3) [51]. However, for privacy consideration, no data released is derived from proprietary customer data.

6.2. Security Application: Process Identification

When investigating a suspicious activity, analysts regularly need to quickly determine the identity, reputation, and functionality of a large variety of processes. Even for well-known, high-reputation processes, an automatic update to a program may cause it to emit previously-unseen behaviors that the analyst must interpret as benign or potentially suspicious. DrSec can aid in this task through performing automated process identification. We treated process identification as an unsupervised learning problem and leveraged the similarity between distributed representations of known and unknown processes emitted by DrSec after pre-training to identify the unknown processes.

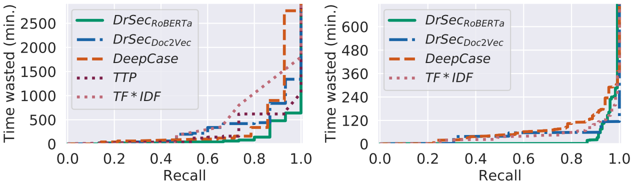
In this experiment, we made use of the $\mathbf{D}_{\mathbf{Large}}$ dataset. We selected all process names with two or more unique program hashes.[1] We then set the processes with the most prevalent hashes as known, and treated the processes with the least prevalent hashes as unknown processes to be identified. This resulted in 2,887 known processes and 94 unknown processes. To predict the process identity, our application measured the Euclidean distance from the test process' representation to all known processes and selected its nearest neighbor(s). We repeated this procedure with both DrSec$_{\text{RoBERTa}}$ and DrSec$_{\text{Doc2Vec}}$, and with TF⋆IDF features, as a baseline. We measured success in terms of the accuracy of unknown process identification.

We found DrSec$_{\text{RoBERTa}}$ was able to correctly identify 65.96% of unknown processes using a single prediction. When inspecting the closest five processes to an unknown process (i.e., top-5 accuracy), DrSec$_{\text{RoBERTa}}$ correctly identified 84.04% of unknown processes, thus substantially reducing the uncertainty about the process' identity from 2,887 possibilities to only five. Upon manual inspection of the 34.04% of misidentified processes, we found that roughly two thirds were mapped to highly related processes. Among mismatched processes were different Microsoft Office programs (e.g., OneNote and Excel), different remote-connection programs (e.g., AnyConnect and GlobalProtect), and different browsers (e.g., Explorer and Chrome). DrSec$_{\text{Doc2Vec}}$ achieved lower accuracy than DrSec$_{\text{RoBERTa}}$ at process-identification: 29.79% top-1 accuracy and 39.36% top-5 accuracy. Still, these were higher than TF⋆IDF's 27.22% top-1 accuracy and 31.95% top-5 accuracy. Notice also that DrSec's performance dramatically outperforms the standard method of identifying processes via hash comparison, which would obtain 0% accuracy in this scenario. DrSec's success in the process identification provides evidence that DrSec's process representations satisfy a desirable property: they are able to summarize process' behavior such that different versions of the same process fall near one another in the Euclidean space.



(a) $\mathbf{D}_{\mathbf{SOC}}$ (b) $\mathbf{D}_{\mathbf{AtlasV2}}$

Figure 6: **Alert-triage PR curves.** The AUCs are reported in the legend.



(a) $\mathbf{D}_{\mathbf{SOC}}$ (b) $\mathbf{D}_{\mathbf{AtlasV2}}$

Figure 7: **Analyst time wasted investigating false alarms.** Following prior work [26], this analysis assumes a uniform average investigation time of 20 minutes per alert.

6.3. Security Application: Alert Triage

Alert triage is an essential security application due to the large number of alerts generated by enterprise security products [1], [10], [56]; in practice, failure to effectively triage the most critical alerts may mean that an actual intrusion goes uninvestigated. We evaluated a DrSec-based alert triage application against two datasets, $\mathbf{D}_{\mathbf{SOC}}$ and $\mathbf{D}_{\mathbf{AtlasV2}}$. Starting with the pre-trained models, we fine-tuned and evaluated DrSec$_{\text{RoBERTa}}$ and DrSec$_{\text{Doc2Vec}}$ on alert triage using $\mathbf{D}_{\mathbf{SOC}}$ and $\mathbf{D}_{\mathbf{AtlasV2}}$. For DrSec$_{\text{RoBERTa}}$, we ran six epochs of fine-tuning and selected the model with the highest validation accuracy. For DrSec$_{\text{Doc2Vec}}$, we experimented with different models (e.g., random forest, support vector machines, and logistic regression), all using the Doc2Vec distributed representations as inputs, and picked random forests, which achieved the highest validation accuracy. Following the same process, we evaluated different classifiers using DeepCase's representations as well as TTP and TF⋆IDF features, finding that random forests had the best validation accuracy in both cases. Due to the imbalanced nature of the datasets, we evaluated performance using precision-recall (PR) curves, per recommended practice [6]. Precision measures the fraction of true alarms out of those classified as true, while recall measures the fraction of true alarms correctly classified out of all true alarms. Higher area under curve (AUC) indicates better performance.

Fig. 6 reports of the performance of each system for the $\mathbf{D}_{\mathbf{SOC}}$ and $\mathbf{D}_{\mathbf{AtlasV2}}$ datasets. DrSec$_{\text{RoBERTa}}$ achieves 10.8%–16.94% higher AUC than the best baselines. Both DrSec variants outperformed DeepCase on both datasets (1.09%–23.43% higher AUC). This result provides evidence that that

[1] Hashes for the same program may differ due to software updates or varied versions installed on different clients.

DrSec may outperform DeepCase when analyzing fine-grained event telemetry; DeepCase was designed to triage much higher-level, coarser-grained events (e.g., “CVE-2018-7600 Exploit”), which is reflected in the design of their architecture that is meant for smaller vocabulary sizes and sequence lengths. We report on the TTP-based system for the $\mathbf{D}_{\mathbf{SOC}}$ dataset in Fig. 6a.[2] The TTP-based approach outperformed DrSec when using the (relatively) simpler Doc2Vec LM (6.23% higher AUC), but was outperformed by the more advanced DrSec$_{\text{RoBERTa}}$ on most operating points, achieving 10.80% lower AUC. Lastly, the TF⋆IDF model achieved surprisingly good performance, with perfect precision for recall values $\leq$40.00% on $\mathbf{D}_{\mathbf{SOC}}$, but was substantially outperformed by DrSec$_{\text{Doc2Vec}}$ on $\mathbf{D}_{\mathbf{AtlasV2}}$ (14.68% lower AUC) and by DrSec$_{\text{RoBERTa}}$ on both datasets (14.30%–16.94% lower AUC).

To interpret these results in a more practical context, we considered the cost (in wasted time) of investigating false alarms based on each triage system. Following prior work, we assumed that analysts require a uniform average time of $\sim$20 to investigate a single alert [26]. For a given percentage threshold of true alerts (i.e., recall), time wasted can then be calculated as 20 minutes times the number of false alarms ranked higher than the lowest ranked true alarm required to reach the threshold. The results are given in Fig. 7. As expected from the PR curves, for almost all operating points, DrSec$_{\text{RoBERTa}}$ could enable analysts to find most true alarms while wasting the least time analyzing false alarms. For instance, on the $\mathbf{D}_{\mathbf{SOC}}$ dataset, analysts using DrSec$_{\text{RoBERTa}}$ could inspect 80% of true alarms while wasting $\geq$2.64× less time (125 vs. $\geq$330 minutes) than other approaches. For the $\mathbf{D}_{\mathbf{AtlasV2}}$ dataset, DrSec$_{\text{RoBERTa}}$ exceeds 80% recall before wasting any analyst time.

Effect of Limited Labeled Data While the amount of labeled data in $\mathbf{D}_{\mathbf{SOC}}$ and $\mathbf{D}_{\mathbf{AtlasV2}}$ is fairly limited compared to the sheer amount of unlabeled data collected via EDR, we wanted to test whether DrSec performs well when labeled data is even more scarce. To this end, we randomly selected 50% of $\mathbf{D}_{\mathbf{SOC}}$'s training set for training, and evaluated the system performance on the *full* test set. In this experiment, we compared the best-performing variant of DrSec, based on the RoBERTa model, and the best-performing baseline, TTP. Fig. 8 in App. A shows the results. Compared to training on the entire training set, the performance of both systems dropped. Still, DrSec$_{\text{RoBERTa}}$ had higher AUC (55.88% vs. 47.32%), attaining better or comparable PR tradeoffs that TTP for most operating points.

Effect of Limited Pre-training Data To assess how the amount of pre-training affects DrSec's performance, we pre-trained a DrSec$_{\text{RoBERTa}}$ variant on parts of $\mathbf{D}_{\mathbf{AtlasV2}}$ and tested it on the remainder of $\mathbf{D}_{\mathbf{AtlasV2}}$ and $\mathbf{D}_{\mathbf{SOC}}$. Specifically, we used $\mathbf{D}_{\mathbf{AtlasV2}}$'s single-host attack scenarios for training and the multi-host scenarios for testing. Overall, the pre-training data of $\mathbf{D}_{\mathbf{AtlasV2}}$ contained 207,470 samples; 438× fewer samples than $\mathbf{D}_{\mathbf{Large}}$. Accordingly, as expected, DrSec$_{\text{RoBERTa}}$'s alert-triage performance when pre-training on $\mathbf{D}_{\mathbf{AtlasV2}}$ was lower than when pre-training on $\mathbf{D}_{\mathbf{Large}}$—52.38% vs. 75.11% PR AUC on $\mathbf{D}_{\mathbf{SOC}}$ and 71.98% vs. 94.74% AUC on $\mathbf{D}_{\mathbf{AtlasV2}}$'s multi-host scenarios. Nonetheless, the DrSec$_{\text{RoBERTa}}$ model trained on $\mathbf{D}_{\mathbf{AtlasV2}}$ gained competitive performance with DeepCase pre-trained on significantly more data from $\mathbf{D}_{\mathbf{Large}}$–52.38% vs. 56.99% PR AUC on $\mathbf{D}_{\mathbf{SOC}}$ and 71.98% vs. 72.31% AUC on $\mathbf{D}_{\mathbf{AtlasV2}}$'s multi-host scenarios. Since we trained it on non-sensitive data, we release the DrSec$_{\text{RoBERTa}}$ model pre-trained on $\mathbf{D}_{\mathbf{AtlasV2}}$ alongside the dataset to aid in reproducibility [51].

Rule name	*DrSec$_{\text{RoBERTa}}$*	*DrSec$_{\text{Doc2Vec}}$*	*TF⋆IDF*
Email run	64.87%	62.45%	51.85%
Key log. w/o UI	86.71%	72.00%	61.94%
Policy bypass	91.67%	90.82%	77.64%
Net server	92.57%	88.89%	79.63%
Non EXE run	81.10%	75.53%	72.48%
Office run	30.99%	18.11%	15.04%
Sys. config.	71.48%	64.08%	14.88%
Util. run	100.00%	100.00%	100.00%
Virus-like	92.12%	80.59%	19.19%
Web run	75.28%	66.29%	61.98%

TABLE 4: **PR AUCs for rule learning on $\mathbf{D}_{\mathbf{Rules}}$.** We report results for DrSec$_{\text{RoBERTa}}$, DrSec$_{\text{Doc2Vec}}$, and TF⋆IDF.

6.4. Security Application: EDR Rule Learning

To our knowledge, DrSec's deployment in EDRs is the first to leverage a hybrid model of rule- and learning-based detection, creating an opportunity for new security application paradigms. As an initial exploration in this direction, we set out to determine whether DrSec could create more generalized representations of expert rules that were more robust to subtle changes in attacker behavior. This approach, if possible, would offer the additional advantage of reducing the manual burden required by experts to tune rules to account for different exceptions and corner cases; instead, it would be possible to simply adjust the threshold on model outputs.

In this experiment, we fine-tuned the pre-trained models on process data that was labeled with EDR alerts, then asked them to predict whether new processes trigger an alert. Processes were organized by alert, with both the training and test splits featuring instances of each alert. We tuned the DrSec$_{\text{RoBERTa}}$ model for two epochs, evaluating validation accuracy after every 500 model updates, then selected the model with the highest validation accuracy. For DrSec$_{\text{Doc2Vec}}$, we used random-forest classifiers with the hyperparameters that did best on alert triage. The classifiers took the Doc2Vec representations as inputs, and predicted whether rules were fired or not for each process. We also tested TF⋆IDF features for rule learning, again using random-forest classifiers and the best-performing hyperparameters from the alert-triage experiments.

For each rule, we computed the PR AUCs obtained by the three systems. Table 4 summarizes the results. Despite facing a challenging imbalanced learning problem,

[2] We were unable to run the TTP model against the $\mathbf{D}_{\mathbf{AtlasV2}}$ dataset as TTP data was not available.

$\texttt{DrSec}_{\texttt{RoBERTa}}$ and $\texttt{DrSec}_{\texttt{Doc2Vec}}$ both performed well on rule learning, achieving 78.68% and 71.88% average AUCs across rules, respectively.[3] As expected, $\texttt{DrSec}_{\texttt{RoBERTa}}$ learned rules more accurately than $\texttt{DrSec}_{\texttt{Doc2Vec}}$, attaining higher or equal AUC for all ten rules. Additionally, both DrSec variants were more accurate than TF$\star$IDF (55.46% average AUC), consistently achieving higher or equal AUC. To our knowledge, this is the first experimental result demonstrating that EDR detection rules can be effectively represented within an ML model.

While the accuracy results are already impressive, it may be that the false positives include suspicious processes that *should* have triggered an alert but did not due to the brittle nature of rule-based detection. To characterize the false positive results, we inspected the top five false positives $\texttt{DrSec}_{\texttt{RoBERTa}}$ for the system configuration and web run rules. For the system configuration rule, three of the five FP's were incurred by an unknown, unsigned process named ids.exe. This process attempts to inject code into other processes via Windows' SetWindowsHookEx API). One false positive was caused by an unsigned program called tcpmain.exe that attempted to inject code into other processes and monitor keystrokes. The last false positive was issued for a PDF reader process that enabled executable memory, hooked other processes, passed commands to the service-control manager, and listened to keystrokes. For the web run rule four of five false positives were Internet Explorer attempting to enable executable memory, listed available processes, and invoked Excel or Acrobat Reader. While we are unable to retroactively determine whether these behaviors were attacks, 9 of the 10 surveyed false positives featured processes that were clearly engaging in suspicious behaviors that were consistent with the goal of the detection rule. Only the last web run false positive appeared to be a true "mistake" by $\texttt{DrSec}_{\texttt{RoBERTa}}$, as it was triggered by a Windows' file explorer process (i.e., explorer.exe) that had been running for six consecutive days and creating processes per user commands. While more verification is needed, these results provide intriguing anecdotal evidence that hybrid rule/learning detection systems may be able to detect suspicious unanticipated attack behaviors.

6.5. Security Analysis

By introducing ML tasks to EDRs, which historically are driven by rule-based detection, the potential for ML attacks must be considered. In testing, an adversary may attempt to force an incorrect classification by altering the behavior of malicious processes to resemble those of benign processes (i.e., mimicry). In training, an adversary may attempt to subtly and incrementally shift the distribution of normal behaviors (i.e., data poisoning) in order to force an incorrect classification in a security application. Here we assess DrSec's susceptibility to various attacks.

Evading Detection at Deployment As with other ML-based systems, DrSec may be misled by adversaries during deployment. To evaluate robustness, we launched evasion attacks against DrSec and other systems. In particular, building off Yang et al.'s work [73], we implemented an attack that induces misclassification. Given an event sequence, the attack iteratively modifies it to evade the model, performing two actions per iteration: (1) identifying the index of the event most associated with the ground truth (e.g., true or false alarms) via a model-interpretation technique [58]; and (2) mimicking the other class by introducing an event closely associated with the erroneous class (mined from the training set). To preserve functionality, the adversary cannot remove or modify existing events. We ran the attack for 100 iterations and measured the evasion success rate and the percentage of events added for successful evasion (i.e., attack budget). Lower success rates and higher attack budgets indicate a more robust model.

We measured robustness on the alert-triage task using $\mathbf{D_{SOC}}$, after tuning $\texttt{DrSec}_{\texttt{RoBERTa}}$, DeepCase, and the TF$\star$IDF model for 86.67% recall (corresponding to a 0.5 threshold on $\texttt{DrSec}_{\texttt{RoBERTa}}$). While we found that all models were vulnerable to evasion, $\texttt{DrSec}_{\texttt{RoBERTa}}$ was the most robust. Attacks against $\texttt{DrSec}_{\texttt{RoBERTa}}$ achieved a 98.84% success rate with a median attack budget of 16.67%. In contrast, attacks against DeepCase and the TF$\star$IDF model attained 99.61% and 99.22% success rate, with median attack budgets of 10.00% and 5.26%, respectively. Crucially, $\texttt{DrSec}_{\texttt{RoBERTa}}$ was markedly more robust than the baselines for true alarms, with an attack success-rate of 80.00% (compared to 92.82% and 86.67% against DeepCase and TF$\star$IDF, respectively) and a median budget of 83.73% (compared to 8.70% and 9.62% for DeepCase and TF$\star$IDF, respectively), rendering it more challenging to evade while hiding actual attacks. We discuss measures to further enhance DrSec's robustness in §7

Poisoning the Pre-trained Model Adversaries may seek to poison the pre-trained, foundation DrSec models–e.g., by introducing triggers leading to predictable representations [52]–to harm their performance. We expect such poisoning to be difficult in our envisioned deployment scenario due to the sheer scale of pre-training data. A single device submitting telemetry contributes <1.5 processes on average to $\mathbf{D_{Large}}$; thus, an adversary must control >60K devices to poison even 0.1% of the data (containing ∼91M processes). Still, prior work has shown that LMs may memorize individual training records (or parts thereof) [12], [59], thus potentially enabling adversaries to manipulate models with little control over training data. To counter such risk, one may leverage techniques developed to ensure sensor integrity, rendering it challenging for adversaries to submit arbitrary data [47]. Furthermore, large companies pre-training foundation models like DrSec can use trusted data (e.g., from employees) to ensure data integrity. As pre-training does not happen on live data, one may also sanitize data to further uphold data integrity. Finally, at the scale of our dataset, one can use theoretically-backed defenses with negligible utility loss (e.g., differentially private training [61]).

Poisoning Downstream Tasks While poisoning the foundation model is challenging, adversaries may still be

[3] Corresponding to 98.06% and 97.42% receiver-operating characteristic AUC, on average, for $\texttt{DrSec}_{\texttt{RoBERTa}}$ and $\texttt{DrSec}_{\texttt{Doc2Vec}}$, respectively.

able to poison downstream tasks. As labeled data used in the supervised tasks (e.g., alert triage) is quality controlled, unsupervised tasks such as process identification may be more susceptible to poisoning. We tested poisoning attacks against process identification, simulating an adversary that takes over a (popular) target process so that its reported telemetry becomes similar to a source process they aim to misclassify at test time. To form poisonous samples, we concatenated event sequences from the source and target processes, sampling a fraction α from the source and 1-α from the target. We sampled the event sequences by selecting the initial event and length at random while ensuring the poisonous sample's overall length is comparable to the source's training samples' lengths (within ×0.5 and ×2 of the average length). We repeated the process-identification experiment from §6.2 with $\texttt{DrSec}_{\texttt{RoBERTa}}$, increasing the number of poisonous samples by one at a time; we stopped when attacks succeeded or half of the training samples were poisonous. We used the most popular process as the target and selected source processes with varying popularity–two from each of the top 10% and 50% (high and medium popularity), and two from the bottom 10% (unpopular). The distances of the source processes' representations from the target's were roughly equal, with a 31.23 average Euclidean distance, whereas $\leq$15.85 distance was needed for identification in the benign setting (i.e., without attacks). We estimated attack success by whether the source processes were (mis)identified as the target, and stealth by the increase in the number of target process' training samples.

For relatively small α values, poisonous sequences fail to achieve substantial attack success. For α=0.10, only one of the six source processes were misclassified as the target. Increasing alpha improves attack success: four of six processes were misidentified as the target for $\alpha \in \{0.5, 0.9\}$. The attacks were least successful for popular source processes, failing for all α values. The attacks succeeded for the processes with the medium and low popularity, but were less stealthy as popularity increased–more poisonous samples were necessary for attack success for medium-popularity sources (9.18% increase in the target process' samples) compared to the least popular processes (4.08% increase). We discuss means to mitigate such attacks in §7.

7. Discussion

Addressing Endpoint Security Challenges (§2.2) Through the introduction of DrSec, we address the *high-dimensionality of event attributes* by instead summarizing process behavior in a single representation. While this does not directly aid human analysts, it does enable better tooling by providing a common core for downstream security tasks to address the *fractured ecosystem of security applications*. Common problems that analysts face, such as *alert fatigue (i.e., too many false positives)* or *missed detections (i.e., too many false negatives)*, can be address in-part by useful automation. As part of our evaluation of DrSec, we demonstrate that its distributed representations can be leveraged for downstream alert triage that can reduce the amount of time wasted by analysts (i.e., reducing the number of false alerts). Finally, for many security applications, there is a *limited labeled data*; given the DrSec foundation model, we demonstrate that downstream tasks can be trained to provide high-accuracy with a limited fraction of labeled data (as compared to a standalone model).

LMs for Interpreting Low-level Telemetry LMs have a demonstrated ability in NLP to capture semantics in a self-supervised manner (e.g., [65]). We leverage this ability to aid in the interpretation of sequences of low-level process events–constituting a language themselves (§5.1)—given the natural semantic gap between these events and downstream task properties (e.g., true or false alarms). In applying DrSec to process identification (§6.2), we directly use these representations to cluster processes in an unsupervised manner: $\texttt{DrSec}_{\texttt{RoBERTa}}$ identifies roughly two-thirds of unknown processes with a single prediction, and most mispredictions map to highly-related processes. From this behavior, we conclude that LMs can help bridge the challenging semantic gap. That said, a limitation of DrSec is its inability to adequately model multi-threaded and inter-host events. For example, sequences of events in multi-threaded processes are interleaved with events from other threads within the process. Thus, two neighboring events within a thread can be separated by many events within the process sequence. While it does not explicitly account for this separation, we believe that our transformer-based model handles it more gracefully than our Doc2Vec-based model due to architectural differences: the Doc2Vec-based model uses a fixed window size for tokens (five in our evaluation). In future work, given appropriate event annotations, we would like to explore: (1) how to best represent event sequences while accounting for behavior caused by multiple threads interacting; and (2) the resulting improvements to the accuracy of downstream applications.

Hardening Against Attacks The LMs employed by DrSec provide no intrinsic protections against evasion and poisoning. These threats can be mitigated by drawing on adversarial ML techniques. As we envision DrSec as a tool employed by EDR vendors or large enterprises, the sheer scale of telemetry data provides natural resistance to poisoning attacks targeting the foundation LMs. Additionally, DrSec's learning-based mechanisms could complement, rather than replace, rule-based EDR detection: since these rules should not be misled by distracting process activities, one can ensure that attacks are not missed due to the introduction of DrSec. Our experiments show that DrSec is more robust than other methods, potentially due to more faithfully deriving meaning from low-level events and reasoning over longer event sequences. However, as with other approaches, DrSec can still be misled by sufficiently powerful adversaries. To further harden DrSec, we can employ techniques such as adversarial training [35] and randomized smoothing [27]. Moreover, similarly to other ML applications, certain downstream applications built on top of DrSec may be susceptible to poisoning. If the task's training data originates from potentially untrusted sources, one should seek to ensure the application's integrity. Poisoning defenses, and chiefly ones that sanitize data before training [13], [62] to remove anomalous sequences produced with large α, could aid in doing so.

Accounting for Concept Drift For DrSec, it is important to handle concept drift; in particular with respect to changes in the input data distribution (i.e., process event sequences). First, this may require changes to the vocabulary due to how the language tokens are selected to support a concise, yet descriptive, vocabulary. For instance, one of the elements of the DrSec vocabulary encodes the TLDs for domains that the process connects to; when new, popular TLDs are introduced, the vocabulary may need to change accordingly. Additionally, endpoint sensors are continually evolving to capture more contextual information about process behavior (and thus improve threat analytics), which would need to be considered in the vocabulary. Given changes in the vocabulary over time, as well as the behavior of applications themselves, it is important to update the DrSec model. For large, foundation models such as DrSec, training can be time and resource intensive; training the $\texttt{DrSec}_{\texttt{RoBERTa}}$ model in our evaluation took seven days using one Nvidia V100 GPU for a single epoch, but this would grow with a more comprehensive dataset and larger number of epochs. As part of future work, it would be useful to explore how to support training from existing models while handling large changes in the vocabulary (e.g., software updates, new software products).

8. Related Work

Machine Learning (ML) in Security ML-based methods are widely deployed in various security tasks, such as malware detection [20], network-intrusion detection [2], alert triage [5] and vulnerability discovery [23]. Training supervised models for computer security is challenging due to the difficulty in curating large amounts of accurately labeled data. At the same time, unsupervised ML often lacks the sufficient accuracy needed for developing fully autonomous security systems. Researchers sometimes manually define features that the ML model would ingest (e.g., [75]). However, feature engineering is often expensive and limits the reusability of models across different tasks. DrSec addresses these problems through (1) limiting the amount of labeled data needed for training; (2) mitigating manual feature engineering by automatically learning distributed representations (i.e., features); and (3) creating foundation models that can be reused across different tasks.

Language Models (LMs) have been proposed for natural languages (e.g., [37], [65]), programming languages (e.g., [3], [18], [46]), and protein sequences (e.g., [30]) for years. They have enabled revolutionary technologies such as GPT-3 [11]—an LM that can produce naturalistic text for different tasks—and Copilot [22]—a tool used to improve programmers' productivity. Particularly appealing is certain pre-trained LMs' property as few-shot learners [11], i.e., their ability to accurately address downstream tasks with limited supervision. We leveraged this property in DrSec to address multiple security tasks with little to no supervision. In computer security, LMs have been used to predict attacks [53], understand attack evolution [54], detect malware [72], triage alerts [64], and more. Unlike DrSec, which can repurpose the LMs for multiple tasks, prior efforts used the LMs to address specific tasks.

Threat Detection and Alert Triage EDR's are the prevailing threat detection solution in industry, using a rule-based approach to generate security alerts by matching local process-level events against known malicious event sequences. Due to the limitations of local context, though, handcrafted rules are often error prone, leading to many false positives. To this end, some provenance-based systems construct causal graphs from process-level event sequences in order to triage alerts [25], [26], [38]. Other systems assist security analysts by correlating alerts via statistical or heuristic approaches [16], [24], [42], [63], [70]. SIEM tools in industry [8], [49] are based on event correlation. Data mining [7] has also automated alert triage based on past analysts' operation traces [76]. However, these methods require laborious accurate labeling of attack events.

In another thread of alert-triaging work, ML is often employed to distinguish true from false alarms [4], [28], [33], [43], [60], [64]. While some alert-triage systems rely on custom, manually developed features, like installed software indicators [43], other systems require fully labeled datasets [60], such as CIC-IDS2017 [50]. By contrast, DrSec automatically learns representations, and requires limited supervision or none at all. DeepCase [64] is likely the most related system to DrSec. Similarly to DrSec, DeepCase relies on an LM, with or without supervision, for alert triage. Yet, unlike DrSec, DeepCase's LM is not reusable across tasks. Moreover, we found that DeepCase is heavily reliant on high-level, easily interpretable events. Consequently, when trained on low-level process events, DeepCase's alert-triage performance was significantly lower than DrSec's.

9. Conclusion

This work proposed DrSec, a system that makes use of distributed representations learned by foundation LMs and makes significant progress toward unifying the fractured ecosystem of security applications. Using little to no labels, we showed that DrSec can tackle three applications—process identification, alert triage, and EDR-rule learning—with substantial success. In doing so, we demonstrated that DrSec's distributed representations can meaningfully summarize process behavior and help ameliorate central challenges faced by endpoint security products (e.g., the alert fatigue and missed detections). In the future, it would be interesting to investigate other applications domains that DrSec can help tackle (e.g., malware or anomaly detection [55], [75]), and extend it to address various real-world challenges, such as concept drift.

Acknowledgments We would like to thank Raghav Batta, Christophe Briguet, Lalit Jain, and Ivan Yang for helpful discussions and technical help.

References

[1] Advanced Threat Analytics. New Research from Advanced Threat Analytics. https://prn.to/2uTiaK6, 2019.

[2] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021.

[3] U. Alon, M. Zilberstein, O. Levy, and E. Yahav. Code2Vec: Learning distributed representations of code. In *Proc. POPL*, 2019.

[4] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu. ATLAS: A sequence-based learning approach for attack investigation. In *Proc. USENIX Security*, 2021.

[5] S. Amershi, B. Lee, A. Kapoor, R. Mahajan, and B. Christian. Human-guided machine learning for fast and accurate network alarm triage. In *Proc. IJCAI*, 2011.

[6] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. In *Proc. USENIX Security*, 2022.

[7] D. Barbará and S. Jajodia. *Applications of data mining in computer security*, volume 6. Springer Science & Business Media, 2002.

[8] D. Berman. What is SIEM? https://logz.io/blog/what-is-siem/, 2021.

[9] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] K. Broughton. Automated Incident Response: Respond to Every Alert. https://swimlane.com/blog/automated-incident-response-respond-every-alert/, 2019.

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Proc. NeurIPS*, 2020.

[12] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramer, and C. Zhang. Quantifying memorization across neural language models. In *Proc. ICLR*, 2023.

[13] C. Chen and J. Dai. Mitigating backdoor attacks in LSTM-based text classification systems by backdoor keyword identification. *Neurocomputing*, 452:253–262, 2021.

[14] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] CrowdStrike, Inc. https://www.crowdstrike.com/falcon-platform/.

[16] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proc. RAID*, 2001.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT*, 2018.

[18] S. H. Ding, B. C. Fung, and P. Charland. Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *Proc. IEEE S&P*, 2019.

[19] FireEye, Inc. How Many Alerts is Too Many to Handle? https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html, 2019.

[20] D. Gavriluţ, M. Cimpoeşu, D. Anton, and L. Ciortuz. Malware detection using machine learning. In *Proc. IMCSIT*, 2009.

[21] Gensim. models.doc2vec - Doc2vec paragraph embeddings. https://radimrehurek.com/gensim/models/doc2vec.html, 2022.

[22] GitHub. GitHub Copilot. https://copilot.github.com/, 2021.

[23] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier. Toward large-scale vulnerability discovery using machine learning. In *Proc. CODASPY*, 2016.

[24] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proc. USENIX Security*, 2007.

[25] W. U. Hassan, A. Bates, and D. Marino. Tactical provenance analysis for endpoint detection and response systems. In *Proc. IEEE S&P*.

[26] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates. NoDoze: Combatting threat alert fatigue with automated provenance triage. In *Proc. NDSS*, 2019.

[27] Z. Huang, N. G. Marchant, K. Lucas, L. Bauer, O. Ohrimenko, and B. I. Rubinstein. Certified robustness of learning-based static malware detectors. *arXiv preprint*, 2023.

[28] N. Hubballi and V. Suryanarayanan. False alarm minimization techniques in signature-based intrusion detection systems: A survey. *Computer Communications*, 49:1–17, 2014.

[29] Hugging Face. RoBERTa. https://huggingface.co/docs/transformers/model_doc/roberta, 2022.

[30] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[31] P. Koehn and R. Knowles. Six challenges for neural machine translation. In *Proc. NMTW*, 2017.

[32] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proc. ICML*, 2014.

[33] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng. Log2Vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proc. CCS*, 2019.

[34] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint*, 2019.

[35] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *Proc. ICLR*, 2018.

[36] Microsoft. Windows Security identifiers. https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers, Sep 2022.

[37] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. NeurIPS*, 2013.

[38] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. Holmes: Real-time APT detection through correlation of suspicious information flows. In *Proc. IEEE S&P*.

[39] MITRE. MITRE ATT&CK: Indicator Removal on Host: File Deletion. https://attack.mitre.org/techniques/T1070/004/, 2020.

[40] MITRE Corporation. MITRE ATT&CK. https://attack.mitre.org, 2019.

[41] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *Proc. ACSAC*.

[42] F. Pierazzi, S. Casolari, M. Colajanni, and M. Marchetti. Exploratory security analytics for anomaly detection. *Computers & Security*, 56:28–49, 2016.

[43] T. Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Proc. RAID*, 2004.

[44] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proc. NDSS*, 2019.

[45] M. Riley, B. Elgin, D. Lawrence, and C. Matlack. Target Missed Warnings in Epic Hack of Credit Card Data. https://bloom.bg/2KjElxM, 2019.

[46] B. Roziere, M.-A. Lachaux, L. Chanussot, and G. Lample. Unsupervised translation of programming languages. In *Proc. NeurIPS*, 2020.

[47] S. Saroiu and A. Wolman. Trusted sensors, 2014. US Patent 8,832,461.

[48] A. Schofield, M. Magnusson, and D. Mimno. Pulling out the stops: Rethinking stopword removal for topic models. In *Proc. EACL*, 2017.

[49] S. S. Sekharan and K. Kandasamy. Profiling siem tools and correlation engines for security analytics. In *Proc. WiSPNET*, pages 717–721, 2017.

[50] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proc. ICISSP*, 2018.

[51] M. Sharif, P. Datta, A. Riddle, K. Westfall, A. Bates, V. Ganti, M. Lentz, and D. Ott. DrSec's public data and model. https://github.com/mahmoods01/DrSec-Oakland-2024, 2024.

[52] L. Shen, S. Ji, X. Zhang, J. Li, J. Chen, J. Shi, C. Fang, J. Yin, and T. Wang. Backdoor pre-trained models can transfer to all. In *Proc. CCS*, 2021.

[53] Y. Shen, E. Mariconti, P. A. Vervier, and G. Stringhini. Tiresias: Predicting security events through deep learning. In *Proc. CCS*, 2018.

[54] Y. Shen and G. Stringhini. Attack2Vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *Proc. USENIX Security*, 2019.

[55] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. IEEE S&P*, 2010.

[56] G. P. Spathoulas and S. K. Katsikas. Using a fuzzy inference system to reduce false positives in intrusion detection. In *Proc. IWSSIP*, 2009.

[57] Splunk Inc. splunk. https://www.splunk.com, Last accessed August 2018.

[58] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *Proc. ICML*, 2017.

[59] K. Tirumala, A. Markosyan, L. Zettlemoyer, and A. Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. In *Proc. NeurIPS*, 2022.

[60] L. Tong, A. Laszka, C. Yan, N. Zhang, and Y. Vorobeychik. Finding needles in a moving haystack: Prioritizing alerts with adversarial reinforcement learning. In *Proc. AAAI*, 2020.

[61] F. Tramer and D. Boneh. Differentially private learning needs better features (or much more data). In *Proc. ICLR*, 2021.

[62] B. Tran, J. Li, and A. Madry. Spectral signatures in backdoor attacks. In *Proc. NeurIPS*, 2018.

[63] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proc. RAID*, 2001.

[64] T. van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna. DEEPCASE: Semi-supervised contextual analysis of security events. In *Proc. IEEE S&P*, 2022.

[65] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. NeurIPS*, 2017.

[66] VMware. VMware Carbon Black EDR. https://www.vmware.com/il/products/endpoint-detection-and-response.html.

[67] VMware. Carbon Black Cloud. https://www.vmware.com/products/carbon-black-cloud.html, Last accessed April 2022.

[68] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *Proc. CCS*, 2002.

[69] C. Wang, S. Ma, X. Zhang, J. Rhee, X. Yun, and Z. Hao. A hypervisor level provenance system to reconstruct attack story caused by kernel malware. In *Proc. SecureComm*, 2018.

[70] W. Wang and T. E. Daniels. A graph based approach toward network forensics analysis. *ACM Transactions on Information and System Security (TISSEC)*, 12(1):1–33, 2008.

[71] R. Winchester. What's in a name? TTPs in Info Sec. https://posts.specterops.io/whats-in-a-name-ttps-in-info-sec-14f24480ddcc, 2019.

[72] Z. Xu, X. Fang, and G. Yang. Malbert: A novel pre-training method for malware detection. *Computers & Security*, 111, 2021.

[73] P. Yang, J. Chen, C.-J. Hsieh, J.-L. Wang, and M. I. Jordan. Greedy attack and gumbel attack: Generating adversarial examples for discrete data. *The Journal of Machine Learning Research*, 21(1):1613–1648, 2020.

[74] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang. Automated IT system failure prediction: A deep learning approach. In *Proc. Big Data*, 2016.

[75] Z. Zhang, P. Qi, and W. Wang. Dynamic malware analysis with feature engineering and feature learning. In *Proc. AAAI*, 2020.

[76] C. Zhong, J. Yen, P. Liu, and R. F. Erbacher. Automate cybersecurity data triage by leveraging human analysts' cognitive process. In *Proc. HPSC*, 2016.

Appendix A. Alert Triage With Limited Labeled Data

Fig. 8 presents the PR curve for alert triage on $\mathtt{D}_{\mathtt{SOC}}$ when using only 50% of the training data for fine-tuning.

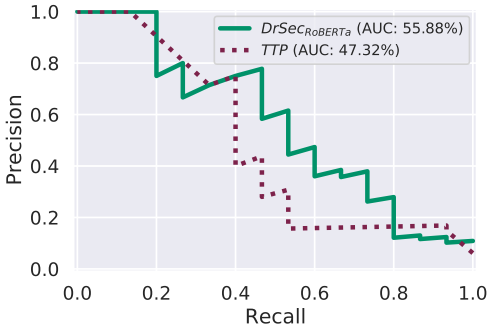


Figure 8: **Alert-triage PR curves on $\mathtt{D}_{\mathtt{SOC}}$, when training on 50% of the training split.** The AUCs are reported in the legend.

Appendix B. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

This paper introduces `DrSec`, a system that provides an intermediate representation of endpoint telemetry data for use in a variety of security applications. The central goal of `DrSec` is to ease the development of downstream security applications by introducing a unified intermediate layer based on a pre-trained language model, which can then be fine-tuned for specific tasks. The evaluation demonstrates that, compared to various baselines, `DrSec` provides better performance and remains robust against various evasion and poisoning attacks.

B.2. Scientific Contributions

- Addresses a Long-Known Issue.
- Provides a Valuable Step Forward in an Established Field.

B.3. Reasons for Acceptance

1) The paper addresses the longstanding problem of efficient event ingestion and processing for endpoint security systems. It shows that having a unified intermediate layer provides several advantages over current approaches and still has desirable security guarantees.
2) The paper introduces a radically different approach to endpoint security solutions. It shows that pre-trained foundation language models are a useful device for efficiently dealing with alert fatigue and missed detections as it relates to EDR.