Experimenting with neurosymbolic AI for defending against cyber attacks

Magnus Wiik Eckhoff [a,b], Jonas Halvorsen [a], Bjørn Jervell Hansen [a], Martin Eian [c], Vasileios Mavroeidis [b], Robert Andrew Chetwyn [b], Geir Skjøtskift [c] and Gudmund Grov [a,b,*]

[a] *Norwegian Defence Research Establishment (FFI), Norway*
E-mails: Magnus-Wiik.Eckhoff@ffi.no, Jonas.Halvorsen@ffi.no, Bjorn-Jervell.Hansen@ffi.no, Gudmund.Grov@ffi.no

[b] *Department of Informatics, University of Oslo, Norway*
E-mails: vasileim@ifi.uio.no, roberac@ifi.uio.no

[c] *mnemonic, Norway*
E-mails: meian@mnemonic.no, geir@mnemonic.no

Abstract. It is generally accepted that all cyber attacks cannot be prevented, which creates a need for the ability to detect and respond to cyber attacks. Both connectionist and symbolic approaches are currently being used to support such detection and response. This paper makes the case for combining the two using neurosymbolic AI. We identify a set of challenges faced when using AI today and propose a set of neurosymbolic use cases we believe are both interesting research directions for the neurosymbolic AI community and can have an impact on the cyber security field. We demonstrate feasibility through several experiments applying neurosymbolic AI to cyber security challenges. This paper is an extended version of a paper published at the NeSy 2024 conference [42]. The main additional contributions are further experimental evidence for our hypothesis that NeSy offers real benefits in this domain and a more in-depth treatment of knowledge graphs for cyber security.

Keywords: AI, neurosymbolic AI, cyber security, incident detection and response

1. Introduction

Protecting assets in the cyber domain requires a combination of *preventive measures*, such as access control and firewalls, and the ability to *defend* against cyber operations when the preventive measures were not sufficient.[1]

Our focus in this paper is on defending against offensive cyber operations, and before going into details, some concepts and terminology need to be in place:

Terminology & concepts The focus of this paper is to defend *assets* against *threats* in cyberspace. An *asset* can be anything from information or physical infrastructure and people to the internal processes of an enterprise. *Threats* manifest themselves in the form of *cyber operations* (or *cyber attacks*) conducted by an *adversary* (or *threat actor*). In our context of defending, the term *incident* is used for a potential attack that is deemed to have an impact on assets, and the process of defending comes under the area of *incident man-*

*Corresponding author. E-mail: Gudmund.Grov@ffi.no.

[1] It is generally accepted that preventive measures are not sufficient in cyberspace. An analogy is that we still need smoke detectors and a fire brigade, even if we take all possible preventive measures to reduce the risk of a fire.

agement [22]. This is typically conducted in a *security operations centre* (SOC), which consists of people, processes, and tools. One of the objectives of a SOC is to detect and respond to threats and attacks, where *security analysts* play a crucial role. Knowledge of threats in the cyber domain is developed by conducting intrusion analysis and producing and consuming *cyber threat intelligence* (CTI). Networks and systems to be protected are monitored, and *events* – e.g., network traffic, file changes, or processes executing on a host – are forwarded and typically stored in a *security information and event management* (SIEM) system, where events can be investigated, correlated and enriched, and queried. We will use the term *observations* for such events resulting from monitoring. Suspicious activity that is observed may raise *alerts*, which may indicate an incident that has to be analysed and responded to in the SOC. Finally, *Neurosymbolic AI* [37], which aims to combine connectionist and symbolic AI, will be abbreviated *NeSy*.

Why is a SOC relevant for NeSy? A SOC essentially conducts abductive reasoning by observing traces and identifying and analysing their cause in order to respond. This involves sifting through masses of events for suspicious behaviour, an area in which extensive research has been conducted for several decades using statistics and machine learning. Identifying the cause of observed suspicious behaviour requires situational awareness, achieved by combining and reasoning about different types of knowledge. There are various ways knowledge is represented, such as structured events and alerts, unstructured reports, and semantic knowledge [66, 100]. In a SOC, the ability to *learn* models to detect suspicious activities, and the ability to *reason* about identified activity from such models to understand their cause and respond, is thus required. These abilities are at the core of NeSy, and our hypothesis is:

A SOC provides an ideal environment to study NeSy with great potential for both scientific and financial impact.

Some early work has explored NeSy in the cyber security domain [28, 48, 51, 83, 88] and our goal with this paper, which extends [42], is to showcase the possibilities and encourage the NeSy community to conduct research in the SOC field with an emphasis on experiments.

Methodology: The identified SOC challenges are derived from a combination of existing published studies, the experience and expertise of the authors, and further discussions with SOC practitioners. The use cases result from reviewing NeSy literature in the context of the identified challenges, and the preliminary experiments conducted are based on a subset of the identified use cases.

Paper structure: In §2, we describe the typical use of AI in a SOC and the identified challenges. In §3, we make the case for NeSy and outline the NeSy use cases. In §4, we describe the proof-of-concept experiments, before we conclude in §5.

Contributions: This paper is an extended version of [42], published at the NeSy 2024 conference. We outline how AI is used today in a SOC and identify and structure a set of challenges faced by practitioners who use AI. We then create a set of promising use cases for applying NeSy in the context of a SOC, review current NeSy approaches in light of them, and demonstrate feasibility through proof-of-concept experiments. In this paper, we extend [42] with a more profound treatment of the use of knowledge graphs when defending against cyber attacks, and, crucially, we address the main limitation of [42] – limited experimental evidence – with the following experiments:

- Our experiment with *Logic Tensor Networks (LTN)* [12] in [42] is replaced with using LTN with a published ML model for network introduction detection [95], and extended with further experiments addressing explainability aspects and prioritising crucial knowledge.
- A NeSy technique called *Embed2Sym* [10] is explored to analyse and contextualise alerts from intrusion detection systems.
- Building on [19, 21], we explore the integration of large language models with a symbolic approach for *threat hunting*.
- We extend our previous work using data-driven enrichment of (symbolic) knowledge [101] with experiments using newly released data and explore the advantages NeSy provides for this challenge.

2. Challenges faced when using AI in a SOC

MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) [56] is a common reference model to structure the different phases when managing an incident.[2] For each phase of MAPE-K, we below discuss the common use of AI, including underlying representations, and identify key challenges security practitioners face when using AI.[3]

Monitor In the *monitor* phase, systems and networks are monitored and telemetry is represented as sequences of *events*. An *event* could, for instance, be a network packet, a file update, a user that logs on to a service, or a process being executed. Events are typically structured as key-value pairs. For a large enterprise, tens of thousands of events may be generated per second. In this phase, a key objective is to detect suspicious behaviours from events and generate *alerts*, which are analysed and handled in the later phases of MAKE-K.

This is a topic where machine learning (ML) has been extensively studied by training ML models on the vast amount of captured event data (see e.g. [5]). A challenge with such data is the lack of *ground truth*, in the sense that for the vast majority of events we do not know if they are benign or malicious. As most events will be benign (albeit we do not know which ones), one can exploit this assumption and use unsupervised methods to train anomaly detectors. This is a common approach. For at least research purposes, synthetic datasets from simulated attacks are also commonly used [58]. However, synthetic datasets suffer from several issues [9, 55] and promising results in research papers using synthetic data tend not to be recreated in real-world settings – whilst anomaly detectors often create a high number of false alerts[4] [17, 102]. Our first challenge, which follows from the *European Union Agency for Cybersecurity* (ENISA) [33], addresses this performance issue for ML models for real-world conditions:

Challenge 1. *Achieve optimal accuracy of ML models under real-world conditions.*

As benign software and malware are continuously updated, the notion of *concept drift* is prevalent, and ML models must thus be retrained regularly. Moreover, in addition to the large amount of data, requiring scalability, real-world conditions will have a large amount of noise (i.e. aleatoric uncertainty) in the data, which is not well reflected in synthetic data.

For previous incidents that have been handled, we know the ground truth of the associated alerts and events. Compared with the full set of events, this dataset will, however, be tiny. Still, it is important as it is labelled and contains data we know are relevant – either in terms of actual attacks experienced or false alerts that should be filtered out. One important challenge, also identified by ENISA [33], is the ability to exploit such labelled "incident datasets" and train ML models based on them:

Challenge 2. *Learning with small (labelled) datasets (from cyber incidents).*

New knowledge about certain threats, attacks, malware, or vulnerability exploits is frequently published (e.g. threat reports). The traditional, and still most common, method of threat detection is so-called *signature-based* detection, where such knowledge is (often manually) encoded as specific patterns (called signatures). Detection is achieved by matching events with these signatures and generating alerts. While signature-based methods have their limitations, such knowledge could improve the performance of ML-based detection models trained on event data, requiring the ability to extract relevant knowledge and include it in the ML models:

Challenge 3. *Extract knowledge (including about threats, malware, and vulnerabilities) and enrich ML-based detection models with it.*

In addition to reports, many knowledge bases and formal ontologies can be used to enrich ML models with such knowledge. There are also attempts to extend the coverage domain for such ontologies: one example is the *unified cybersecurity ontology* (UCO) [104] and another is the SEPSES knowledge graph [57]. To represent cyber

[2] Other common reference models are the OODA (Observe-Orient-Decide-Act) loop [15] and IACD (Integrated Adaptive Cyber Defense) framework [29].

[3] There has recently been a vast number of proposals for using *large language models* (LLMs) across MAPE-K. Here, we include what we currently consider the most promising uses of LLMs and refer to [14, 77] for a more complete discussion.

[4] As most events are benign, the base rate fallacy is important in this domain.

threat intelligence (CTI), a commonly used schema is *Structured Threat Information eXpression* (STIX) [81], with the associated *Threat Actor Context* (TAC) ontology [71]. A widely used knowledge base for threat actors and attacks is MITRE ATT&CK [76]. ML, and in particular *natural language processing* (NLP), is being explored for extracting CTI into symbolic forms (e.g. STIX) [70] or to map with MITRE ATT&CK [64]. *Large Language Models* (LLMs) are also explored for this topic [46, 65], with limitations identified [110]. However, we are not familiar with approaches to combine and integrate such knowledge with ML models for detection trained on events.

A different approach to identify malicious behaviour is *cyber threat hunting*. This is a *hypothesis-driven* approach where hypotheses are iteratively formulated (typically using CTI) and validated using event logs, as well as other knowledge [99]. Automating this process is our final challenge for the monitor phase:

Challenge 4. *Automated generation of hypotheses from CTI and validation of hypotheses using observations for threat hunting.*

There are AI-based approaches to facilitate threat hunting [79]. Most have focused on supporting hypothesis generation by extracting relevant CTI, and using both ML/NLP [36] and symbolic AI [91]. Symbolic AI has been used to support validation [20].

Analyse The goal of the analysis phase is to understand the nature of the observed alerts, determine possible business impact and create sufficient situational awareness to support the subsequent pland and execute phases.

Both malware and benign software continuously evolve. This makes it difficult to separate malicious from benign behaviour [33], despite continuous detection engineering efforts improving the capabilities. For example, an update to benign software may cause a match with an existing malware signature, and may also show up as an anomaly in the network traffic. As a result, most of the alerts are either false or not sufficiently important for further investigation [17]. The analysis phase is, therefore, labour-intensive, where security analysts must plough through and analyse a large number of alerts – most of them false – to decide their nature and importance. This could lead to so-called *alert fatigue* among security analysts:

Challenge 5. *The volume of alerts leads to alert flooding and alert fatigue in SOCs.*

Understanding the nature of alerts is important, and studies have shown that a lack of understanding of the underlying scores, or reasoning, behind the alerts have led to misuse and mistrust of ML systems [82]. Both studies [17] and guidance from ENISA [33] have highlighted the need for alerts to be reliable, explainable and possible to analyse. The use of explainable AI to support this has shown some promise [30], and both knowledge graphs [17] and LLMs[5] have been identified as promising approaches.

An alert is just one observation and needs to be put into a larger context to identify an *incident* and provide necessary *situational awareness* as a result of the analysis [34]. Such contextualisation includes enriching alerts with relevant knowledge from previous incidents, common systems behaviour, infrastructure details, threats, assets, etc. The same attack – or the same phase of a larger attack – is likely to trigger many different alerts. Different ML techniques, particularly clustering, have been studied to fuse or aggregate related alerts [59, 105]. In addition to understanding an incident and achieving situational awareness, contextualisation will also help a security analyst understand individual alerts. Similar to challenge 3, contextualization of alerts will necessarily involve extracting symbolic representation from a vast amount of available (and typically unstructured) information. A cyber attack conducted by an advanced adversary will, in most cases, manifest itself over several phases, creating a need to discover the relationships (between the alerts) across the different phases of an attack. A common reference model to relate such phases is the *cyber kill chain*, originally developed by Lockheed Martin and later refined into the *unified cyber kill chain* [89]. Other formalisms that enable modelling different phases of attacks include *MITRE Attack Flow* [75] and the *Meta Attack Language* (MAL) [52]. Different approaches have been studied to relate the different phases, including symbolic approaches [85], AI planning [7, 74], knowledge graphs [20, 60], state machines [108], clustering [44] and statistics [46]. However, this research topic is considerably less mature compared with ML models for detection in the monitor phase. We summarise the challenges of combining, understanding and explaining observations in the following challenge:

[5] E.g. Microsoft security co-pilot and an Elastic/LongChain initiative [1]

Challenge 6. *Combine observation with knowledge to analyse, develop and communicate situational awareness.*

Developing cyber situational awareness requires connecting a plethora of different sources, such as alerts and details about infrastructure and threats. There have been proposals to use knowledge graphs to combine these different sources to support analysis [66, 100], including explanation [17].

When an incident is understood and sufficient situational awareness is achieved, a suitable amount of resources have to be allocated to handle the incident. There may be multiple incidents requiring some prioritization between them. This involves understanding the risk and potential impact of the incident, including any mitigating actions that may be taken in subsequent MAPE-K phases:

Challenge 7. *Understanding the risk, impact, importance and priority of incidents.*

Plan & Execute The last two phases of MAPE-K, plan and execute, focus on responding to detected incidents. This involves finding suitable responses in the plan phase and preparing and executing the response(s) in the execute phase. From an AI perspective, research in these phases is not as mature as in the monitor and analyse phases. Therefore, we will only focus on the plan phase, which we currently consider to have more interesting AI-related challenges. To plan a suitable response, two promising AI techniques are *AI planning* (e.g. [40]) and *reinforcement learning* (RL – e.g. [50, 80]). Each of them has its pros and cons: AI planning requires considerable knowledge and formulation of the underlying environment, whilst reinforcement learning requires a considerable amount of interactions/simulations (often in the millions). In certain cases, a quick response time is necessary, which means this level of interaction would be too time-consuming. When generating response actions, their risk and impact must be taken into account (including the risk and impact of not acting), which is an unsolved problem when using AI. Moreover, when proposing a response action, an AI-generated solution must be able to explain both what the response action will do and why it is suitable for the given problem:

Challenge 8. *Generate and recommend suitable response actions in a timely manner that take into account both risk and impact and are understandable for a security analyst.*

To support such generation, there are several frameworks and formal ontologies that can be used, such as MITRE *D3FEND* [54], *RE&CT* [2] and *CACAO playbooks* [72].

Shared knowledge The 'K' in MAPE-K stands for *knowledge* shared across the phases, and we have, for instance, seen knowledge about threats and the infrastructure being protected used across different phases. Moreover, this knowledge takes different forms and representations (structured and unstructured) and is analysed using different techniques (symbolic and sub-symbolic). In addition to consuming knowledge, it is also important to share knowledge with key stakeholders, both technical and non-technical [32]. This may be a report about an incident for internal use (e.g. to board members) or sharing threat information with a wider community:

Challenge 9. *Generating suitable incident and CTI reports for the target audience.*

To summarise, we have shown the need to learn and reason across MAPE-K and that both symbolic and connectionist AI are being used across the phases. We have identified several challenges, and next, we make the case for NeSy to address them.

3. The case for neurosymbolic AI

Kahneman's [53] distinction between (fast) instinctive and unconscious '*system 1*' processes from (slow) more reasoned '*system 2*' processes, has often been used to illustrate the NeSy integration of neural networks (system 1) and logical reasoning (system 2). This interdisciplinary approach integrates neural networks adept at learning from vast amounts of unstructured data with symbolic representations of knowledge and logical rules to enhance the interpretability and reasoning capabilities of AI systems. Building on the above analogy, system 1 can, in a SOC, be seen as the ML-based AI used to identify potentially malicious behaviour in the monitor phase. Here, a large amount of noise needs to be filtered out from the large amount of events (thus a need for speed and scalability). System 2

is the reasoning conducted in the analysis phase, which requires deeper insight with the need for scalability less significant.

This dichotomy of requirements entails that neither end-to-end pure statistical nor pure logical approaches will be sufficient, and a NeSy combination seems ideal. Three commonly used reasons for pursuing NeSy are to design systems that are *human auditable and augmentable*, can *learn with less* and provide *out-of-distribution generalisation* [41]. We have seen examples of each of these in the challenges described in §2: the use of knowledge to contextualise, analyse and explain alerts; generate and explain response actions; learn from (relatively few) incidents; and handle concept drifts and noise in order to achieve high accuracy of ML models under real-world conditions.

From the challenges in §2, we will here outline a set of NeSy use cases we believe are promising and identify some promising NeSy tools and techniques for each of them. This work is not complete and should be seen as a start (see §5). Moreover, this section is speculative by nature, but we provide some evidence in terms of existing work and experiments conducted in §4.

Monitor The ability to integrate relevant knowledge into ML-based detection models (challenge 3) falls directly under the NeSy paradigm, and could both improve performance under real-world conditions (challenge 1) and help to reduce the number of false alerts (challenge 5):

Use case 1. *Use (symbolic) knowledge of threats and assets to guide or constrain ML-based detection engines.*

A similar case for such a NeSy use case is made in [88]. *Logical Neural Networks (LNN)* [94] are designed to simultaneously provide key properties of both neural nets (learning) and symbolic logic (knowledge and reasoning), enabling both logic inference and injecting desired knowledge (e.g. about threats and infrastructure) into the neural architecture. In *Logic Tensor Networks (LTN)* [12], a membership function for concepts are learned based on both labelled examples and abstract (logical) rules. LTN introduces a fully differentiable logical language, called *real logic*, where elements of first-order logic can be used to encode the necessary knowledge. LTN has been studied to detect suspicious behaviour [83] and is the topic of one of our experiments in §4.

In challenge 2, we highlighted the need to learn from (relatively small) datasets, which is one of the key features of NeSy [41]:

Use case 2. *Learn detection models from a limited number of (labelled) incidents*

Additional embedded knowledge in an LNN or LTN may help to reduce training time. *NS-CL* [69], which builds models to learn visual perception including semantic interpretation of the images, has shown it can be trained on a fraction of the data required by comparable methods – albeit in a different domain with different data sources. NeSy-based inductive logic programming variants, such as *∂ILP* [31], would also be able to learn from small datasets. The learned logic program will also be inherently explainable (see challenge 6).

Threat hunting involves generating suitable hypotheses, applying and validating them, then update and iterate (challenge 4). Work has started investigating LLMs for this challenge [87]. It has been argued for symbolism in LLMs [45], and based on that we define an LLM-based NeSy threat hunting use case:

Use case 3. *LLM-driven threat hunting using symbolic knowledge and reasoning capabilities.*

LLMs have been used for hypothesis generation in other domains [92], which can be further investigated for threat hunting. Hypothesis generation is typically driven by CTI, which can be captured in a knowledge graph. The integration of LLMs and knowledge graphs is an active research field [86]. In addition, symbolic or computational methods could be used for other steps in the hunting process, including: planning how to answer the hypothesis; reasoning about available data sources to execute this plan; ensuring correct translation to required query language[6] to validate the hypothesis using the observations; and finally, reason about the results from the execution and provide input for any refinement of the hypothesis for a new hunting iteration.

[6] Events are stored in a SIEM system, which will have a query language.

Analyse A prominent characteristic of NeSy is its capacity to combine learning and reasoning. Such a combination is desirable in a SOC, and our next use case, which cuts across the monitor and analyse phases, addresses several of the challenges from §2:

Use case 4. *Incorporate learning of detection models with the ability to reason about their outcomes to understand and explain their nature and impact.*

In [88], the case for such integration of detection and analysis using NeSy is also made. One way to achieve this is to simultaneously train a neural network (for detection) with related symbolic rules that can be used for contextualization, analysis and explanation (challenge 6). Two NeSy techniques that can accomplish this are *Deep Symbolic Learning (DSL)* [26] and *Neuro-Symbolic Inductive Learner (NSIL)* [25]. *dPASP* [39] and *NeurASP* [111] are techniques based on Answer Set Programming (ASP) [16], which seems promising for this use case. *dPASP* is based on furnishing ASP with neural predicates as interface to both deep learning components and probabilistic features in order to afford differentiable neurosymbolic reasoning. dPASP is suitable for detecting under incomplete information, abductive reasoning, analysis of competing hypotheses (ACH) [47], and what-if reasoning.[7] *NeurASP* improves the results from neural classifiers by applying knowledge-driven symbolic reasoning to them. This is achieved by treating the output from the neural classifier as a probability distribution over the atomic facts, which are then treated in ASP. The ASP rules can also be used to improve the training of the neural networks (use case 1). *DeepProbLog* [68] and *DeepStochLog* [109] incorporate reasoning, probability and deep learning, by extending probabilistic logic programs with neural predicates created from a neural classifier, and may thus provide the necessary combination of learning and reasoning for this use case.

Breaking use case 4 into smaller sub-cases, the first being extracting symbolic alerts, in order to support alert contextualization, analysis and explanation:

Use case 5. *Extracting alerts in a symbolic form.*

In [48], symbolic alerts are extracted from a *graph neural network* (GNN) based detection engine. A combination of *GNNExplainer* [112] and DL-Learner [63] is used to extract the symbolic alerts. The symbolic rules learnt by both DSL and NSIL may also provide such symbolic alert representation, and the use of e.g. *∂ILP* for detection will learn symbolic alerts by design. *Embed2Sym* [10] extracts latent concepts from a neural network architecture, and assigns symbolic meanings to these concepts. These symbolic meanings can then be used to encode symbolic alerts.

A SOC typically receives a large volume of threat intelligence, which is too large to thoroughly analyse manually. Such intelligence is used to contextualise alerts, and it is thus desirable to enrich the SOCs knowledge bases with relevant intelligence reports:

Use case 6. *Use statistical AI to enrich or extract symbolic knowledge.*

This use case addresses challenge 6. In §2, we discussed several approaches to extract knowledge in a suitable symbolic form from reports [64, 65, 70]. *STAR* [93] is a possible NeSy technique that can be used. It combines LLMs with ASP, where ASP can be employed to reason over the extracted knowledge, in addition to just extracting it.

This ability to reason is crucial as the intelligence report may be incorrect or superseded for different reasons, including underlying (aleatoric) uncertainty, deterioration over time, or from sources one does not fully trust. It may also simply not be relevant for our purposes, or more importantly, intelligence reports may conflict with our existing knowledge or our observations. It would therefore be desirable to quantify and reason about knowledge, including the level of trust, from both own observations and existing knowledge:

Use case 7. *Reason about and quantify knowledge.*

This use case is aimed at addressing challenge 7. It may play a role in implementing a technique known as *risk-based alerting* [103], which involves using data analysis to determine the potential severity and impact of alerts and incidents. *Probabilistic attack graphs* [43] has been used to add probabilities to CTI. One potential NeSy approach

[7] dPASP can leverage existing LLM-ASP integrations [93] to facilitate use case 3.

for this use case is *Recurrent Reasoning Networks* (RRNs) [49]. RRNs could be used to train a ML model from observations to reason about our existing knowledge graph, e.g. to quantify or identify inconsistencies. Another NeSy example is *Neural Probabilistic Soft Logic* (NeuPSL) [90], where the output from the trained neural networks is in (symbolic) *Probabilistic Soft Logic* (PSL) [11], which can be treated by probabilistic graphical models. NeurASP, dPASP, DeepProbLog and DeepStochLog may also be applicable here.

As discussed in §2, a cyber attack conducted by an advanced adversary will consist of multiple phases and the ability to relate these phases is essential when developing cyber situational awareness (challenge 6):

Use case 8. *Relate the different phases of cyber incidents.*

One concrete NeSy use case would be to merge the statistics- and semantics-driven approaches outlined in [8]. Further, *PyReason* [4] enables temporal reasoning over graphical structures, such as knowledge graphs. This can be used to exploit the temporal aspect of relating the different phases. The second experiment in §4 addresses this use case by exploring temporal reasoning using a combination of LLMs, temporal logic, ASP and plan recognition. The ontological reasoning supported by RRNs also seems promising for this type of problem.

Plan & execute *Neurosymbolic reinforcement learning* (NeuroRL) [3] combines the respective advantages of reinforcement learning and AI planning. NeuroRL can learn with fewer interactions compared with traditional RL by using inherent knowledge, thus making it more applicable than both RL and AI planning when (near) real-time response is required and a complete model of the environment is infeasible. Moreover, it has the promise of more explainable response actions, whilst a reasoning engine could, in principle, help to take into account both risk and impact[8]. Thus, this seems like promising approach for challenge 8:

Use case 9. *Generating impact and risk aware explainable response actions in a timely fashion using neurosymbolic reinforcement learning,*

Neurosymbolic reinforcement learning has been used in offensive cyber security settings for penetration testing[9] [28]. Whilst there are some commonalities with our challenges, defending has their own peculiarities. For example, speed, risk, impact and explainability are more prominent when defending against attacks.

Shared knowledge A widely applied form of symbolic AI in the context of cybersecurity is in semantic ontologies. Ontologies provide a formal and structured way of representing knowledge that both humans and machines can interpret while accounting for interoperability across systems. Built upon symbolic AI principles, ontologies focus on knowledge representation, logic, and reasoning, using well-defined structured models of the world. They define concepts, their composition, and their relationships within a domain, and they provide a clear distinction between the data (and the information itself) and the underlying model that defines how that information is organised, represented, and processed. This paradigm enables model evolution without data disruption, a known limitation of traditional relational models and other data serialization formats that inherently combine representations and data elements. This powerful paradigm allows for a more seamless integration of federated and siloed (in too many cases heterogeneous) data and can provide ensembles of contextual knowledge graphs in support of answering complex questions for decision-making. In addition, ontologies are the backbone of a knowledge base that can guide learning, ensure consistency, facilitate inference, and provide explainability, making neuro-symbolic systems more capable of handling real-world, knowledge-intensive tasks. Our final use case directly addresses challenge 9. Cyber threat intelligence is commonly shared in both structured and unstructured forms. LLMs are extensively studied for generating reports and this is also the case for cyber defence [77]. The generation process is likely to use symbolism (e.g. knowledge graphs [86]). The reports need to be correct, which is one area symbolic AI can help [45]. We, therefore, rephrase challenge 9 as a NeSy use case:

Use case 10. *Generation of incident reports and CTI reports tailored for a given audience and/or formal requirements, using (symbolic) knowledge and LLMs.*

[8] We note that there are additional challenges when generating risk and impact-aware responses, such as both deriving the requirements in the first place and representing them in a suitable way.

[9] Penetration tests are simulated attacks against the infrastructure and assets being protected, for instance, to identify vulnerabilities.

4. Proof-of-concept experiments

To showcase the feasibility of using NeSy for our use cases, we have conducted several experiments addressing different use cases:

- In §4.1 we address use case 1. This is shown by using Logic Tensor Networks (LTN) [12] to show how knowledge in symbolic form can be used to improve an ML-based detection engine as well as improving explainability.
- In §4.2 we address use case 8. Here, LLMs and ASP are used to elicit and reason with adversary attack patterns and observed alerts for situational awareness.
- In §4.3 we address use case 4, including elements from the (sub-) use cases 5, 7, and 8. Here, a NeSy solution based on the Embed2Sym [10] approach is explored to contextualize alerts. That is, we use ASP and formalized domain knowledge to label clusters of embeddings according to what cyber kill chain phase they are likely to represent.
- In §4.4 we address use case 3. Here, we build on [19, 21] by exploring the integration of LLMs with a symbolic approach based on knowledge graphs for threat hunting.
- In §4.5 we address use case 6 (with some elements of use case 8). Here, we extend our previous work using data-driven enrichment of (symbolic) knowledge [101] with experiments using newly released data and explore advantages NeSy provides for this challenge.

4.1. LTN for knowledge-aware intrusion detection

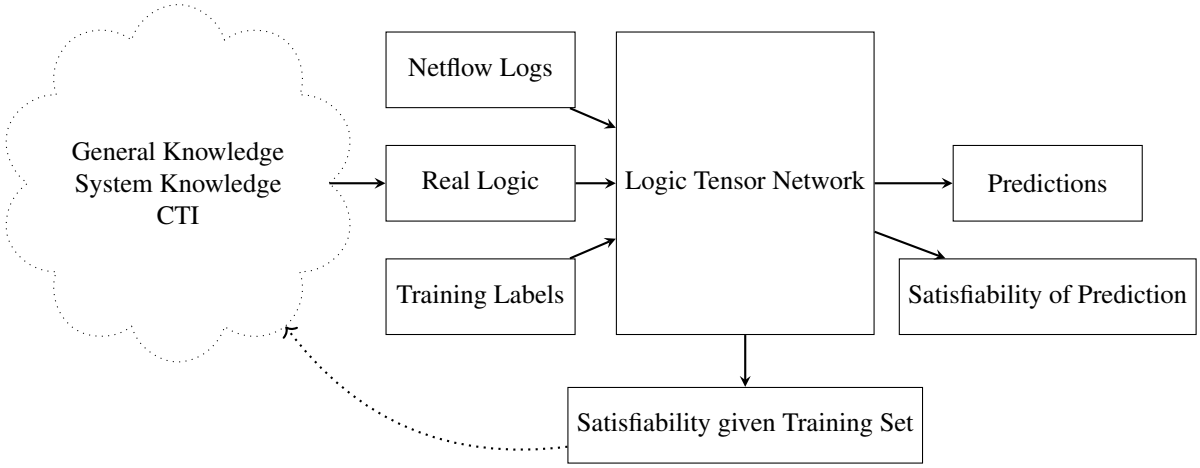


Fig. 1. Overview of LTN-based approach

An ML-based intrusion detection systems need to learn how to correlate data and their classes[10], capturing both simple and complex relationships. However, information that is not present or prevalent in the data might not be used, even if it is obvious to an analyst. For this reason, we use a Logic Tensor Network (LTN) [12] to learn from data while being guided by expressed knowledge. Most people intuitively know that a vulnerability in Microsoft Word is not a danger for machines without the software installed. Neural networks, on the other hand, need to learn this from seeing it repeatedly in the training data. Common sense knowledge like this can easily be expressed as logic statements, which are used to help guide the learning of the model. In intrusion detection, analysts often have information to help detect that is not expressed explicitly in the logs used by the detection engine. For example,

[10]Classes could, for instance, represent malicious or benign data – or specific attack steps or attack techniques. This depends on the type of classifier.

if a piece of software has a known vulnerability, this information might not be represented in the log data. Such additional knowledge can also be encoded into the LTN.

This experiment addresses use case 1 and is placed in the monitor phase of MAPE-K. Here, the goal is to detect malicious traffic by training LTN-based classifiers to detect two types of malicious traffic: brute force attacks and cross-site scripting (XSS) attacks. A *brute force attack* is a trial-and-error approach that, for instance, tries to guess the correct password, while *cross-site scripting (XSS)* attacks essentially inject malicious code into webpages.

This is achieved by training two different LTN-based classifiers: one classifier that separates brute force attacks from benign traffic and one classifier that separates XSS attacks from benign traffic.[11] Both classifiers use aggregated traffic in terms of *NetFlow* entries [23]. A NetFlow entry contains information about traffic between two distinct ports on distinct IP addresses for a given protocol within a given timeframe (which may vary). It will typically contain information about the number of packets and amount of data transferred, in addition to a large range of other features. For our experiment, we have used more than 80 different features.

The LTN-based intrusion detector will generate an alert if a NetFlow entry is classified as brute force or XSS. This alert will typically be manually analyst by a SOC analyst – or as we will see in other experiments below – further enriched by e.g. other NeSy approaches.

An overview of the approach can be seen in Figure 1. As the learning is supervised, the model takes the ground truth label for each NetFlow entry as input in addition to the NetFlow entries themselves. The main difference from a vanilla neural network is that we encode and provide knowledge that is hard or not possible to learn from the data in terms of *real logic* statements [96]. This could, for instance, be general knowledge, knowledge about the systems being protected or cyber threat intelligence (CTI) about the threat we are trying to detect. The statements are used in the classifier's symbolic part, while the labels and NetFlows are used in its neural part. The model's output is a predicted class for each NetFlow, along with information about whether the classification is in accordance with the provided statements. The LTN provides a classification with improved performance and explainability over a vanilla neural network.

The experiment consists of two parts: In the first part, a three-layered fully connected neural network is trained and used as a baseline. In the second part, a LTN with the same underlying neural network structure is enriched with additional knowledge [12]. In both cases, 70% of the data is used for training and 30% for testing. The experiment is inspired by [83], where LTN is used in a similar fashion to distinguish benign NetFlows from multiple classes of attack NetFlows. Our experiment takes this further by also encoding cyber security knowledge into the LTN and comparing a classifier with and without the knowledge.[12]

We use the CICIDS2017 dataset [97] for our experiment. This dataset simulates benign traffic and attacks over five days, varying the attacks performed each day. In our experiment, we use the subset called "Tuesday morning". The labelled flows are categorized into three classes: “Benign”, “Web Attack – Brute Force” and “Web Attack - XS”. The classes are significantly imbalanced, with $168,000$ flows in the benign class and $2,159$ flows in the remaining classes. An imbalance between benign traffic and attacks is common in intrusion detection. The dataset is partitioned into a $30/70$-split between the training set and the test set. To account for the class imbalance during training, we undersample the benign class to match the number of samples from the attack classes and the benign class. We include all features from the NetFlows into the sample except for the properties *source IP*, *destination IP*, and *source port* as they will cause over-fitting and not generalise well. We one-hot encode the *destination port* with the 22 most common ports being their own feature, while the remaining ports are regarded as “other ports”. We also one hot encode the *protocol* feature. All features are normalized using min-max normalization. Each NetFlow is represented as a vector of length 92. The LTN consists of one predicate for class membership:

$$P(x, class).$$

[11] We consider training two binary classifiers more realistic than a multi-class approach for this particular problem. However, note that we also did train a single multi-class network with comparable results.

[12] Similar work using LTN for intrusion detection has been in [13]. Note that this work was published after our initial LTN experiments [42], which we extend here.

This predicate is configured as a fully connected deep neural network with an architecture of 92 input features, two hidden layers of size 256, and an output layer of size one. ELU is used as the activation function of the hidden layers, while sigmoid is used for the output layer [24].

Real logic statements are used to shape the training of the neural network. The idea is that such statements should be created by a cyber security analyst and be based upon knowledge about the system, the current threat landscape, and any other relevant information the analyst has. Training consists of updating the neural network P to maximize the accumulated truth value of the axioms [12]. In this experiment, we first define the following axioms:

$$(1) \quad \forall x \in B : P(x, \text{Benign}) \quad (2) \quad \forall x \in BF : P(x, \text{Brute\_force}) \quad (3) \quad \forall x \in X : P(x, \text{XSS})$$

The first three axioms describe how all flows in the training set labeled as a given class (B : *benign*, BF : *brute force*, X : *xss*) should be a member of that class. This encodes the information of the baseline neural network with no additional knowledge. We then define the following axioms:

$$(4) \quad \forall x \in NWS : \neg(P(x, Brute\_force) \vee P(x, XSS)) \quad (5) \quad \forall x \in IT : \neg(P(x, Brute\_force) \vee P(x, XSS))$$

The fourth axiom describes how all NetFlows not going to or from a web server *(NWS: Not web server)* cannot be a web attack. Both the XSS and Brute force attack in this dataset are for web servers specifically. The information about what is not a web server is knowledge extracted from the topology of the network we are tasked to defend (i.e. the victim of the attack). The fifth axiom defines all traffic between machines in the victim network *(IT: Internal traffic)* as not part of an attack. This is because we expect all XXS or brute force attacks to come from outside of the organization's network. The experiments seek to answer to following research question:

> *A classifier enriched with knowledge will perform better and provide better insight into what has been learnedthan a purely data-driven classifier.*

We trained one baseline model and one LTN model for each of the two attack classes. Both the baseline and LTN use same training set and test set, and have the same configuration of the underlying neural network. We trained all models over 80 epochs with a batch size of 250. The results are presented in table 2. The results show that both solutions have a high recall when it comes to distinguishing benign traffic from attack traffic. Most importantly can we see that the precision of the LTN classifier is more than double the baseline classifier (0.154 vs 0.066 for brute force) (0.104 vs 0.028 for XSS) indicating that adding knowledge can improve classification.

	Baseline Neural Network			**Logic Tensor Network**		
Labels	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Brute Force	0.066	0.847	0.122	0.154	1.000	0.267
XSS	0.028	0.929	0.055	0.104	0.964	0.188

Fig. 2. Results from LTN Experiment.

Real logic statements in a LTN are an effective way of injecting knowledge into a neural classifier. They can also help in understanding and influencing the model's training and focus. Next we explore different ways the explainability aspects of LTN can be used by a SOC analysts.

During the training of the LTN, the goal is to maximize the aggregated truth of all the provided statements. This is done by deriving the loss of the model from the aggregated truth. Real logic is a fuzzy logic and we would not expect that all statements hold for all cases. After the training is completed, one can analyse how well the rules hold on all the provided training data to provide insights into how the model works. This can also be used as feedback to the analyst to help change or tweak the rules. In figure 3(a), we plot the satisfiability of the five statements on the training set. Here, we can see that, after training, rule four and five are generally satisfied. They are the rules that reduce false positives (false alerts) for the two attack classes. We can also see that the performance for the third rule, which classifies XSS, is significantly lower. This is in accordance with the results in figure 2.

When creating real logic statements there is a risk of creating a statement that does not accurately reflect the data. This can be the result of errors made when defining rules, or due to incorrect intuitions. For example, a rule asserting

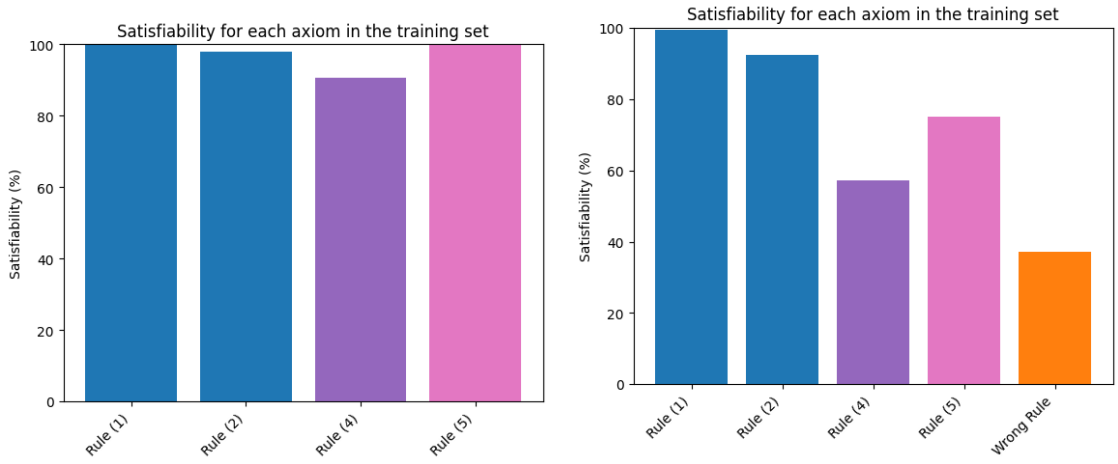
that only mobile phones are targets of attacks, does not accurately capture reality, as computers and other . If a bad rule is introduced we would expect the LTN would have a hard time satisfying this rule at the same time as the other rules. We can therefore use the low satisfiability of a rule (after training) as an indication that there is a problem with the rule. The fact that the LTN was not able to find a way to make the rule true, is an indication that it does not describe the data correctly. To demonstrate this, we conduct small experiment where we introduce a new rule that is obviously not true stating that all traffic labelled benign should be classified as brute force:

$$(6)\ \forall x \in B(P(x \in B, Brute\_force)).$$

After training an LTN with this incorrect rule, we can see in figure 3(b) that the satisfiability of the false rule is significantly lower than the other rules.

In addition to expressing rules, an analyst may be interested in expressing the relative importance of different rules. For example, a rule relating to a rare attack with limited consequences may be given a low priority. Conversely, a rule expressing something very prevalent and critical may be prioritised. To reflect this, we add weights to the rules to give a simple way of assigning the importance of a rule compared to the others, where a high weight results in the statement contributing more to the total aggregated truth.

To provide additional insights, the analyst can investigate the satisfiability of the different rules for a given NetFlow. For the majority of NetFlows, we expect all rules to hold, as this is what the LTN is optimizing for. However, if some rules are not satisfied, we could pass the information on to an analyst to provide additional explanation and context for their analysis.



(a) Standard rules

(b) Standard rules + wrong rule

Fig. 3. Average satisfiability on training set (after training)

To summarise, this experiment illustrates the potential of using NeSy to embed additional knowledge into ML models. We also see how LTNs can help analysts understand what the model learns and how it predicts. We have showcased that a neural classifier can be improved by adding knowledge in the form of an real value logic. When using the knowledge-enriched LTN, the number of false alerts was reduced without impacting recall. The LTN also provides multiple techniques to improve the model's explainability. By examining the satisfiability of the statements after training, we can gain valuable insight into what the model has learned or what it is not able to learn. With this information, we can change and improve the defined statements. The satisfiability can also be used to gain insight into the model's predictions. This shows promise for using NeSy to enrich ML-based models with (symbolic) knowledge.

4.2. LLMs and ASP for situational awareness

The previous experiment in §4.1 provided an example of how alerts can be raised, while we have addressed the need for supporting alert analysis in §2. In this experiment, we demonstrate such analysis by illustrating the use of NeSy to relate different phases of an attack (use case 8). Here, alerts sequenced by time are mapped to adversary attack patterns, gleaned from textual CTI reports into symbolic form using statistical methods (use case 6). The experiment is inspired by existing work such as: neurosymbolic plan recognition [6], attack plan recognition [7], and the use of LLMs to extract both *linear temporal logic* (LTL) [35] and CTI (in the form of MITRE ATT&CK *tactics* or *techniques*)[13] [46, 84, 113].

An LLM is first used to elicit formal representations of attack patterns described in CTI reports, affording us a rapid way of converting CTI to symbolic knowledge. Here, we use the `NL2LTL` Python library [35] to extract representations of attack patterns in LTL_f [27], a temporal logic for finite traces. We are using Open AIs GPT-4 model with few-shot learning. We define a custom pattern template *ExistenceEventuallyOther* to express the LTL property $\Diamond a \wedge \circ\Diamond b$. Each prompt parses two lines from the attack description and finds the appropriate ATT&CK technique (from the allowed symbols), and a LTL formula. We combine the formulas from each prompt into one long LTL formula expressing the whole attack pattern. The following is one prompt generated by NL2LTL:

Prompt generated by NL2LTL

Translate natural language sentences into patterns:
ALLOWED_PATTERNS: ExistenceEventuallyOther
ALLOWED_SYMBOLS: T1548 (Abuse Elevation Control Mechanism), T1552 (Unsecured Credentials), T1133 (External Remote Service), T1586 (Compromise account) [...]

Few shot learning example
NL: The adversary logs into the Kubernetes console.
This leads to: The adversary can view plaintext AWS keys in the Kubernetes console.
PATTERN: ExistenceEventuallyOther
SYMBOLS: T1133, T1552

Query
NL: Attackers leveraged spearphishing emails with malicious links to gain access to the system.
This Leads to: Attackers modifies the tty_tickets line in the sudoers file to gain root access.

A conceptual adversary attack pattern, sequencing MITRE ATT&CK techniques is visualized in Figure 4.

$$\begin{array}{ccccccccc} \{t1556\} & \ldots & \{t1059\} & \ldots & \{t1548\} & \ldots & \{t1059\} \\ \longrightarrow \bullet & \longrightarrow \ldots & \longrightarrow \bullet & \longrightarrow \ldots & \longrightarrow \bullet & \longrightarrow \ldots & \longrightarrow \bullet \\ t_i & \ldots & t_j & \ldots & t_k & \ldots & t_l \end{array}$$

$$\text{LTL}_f : \Box(\mathbf{I} \rightarrow \circ\Diamond(t1556 \wedge \circ\Diamond(t1059 \wedge \circ\Diamond(t1548 \wedge \circ\Diamond t1059))))$$

Fig. 4. Adversary attack pattern

Each '*txxx*', where *x* is a number, is a unique technique from the ATT&CK framework, $\mathbf{I}$ is the initial state, and $\Box$, $\circ$ and $\Diamond$ are the 'always', 'next' and 'eventually' operators in LTL. Next, `telingo`[18], an ASP solver for temporal programs, is used to postdict possible attacks. The `telingo` program encoding the problem is shown

[13] A MITRE ATT&CK *tactic* describes why an adversary performs an action, while a MITRE ATT&CK *technique* describes how the action is performed [76].

in Figure 6. The program details the sequences of observed alerts (lines 5–11), the LTL_f representations of known attack patterns (lines 14–21), and the relationships between alerts and ATT&CK techniques (lines 25–28).

The attack patterns in the program are acquired by the elicitation step described above, and the sequences of observed alerts are assumed to come from a SIEM system. That is, alerts produced are in a structured form amenable to be represented as Prolog/ASP terms. We assume that this conversion of alerts to symbolic form (use case 5) exists (see e.g. [48]). Furthermore, they are temporally ordered inducing a sequence of alerts (where a_x is an alert in symbolic form), as shown in Figure 5.

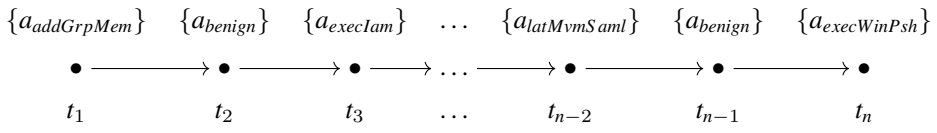


Fig. 5. Trace of alert observations

```
1  #program initial.
2  1 { plan(plan1;plan2) } 1 .
3
4  %Trace of alerts
5  &tel{&true
6       ;> alert(addGrpMem)
7       ;> alert(benign)
8       ;> alert(execIam)
9       ;> alert(latMvmSaml)
10      ;> alert(benign)
11      ;> alert(execWinPsh)}.
12
13 %Adversary attack plans
14 :- plan(plan1), not &tel{
15      >? techn(t1556)
16      & > (>? techn(t1059)
17      & > (>? techn(t1548)
18      & > (>? techn(t1059))))}.
19 :- plan(plan2), not &tel{
20      >? (techn(t1556)
21     & > (>? (techn(t1059) | techn(t1548))))}.
22
23 #program dynamic.
24 %abduce technique based on alert
25 1 {techn(t1556); techn(t1548)} 1 :- alert(addGrpMem) .
26 1 {techn(t1059)} 1 :- alert(execIam) .
27 1 {techn(t1548)} 1 :- alert(latMvmSaml) .
28 1 {techn(t1059)} 1 :- alert(execWinPsh) .
```

Fig. 6. `Telingo` program encoding the problem.

Finally, we assume that all the alerts produced can be associated with ATT&CK techniques, which is the case for many signature-based alerts. Note, however, that it is a many-to-many relationship: an alert can be an indicator for several techniques, and a technique can have several alert indicators. This knowledge[14] can be represented in ASP

[14]Extracted from alert rules found at https://github.com/SigmaHQ/sigma.

with choice rules, as illustrated below:

$$1\,\{t1556; t1548\}\,1 \leftarrow a_{addGrpMem}$$
$$1\,\{t1059\} \leftarrow a_{execIam}$$
$$1\,\{t1548\} \leftarrow a_{latMvmSaml}$$
$$1\,\{t1059\} \leftarrow a_{execWinPsh}$$

The outcome of the run in this experiment following the program in Figure 6, is that there are 2 stable models. This means that it is plausible that the input trace is an instance of the attack plan. Had there been no stable models, the conclusion would have been that this could not have been the case.

4.3. Neuro-symbolic alert contextualisation

Deciding which alerts are important and require attention, and understanding how they belong in the bigger picture, is essential in a SOC. This, however, requires contextualizing alerts with knowledge such as knowledge about the systems and networks in which the alerts were raised, cyber threat intelligence, and background knowledge accumulated by analysts over time. This is illustrated in Figure 7, where the context allows an analyst to follow a continuous path through alerts and log events.

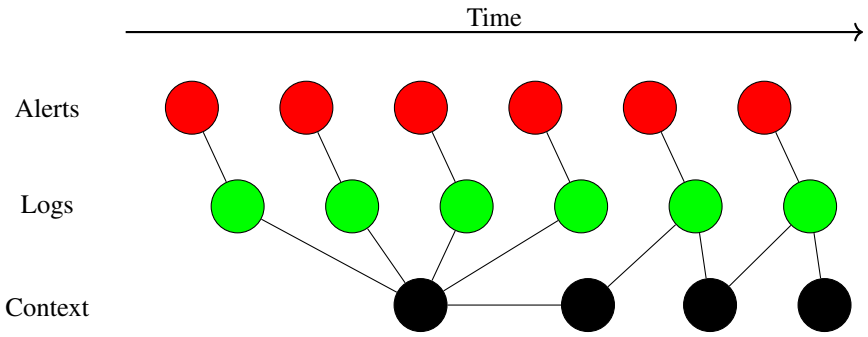


Fig. 7. Illustration of how context contributes to create a continuous path through logs and alerts.

In this experiment, we assume both rule-based and anomaly-based alerts, which differ widely in contextual richness, and we aim to classify these alerts by the cyber kill chain step to which they are likely to belong. The experiment mainly addresses use case 4, but also includes aspects of use cases 5, 7 and 8.

Rule-based alerts are generally contextually richer than anomaly-based alerts; the former are mostly hand-crafted and contextualized with descriptive knowledge as to what type of suspicious behaviour it detects (e.g. which MITRE ATT&CK technique the alert indicates), while the latter are generally more primitive alerts that flag any abnormal attribute values (that deviate from normally seen values during training). These are thus less descriptive with regards to what behaviour is detected. Hence, it is for example easier to associate cyber kill chain phases with rule-based alerts than with anomaly-based alerts. [15] On the downside, contrary to anomaly-based alerting, rule-based alerting is not able to detect novel and previously unseen suspicious behaviour. Both types of alerts are thus useful when detecting cyber attacks.

For this experiment

> *we wish to classify alerts according to cyber kill chain steps, yet we have alerts with a highly varying degree of contextual richness on which to do so.*

[15] We note, however, that a detection rule can be (and often is) an indicator of more than one TTP/cyber kill-chain phase and therefore non-trivial to uniquely identify it.

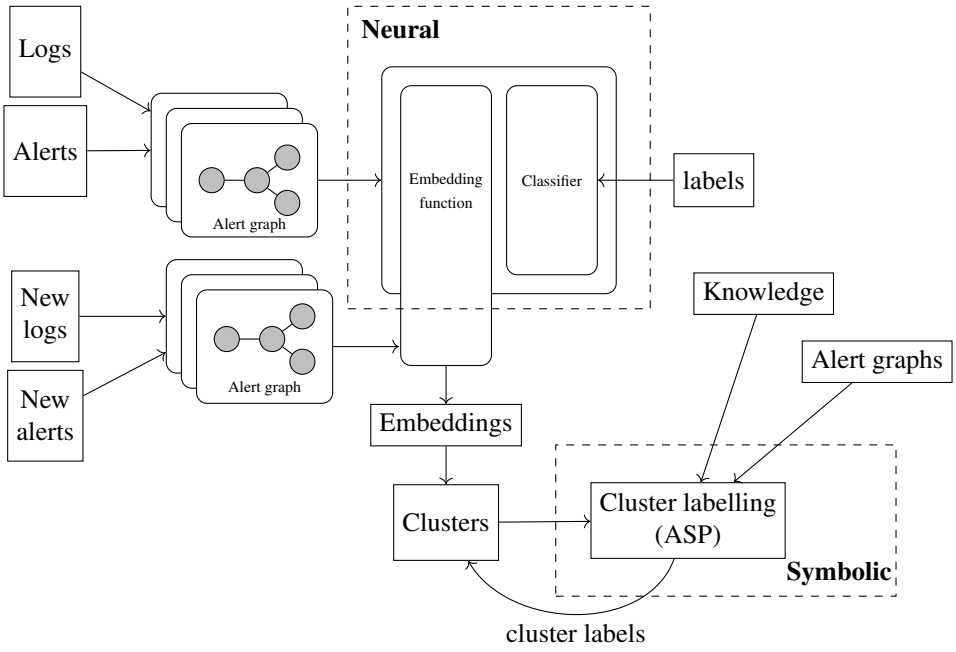


Fig. 8. Embed2Sym adapted to our experiment

The approach we explore in this experiment is that of using ASP and formalized domain knowledge to label clusters of alert embeddings according to what cyber kill chain phase they are likely to represent. Here, the embeddings are created using a neural component and then clustered into groups. The effect of this is that the clusters are likely to contain both descriptive and non-descriptive alerts. The task we formalize in ASP is essentially an optimization problem, where we use weak constraints to promote cluster labellings where:

a) The cyber kill chain step that an alert is classified as (i.e. the label assigned to the cluster group) should preferably comply with any domain knowledge about the detection rule.
b) Alerts from different cyber kill chain steps that share attributes (e.g. same users, or overlapping source and destination addresses) should preferably adhere to the temporal order dictated by the cyber kill chain.

Exemplifying (a), the cluster that contains an alert generated by a rule that detects MITRE ATT&CK technique T1548 should ideally be labelled *Lateral Movement*. Exemplifying (b), if $alert_1 \in cluster_1$ and $alert_2 \in cluster_2$, they both involve username *user*, and $alert_1$ happens before $alert_2$, then $cluster_1$ should ideally be labelled with a cyber kill chain stage that occurs before the label assigned to $cluster_2$, e.g. *Lateral Movement* before *Privilege Escalation*.

Turning to the technical details, we follow the approach presented in [10], referred to as *Embed2Sym*, where a neural perception and reasoning component is combined with a symbolic optimization component for extracting learned latent concepts. For the neural component, perception and reasoning is conducted in two separate stages. The latter stage (reasoning) is designed to solve a downstream task, whereas the perception stage creates vector embeddings of the input data. By solving the downstream task, the reasoning stage discovers structure in the data relevant to the domain. This is fed back to the perception stage, influencing the vector embeddings. Finally, the embeddings are clustered, and symbolic optimization using ASP is used to label the clusters according to the latent concepts.

This experiment is built upon logs and alerts from the dataset described in [61] and [62]. The logs and alerts are collected from a testbed emulating a small enterprise where a multi-step attack is being performed. The data also includes ground truth, making it possible to see exactly where in the logs and alerts the attack is captured and also what hostile activity gave rise to each of these log lines and alerts.

We transform this data into graph form, and we will henceforth refer to the graph representation of each log line as one *log graph* and the graph representation of each alert will be considered as one *alert graph*. The whole dataset, then, is represented as a union of the log and alert graphs. The latent concepts of interest are cyber kill chain stages, and the cyber kill chain stages in the experiment is based upon the stages used in the attack described in [61]:

1. Reconnaissance
2. Initial intrusion
3. Obtain credentials
4. Privilege escalation
5. Lateral movement

Our instantiation of Embed2Sym is shown in Figure 8. The neural processing is conducted in two stages:

1. First, training an embedding function (which would be the perception stage in Embed2Sym) and a classifier (reasoning stage in Embed2Sym).
2. Then, in the downstream task for the classifier, classify the alert log graphs according to hostile activity, utilizing the ground truth labelling in the data. Notice here that as each alert is related log lines, each alert graph also includes a log graph.

In the training of the neural processing these two stages interact, meaning that the trained embedding function is influenced by the classification stage.

For the next step in the process, we consider the set of alert graphs that we wish to analyse. Using the embedding function, these alert graphs are transferred into the embedding space, where they are clustered. The intuition behind this step is that the downstream task of classifying the alert graphs according to the actual hostile activity should force the embeddings of alert graphs from the same stages of the attack closer together in the vector space.

The final step is to apply symbolic reasoning, utilizing formalized domain knowledge, to label the clusters accordingly. The task is encoded as an ASP program, shown in Figure 10, and the clustered (alerted) logs as ASP facts as shown in Figure 9.

```
1 log_in_cluster(log1, cluster1).
2 happensAt(log1, 100000).
3 dst_host(log1,"10.0.0.1").
4
5 detected(log1, "T1000").
6 src_user(log1, "John").
7 dst_user(log1, "Jack").
8 src_host(log1, "192.168.0.10").
```

Fig. 9. ASP instance encoding

Starting with the encoding of the instance data, lines 1-3 in Figure 9 encode that a log belongs to a cluster, that it happened at a certain unix epoch time, and was logged on a specific host, respectively. Line 5 encodes that the alerted log was generated by a rule that detects instances of MITRE ATT&CK tactic T1000, while lines 6-8 encode the ip-address that initiated the process that was logged, the username was associated with the logged event, and the username that originally initiated the logged event (e.g. a *su* operation), respectively. Lines 1-3 encode facts that all logs will contain, while lines 5-8 depend on the underlying log event type (e.g. network traffic log or process execution log).

Proceeding to the encoding of the task itself, shown in Figure 10, the first part establishes some basic cyber kill chain and TTP domain knowledge. That is, lines 2-4 define cyber kill chain phases, and their relative ordering in the chain, while lines 6-10 map MITRE ATT&CK techniques to the cyber kill chain phases used in our experiment.

The next part introduces rules pertaining to the order and shared features of log events. The rule in line 13 captures the temporal order of log events, while lines 13-28 capture shared features between logs, such as sharing users, source and destination addresses etc.

The following part, lines 31-34 deals with defining what constitutes labels and clusters, while lines 37-38 are responsible for allocating labels. That is, the choice rule in line 37 ensures that each cluster is assigned a label,

```
% cyber kill chain / TTP domain knowledge
kill_chain_phase(latMvmt,5). kill_chain_phase(privEsc,4). kill_chain_phase(obtainCreds, 3).
kill_chain_phase(initialIntrusion,2). kill_chain_phase(recon,1).
kc_before(P, Ptac) :- kill_chain_phase(P,S), kill_chain_phase(Ptac,Stac), S < Stac.

phase_ttp_map(latMvmt,("T1021.004";"T1021")).
phase_ttp_map(privEsc,("T1548.003";"T1055";"T1078")).
phase_ttp_map(recon,("T1083";"T1595.002")).
phase_ttp_map(obtainCreds,("T1110";"T1110.001")).
phase_ttp_map(initialIntrusion,("T1190";"T1078")).

% temporal rule
happensBefore(Log, LogTac, before(T1,T2)) :- happensAt(LogTac, T2), happensAt(Log, T1), T1 < T2.

% rules establishing feature-based links
%% 1. if on same host, then two logs must a) share users OR b) share connection
%% 1.b
common_feature(Log1, Log2,same_source_dest(Dest,Source)) :- dst_host(Log1, Dest), dst_host(Log2, Dest),
        src_host(Log1,Source), src_host(Log2, Source), Log1 != Log2.
%% 1.a
common_feature(Log1, Log2,same_source_user(Dest,User)) :- dst_host(Log1, Dest), dst_host(Log2, Dest),
        dst_user(Log1,User), dst_user(Log2, User), Log1 != Log2.
common_feature(Log1, Log2,same_source_user(Dest,User)) :- dst_host(Log1, Dest), dst_host(Log2, Dest),
        dst_user(Log1,User), src_user(Log2, User), Log1 != Log2.
%% 2. if on different hosts, must be src->dst
common_feature(Log1, Log2,source_to_dest(Dest)) :- dst_host(Log1, Dest), src_host(Log2, Dest), Log1 != Log2.
%% symmetric
common_feature(Log1, Log2, Reason) :- common_feature(Log2, Log1, Reason). % symmetric

% Cluster labeling
cluster(X) :- log_in_cluster(_,X).
label(X) :- kill_chain_phase(X,_).
label(X) :- benign_class(X).
benign_class(benign).

%% Assign label to each cluster.
{assigned_label(Cluster, Label) : label(Label)} = 1 :-  cluster(Cluster).
:- assigned_label(Cluster1, Label), assigned_label(Cluster2, Label), Cluster1 != Cluster2.
log_label(Log,Label) :- log_in_cluster(Log,ClusterID), assigned_label(ClusterID,Label).

% Optimization constraints
%% Constraint a)
label_ttp_mismatch(Log,Label,T) :- detected(Log,T), log_label(Log, Label), not phase_ttp_map(Label,T).
:~ label_ttp_mismatch(Log,Label,T). [1@2,Log,Label,T]

%
%% Constraint b)
kill_chain_sequence_mismatch(S,Stac) :- common_feature(Log, LogTac, Just), kc_before(S,Stac), log_label(Log,S),
        log_label(LogTac,Stac), S != Stac, not happensBefore(Log,LogTac,_).
:~ kill_chain_sequence_mismatch(S,Stac). [1@1,S,Stac]

#show assigned_label/2.
```

Fig. 10. `ASP` program encoding the problem.

```
[...]
Answer: 6
assigned_label("recon",recon) assigned_label("benign",benign) assigned_label("privilege_escalation",privEsc)
assigned_label("cracking",obtainCreds) assigned_label("webshell",initialIntrusion) assigned_label("reverse_shell",latMvmt)
Optimization: 0 1
OPTIMUM FOUND

Models       : 6
  Optimum    : yes
Optimization : 0 1
Calls        : 1
Time         : 96.698s (Solving: 0.01s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 96.659s
```

Fig. 11. `Clingo` console output

and line 38 ensures that each cluster is only assigned a single label. Line 39 is a convenience rule that in practice classifies a log based on the assigned label of the cluster it belongs to. Finally, lines 43-50 capture the two weak constraints that encode the optimization tasks that is described in the beginning of this section.

For the experimental run itself, we clustered 1900 alerts from the dataset into 6 clusters (number of cyber kill chain steps plus one 'benign'). Of these 1900 alerts, 290 had information that associated them with MITRE ATT&CK tactics. For convenience, we identified the clusters with meaningful names in order to make validation easier (e.g. cluster2 was named "webshell"). We then ran the ASP encoding of the task and instance data through the Clingo ASP grounder and solver [38], which was able to classify the labels correctly, as shown in Figure 11.

Although this experiment was limited to detecting one specific instantiation of a cyber kill chain within a generated but realistic dataset, we believe that the results indicate that the approach is feasible for classifying alerts according to cyber kill chain steps even when contextual information regarding the alerts varies widely.

4.4. NeSy-driven threat hunting

The previous experiments have focused on intrusion detection and subsequent analysis of the raised alerts. This experiment focuses on a different approach to discovering malicious behaviour called threat hunting (see challenge 4). The experiment addresses use case 3 and explores the efficacy of leveraging LLMs to develop a taxonomy of behavioural indicators for the (symbolic) *indicators of behavior* (IOB) approach to threat hunting [19, 21]. This symbolic threat hunting approach utilises an ontology and semantic reasoning to infer a set of contextualised adversarial behaviors across a series of logged security event data. Whilst this IOB-concept has previously been demonstrated [19, 21], it lacks a taxonomy of behaviours and reusable IOB identifiers.

Natural language processing techniques have been utilised for extracting Indicators of Compromise (IOCs) from CTI Reports [67] and, more recently, extended to the utilisation of LLMs for extracting IOCs from CTI reports [106]. Here, we explore the implementation of LLMs for developing an IOB taxonomy. The utilisation of LLMs for generating detection logic from a conceptualised task has been demonstrated in industry [98]. Motivated by this, in this experiment we explore the automated development of rule-based reasoning into our taxonomy.

Threat hunting involves the generation of suitable hypotheses, followed by applying and then validating the hypotheses (see challenge 4). This experiment uses a scenario-driven approach to IOB development, where the scenario is a hypothesis of describing what an adversary, tool or general user is trying to achieve. For a given scenario, the LLM is tasked with generating a set of low-level behavioural indicators that analyse syntax, commands and other properties at a low-level of details and reason over these indicators to infer a higher level of abstraction.

A simple example of such a scenario is `notepad.exe` being launched by the Windows built in *system user*[16] and then establishing a network connection. This example is a set of low-level behaviours. By combining these individual behaviours, a higher level of abstraction can be inferred as an interactive tool being launched by the system user for the purpose of networking. We can chain this behaviour together with other inferred behaviours to get an even higher level of abstraction.

To semi-automate this scenario-driven approach to developing an IOB taxonomy, we require an LLM that can:

- Contextualise elements of the cybersecurity domain.
- Contextualise how adversaries behave.
- Generate behavioural scenarios and transform these scenarios into a chain of events.
- Contextualise how tools, systems or programs operate in a given scenario.
- Generate a set of reusable IOB identifiers.
- Relate low-level security events together to form a higher level of abstraction and context.
- Transform an IOB scenario into a symbolic representation.
- Create the detection logic for low-level events as SWRL[17] rules.
- Be both granular and descriptive to provide context commonly missing from MITRE ATT&CK technique procedures [21].

[16] https://learn.microsoft.com/en-us/windows/security/identity-protection/access-control/local-accounts
[17] https://www.w3.org/submissions/SWRL/

These requirements are a mix of concepts and subject areas, where tailored LLMs may struggle to fulfill some requirements and excel in others. As a results of this, we primarily focus on general-purpose LLMs rather than purpose-built LLMs. We compare the following models for their ability to semi-automate the development of an IOB taxonomy:

- GPT4-Omni
- GPT3.5-Turbo
- Llama 3.2-3b[18]
- SecurityLLM[19]

The first three are general purpose, while the last model is purpose-built for cyber security. Various GPT-models have been used in the cyber security domain for a variety of purposes[78], including payload generation for offensive security tasks, leveraging knowledge from the MITRE ATT&CK framework and detection engineering. Llama has been available for research use [73], and SecurityLLM is based on Llama with the intent to provide cyber security guidance [115], including threat hunting, cyber kill chains and MITRE ATT&CK. Both Llama 3.2-3b and SecurityLLM were run offline for this experiment.

The scale of symbolic security event data makes it inefficient to process line-by-line via an LLM, due to context size and memory limits. To illustrate, the symbolic event data file generated from a single node in an emulated attack scenario used for our experiment contains $490,248$ lines, and $24,161,441$ characters. Instead, we leverage the LLM to generate IOB scenarios with relevant rule based detection logic, which utilise a reasoning engine for decision making. We define a set of constraints through prompt instruction and by embedding tasks within the message to the model to generate these scenarios.

LLMs perform more accurately when prompts utilise chain-of-thought prompting [107] and least-to-most prompting [114], which we combine in a hybrid approach. Chain-of-thought prompting is used to create a set of behaviors, an abstract definition, a summary, detection logic and the semantics are different concepts. Least-to-most prompting is used to create sub-tasks for the model to maintain accuracy and reduce the risk of LLM hallucination. Each sub-task has an updated prompt instruction set. Each model is tested without a system prompt. The GPT-models are non-configurable, hosted by the OpenAI, and operated in the web browser. Therefore, no configurations are available to share. Llama and SecurityLLM are open and their configurations are included during assessment.

Each model has two tasks:

1. Create the IOB scenario.
2. Create the symbolic representation.

The prompt hand task specification is developed based on the requirements for symbolic threat hunting as elucidated above, and shown in figure 13. Here, prompt instructions (A) and (B) are used for generating IOB scenarios and sub-tasks, while (C) and (D) are used for the generation of symbolic representation.

GPT4-Omni was the best-performing model. Without a system prompt, it produced various IOB scenarios and transformed these into a taxonomy. Like most of the models, it arbitrarily chose an IOB ID rather than generating the same each time when a system prompt was not provided. The model worked best with a system prompt, producing consistent IOB scenarios that can be concatenated into a behavioural taxonomy. An example scenario output by GPT4-Omni can be seen in figure 13.

The scenario in figure 13 creates the top-level behaviour *H01 ("Data Exfiltration via Command Prompt")* and generates a set of associated behaviours for this scenario. This is the optimal output expected from the LLM. Unlike Llama, this model was capable of producing an SWRL rule-set based on the possible command examples present in the IOB scenario. However, without the contextualised prompt instructions, it would create its own taxonomy. Instead of using SWRLs built-in regular expressions[20] to trigger within the `commandLine` data property, it preferred to create a `commandLine` class and reason over a `hasCommand` relationship. After defining the scope in the prompt instruction and stating the task, it was capable of correctly generating the SWRL relationships. An example

[18] https://huggingface.co/meta-llama/Llama-3.2-3B
[19] https://huggingface.co/ZySec-AI/SecurityLLM
[20] https://www.w3.org/submissions/SWRL/#8

(A) Prompt Instruction for creating an IOB scenario

I want you to use a scenario-based threat hunting method for high-level abstractions.
All scenarios are post-compromise scenarios. The threat actor has access to the network.
Atomic level / Low level behaviors require a detection analytic.
A detection analytic analyses the data logged by Sysmon in the Sysmon schema.
Focus on the context that commands, parameters and syntax are trying to achieve.
Focus on the context of these commands, parameters and syntax as a chain of events.
Focus on Windows enterprise environments.
Provide an extensive taxonomy rather than a few examples.
Behaviors should focus on common behaviors associated with adversaries.
Summarise with a table at the end.

(B) Tasks Instruction for creating an IOB scenario

I want to create a taxonomy of adversarial behavioral indicators. Do not repeat the MITRE ATT&CK framework, I want to compliment it with more detailed levels of abstraction. This is a more detailed level of abstraction of what an adversary is trying to achieve over a chain of events.
For this task focus on behaviours related to command prompt and how and adversary can leverage command prompt for its goals. Focus on the relation command prompt has with other aspects of adversarial behaviours.
This taxonomy and the behaviors is a tiered approach where very low level atomic behaviors are sub behaviors of more abstract concepts. Several medium level behaviors may form a higher level abstraction which is the adversaries main goal.
Be detailed with the high level abstraction, model it as a scenario and how an adversary may achieve it through its sub behaviours. The high level abstraction should be a reasoned chain of sub behaviors.
Each behavior has a behavior ID value assigned to it. Prefix with a behavior category and suffix with a number value. Provide detection analytics for the low level behaviors where possible.
Summarise at the end with a table with the following: Behavior ID, Description, Commands where relevant and Detection Analytics

(C) Prompt Instruction for transforming an IOB detections into SWRL syntax

Use the SWRL syntax for reasoning rules.
Command Line is a dataproperty named "commandLine" in the ontology.
A process is a member of the "Image" class.
A process individual will have the data property "commandLine" and contain some string to check.
The ontology prefix is "threat".
Each Behavior ID is a class.

(D) Task Instruction for transforming an IOB detections into SWRL syntax

I will represent an example SWRL rule. Can you transform the example into an SWRL rule for each individual command where relevant.
$Image(?e) \wedge commandLine(?e, ?c) \wedge swrlb : contains(?c, "nc - u/server/share") \rightarrow LM - 1A(?e)$

Fig. 12. Prompts used for NeSy-driven threat hunting.

Behavior ID	Description	Commands
H01	Data Exfiltration via Command Prompt	
M01	Collect Sensitive Data	
L01	Search for Files of Interest	`dir /s *, dir C:\Users\*\Documents\*.doc*`
L02	Copy Files to Temporary Location	`xcopy C:\Users\*\Documents\*.doc* C:\Temp\`
L03	Compress Files for Exfiltration	`compress.exe /c /s C:\Temp\*.doc*`
M02	Stage the Data for Exfiltration	
L04	Encrypt Data for Safe Transmission	`cipher.exe /e C:\Temp\compressed_data.zip`
L05	Rename Files to Avoid Detection	`rename C:\Temp\compressed_data.zip C:\Temp\data.enc`
L06	Move Data to Staging Area	`move C:\Temp\data.enc \\StagingServer\Share`
M03	Exfiltrate the Data	
L07	Exfiltrate via HTTP	`curl -X POST -d @C:\Temp\data.enc http://example.com/upload`
L08	Exfiltrate via FTP	`ftp -n -s:ftp_commands.txt`
L09	Use Email for Exfiltration	`powershell Send-MailMessage -From "attacker@domain.com" -To "attacker@external.com" -Subject "Data" -Attachments "C:\Temp\data.enc"`

Fig. 13. An example scenario generated by GPT-4 Omni for generating an IOB taxonomy. This scenario is transformed into an OWL ontology and processed by a reasoning engine. SWRL reasoning logic is generated from the list of commands.

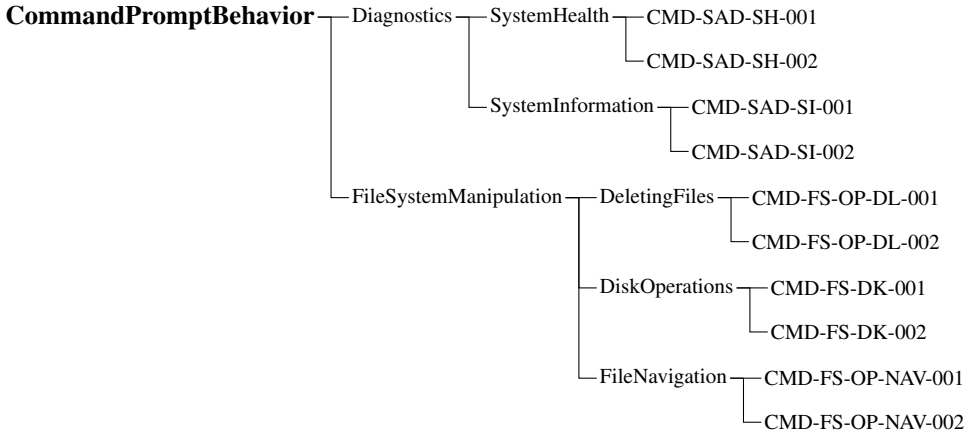


Fig. 14. Example class hierarchy of IOB Behaviors for the CommandPromptBehavior class. Each indent is a subclass. This class hierarchy was generated by GPT-4 Omni.

class hierarchy of IOBs can be seen in Figure 14. These behaviours and sub-behaviors were generated by the LLM, merged and output as an OWL2 schema[21].

The same limitations found in GPT4-Omni were also present in *GPT3.5*, and there was little variance between the findings for these two models. *Llama 3.2-3b* primarily used its default configuration.[22] When including a system prompt, Llama was able to perform the IOB scenario generation task and develop a general taxonomy of behaviours. However, the model is prone to inconsistencies with its output. Without a system prompt, the model

[21] https://www.w3.org/TR/owl2-primer/

[22] The following notable settings were used: 28 discrete model layers, three CPU threads, the temperature was 0.5 and the context length was 8, 192.

arbitrarily chooses its IOB identifier making repeatable scenarios challenging without human input. The model had challenges when generating SWRL, generating nonsensical URIs to domain concepts that didn't exist. It continuously created the URI `xmlns:swe="http://swe.stanford.edu/swe/"` and the annotation property `swe:Annotation property="swe:sweeprules"`, rather than a reasoning rule in the SWRL syntax. An example SWRL rule had to be provided in its task description to ensure the model consistently produced SWRL rules in the correct syntax. Once this was provided the model consistently output SWRL rules for any detection logic generated in the taxonomy task. An example output can be found in Figure 15

LM-1A SWRL Behavior Classification Rule:

$$\text{Image}(?i) \wedge \text{commandLine}(?i, ?c) \wedge \text{swrlb:contains}(?c, \texttt{"nc -u //server/share"}) \rightarrow \text{LM-1a}(?i)$$

LM-1B Behavior SWRL Classification Rule:

$$\text{Image}(?i) \wedge \text{commandLine}(?i, ?c) \wedge \text{swrlb:contains}(?c, \texttt{"ms14-064"}) \rightarrow \text{LM-1B}(?i)$$

DF-1A Behavior SWRL Classification Rule:

$$\text{Image}(?i) \wedge \text{commandLine}(?i, ?c) \wedge \text{swrlb:contains}(?c, \texttt{"Invoke-WebRequest -Uri https://..."}) \rightarrow \text{DF-1A}(?i)$$

Fig. 15. Example output of SWRL rule generation based on IOB scenario - Generated by Llama 3.2

SecurityLLM primarily used its default configuration[23], and performed the worst in this experiment. System prompts were not possible, which limited the ability to narrow the scope for the experiment. The model is capable of providing abstract definitions for IOB concepts and the process of developing a taxonomy of behaviours, but not any of the other tasks, regardless of how prompts and tasks were tweaked.

To summarize, except SecurityLLM, all models fulfilled all the requirements for developing an IOB taxonomy, but there were difficulties in handling symbolic aspects for all cases. Without prompt instructions defining the context and the ontology schema, each model would create its own taxonomy with variance between each of them. This variance makes it difficult to integrate when the concepts, rules and relationships vary each time. Once this context, constraints and rules were established, each model (except SecurityLLM) were capable of transforming the taxonomy into an OWL2 schema. The GPT-models were capable of creating SWRL rules without an example rule provided, wheras Llama had difficulties in understanding this context and tried to form its own ontology for rules.

This experiment bridges the gap between the lack of an IOB taxonomy [19, 21] and the symbolic approach to threat hunting, thus demonstrating the need for NeSy. The symbolic approach to threat hunting has previously shown that reasoning engines can infer complex adversarial behaviours [19, 21], but rely on user-defined rule-based reasoning. We have shown that LLM-models can aid in the semi-automation of IOB development and automating the transformation of IOB scenarios into symbolic representations amendable for such reasoning.

4.5. Data-driven enrichment of semantic kill-chain models

We have previously [101] applied data-driven enrichment of symbolic knowledge to help incident responders answer the questions:

- "*What did most likely happen prior to this observation?*"
- "*What are the adversary's most likely next steps given this observation?*"

One of the major issues we faced in this research was the lack of sufficient data on computer security incidents. MITRE Engenuity recently published the tool *Technique Inference Engine (TIE)*[24], which uses a recommender

[23]The following notable settings were used: 32 discrete model layers, three CPU threads, the temperature was 0.5 and the context length was 8,192.

[24]https://mitre-engenuity.org/cybersecurity/center-for-threat-informed-defense/our-work/technique-inference-engine/

model to infer a list of related techniques given a list of observed techniques. The dataset used to train the TIE model is available on Github[25], and covers more than $6,000$ computer security incidents. This dataset is significantly larger than the dataset used in [101]. We have conducted new experiments using the TIE data set, and we present the results of those experiments and explain the difference between our neuro-symbolic approach and TIE's purely data-driven method.

The available data for our method and TIE is a set of known incidents. Each incident contains an unordered set of MITRE ATT&CK techniques and sub-techniques. TIE uses this data to train a recommender model, which, when given a set of observed techniques as input, will output a set of techniques that most likely were used in the same incident. This gives incident responders guidance on what they should investigate, i.e. which techniques to look for. It does not cover the temporal aspect, i.e. what happened just before and after a specific observation of a technique. TIE's approach does not include symbolic knowledge – it is purely data-driven. To answer our two questions above, i.e. the most likely prior and next steps with the available data sets, a neurosymbolic approach is needed.

Our first step is the symbolic part, i.e., to formally model our knowledge of techniques. Every technique requires a set of abilities to be executed. Furthermore, every technique provides a set of abilities when executed. We developed a vocabulary of these abilities and mapped them to all the techniques and sub-techniques in ATT&CK. We then developed a tool[26], which when given a set of techniques and the mapped abilities as input, would output a set of stages with the techniques that are possible to execute at each stage. The stages represent a temporal ordering of the technique: A technique in stage $n+1$ depends on one or more techniques in stage n.

```
"tool_available": {
  "T1587.001": 83,
  "T1588.002": 181,
  "T1588.004": 16,
  "T1588.003": 27,
  "T1587.002": 9,
  "T1588.001": 37,
  "T1587.003": 18,
  "T1588": 45,
  "T1587": 48,
  "T1587.004": 7,
  "T1588.006": 7,
  "T1588.005": 8
}
```

Fig. 16. The number of times that techniques have provided the ability `tool_available`. In total (including the TIE data set), we observed 486 instances of a technique providing the ability `tool_available` to another technique observed in the same incident. In our original dataset, this ability had 100 observed instances. The most frequently observed technique is Obtain Capabilities: Tool (T1588.002) with 181 occurrences, which corresponds to a Markov chain transition probability of $p = 181/486 \approx 0.3724$.

Our second step was to apply the symbolic model to add temporal information to the dataset. For each incident in the dataset, we record each instance of an ability being provided by one technique in that incident to another technique that requires that ability in the same incident. We transform the dataset to a set of abilities, where each ability contains a set of techniques and a count of how many times we have observed that technique, provided that the ability to another technique exists in the same incident. Figure 16 shows an example of the technique counts for the ability "`tool_available`".

Finally, we implemented a tool[27] that uses the technique counts from the previous step to determine the transition probabilities of a Markov chain, as explained in [101]. We then used Markov chain Monte Carlo simulations to

[25] https://github.com/center-for-threat-informed-defense/technique-inference-engine/blob/main/data/combined_dataset_full_frequency.json
[26] https://github.com/mnemonic-no/provreq
[27] https://github.com/mnemonic-no/provreq-mcmc

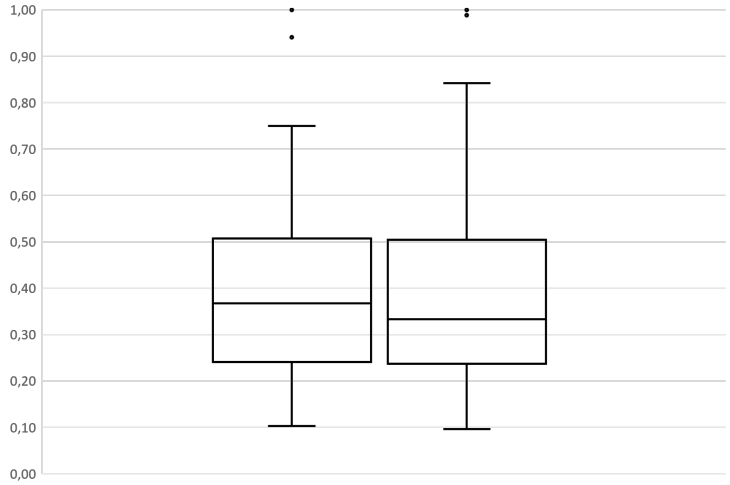


Fig. 17. Box plot with interquartile range 1.5, showing the original data set to the left and the TIE data set to the right. The data used to create the box plot is the highest Markov chain transition probability for each ability. The outliers close to 1.00 are abilities where either a single technique was observed or where a single technique had an overwhelming number of observation compared to the other techniques. The outlier techniques for the original data set were Process Discovery (T1057), Container and Resource Disvocery (T1613), and Gather Victim Org Information: Business Relationships (T1591.002), while the outlier techniques for the TIE data set were Process Discovery (T1057), Network Share Discovery (T1135), Modify Authentication Process: Network Device Authentication (T1556.004), and Gather Victim Org Information: Business Relationships (T1591.002).

determine the most likely technique prior to the observed technique. Our conclusions in [101] were that this approach is able to determine the prior technique with high probability, but if we try to determine a long attack chain, e.g. from observed exfiltration all the way back to initial access, then the most likely attack chain still has a very low probability. The example given for *Exfiltration over C2 Channel* (T1041) had a probability $p = 0.0202$, which is too low to be useful to an incident responder. In [101], we speculated that a larger dataset might improve the performance for long attack chains.

After running the same experiments on the TIE dataset, our results are similar. In one of the examples from [101], we see a clear improvement in the probability when we try to predict the prior technique: the most probable attack chain for the technique *User Execution* (T1204) had the probability $p = 0.6977$ in [101], while with the TIE data set this increased to $p = 0.8086$. In general, however, we see lower probabilities with the TIE dataset compared with the dataset in [101], e.g. with the *Exfiltration over C2 Channel* (T1041) example above. On closer examination, the reason for this result is that the TIE dataset is more varied than our original data set as it covers a much larger number of techniques, and the technique observations are more evenly distributed.

To illustrate the difference between the datasets, we extracted the maximum Markov chain transition probability for each of the abilities in the transformed dataset and created a box plot, shown in Figure 17. The plot shows that the TIE dataset has a lower median than the original data set, which means that in general a long attack chain generated from the TIE dataset will have a lower probability than one created from the original data set (used in [101]).

Our conclusions from [101] are still valid after testing our tools on the TIE dataset: we are able to answer the questions in the introduction. However, our remarks that the low probability of long attack chains due to a lack of training data are not valid. Attackers are different, and they use a varied set of techniques. Furthermore, new techniques are added to ATT&CK with each new revision. This leads us to conclude that our approach is unlikely to give useful results for very long attack chains, and that our tools should rather be used iteratively during incident response: predict the most likely prior step, investigate, and then repeat the process once the prior attack step is confirmed.

5. Conclusion

Our main goal with this paper has been to showcase and demonstrate through experiments the possibilities for NeSy in cyber security, focusing on problems within SOCs. We hope this will help stimulate a concerted effort in studying NeSy in this domain. The use of NeSy in cyber security is in its infancy, with some work having appeared over the last few years, including using NeSy for detection [83**?**], generating symbolic alerts [48] and extracting semantic knowledge from reports [70].

We have demonstrated that a considerable amount of symbolic and statistics-based AI is studied in SOC settings, and using it in real-world settings presents several challenges. We believe NeSy can address many of these challenges. Others have made some of the same points [51, 88], but not to the extent as we do here. We have contributed by defining a set of NeSy use cases and identifying promising NeSy approaches that serve as a starting point –several of them have been demonstrated in our experiments, which are the main new contributions of this paper compared with [42]. This work is just a start, and we both hope and expect that many new use cases and promising NeSy approaches that we have not covered here will appear in the not-too-distant future. The use cases and experiments provide a starting point for such future work. A challenge with AI in the cyber security domain is available datasets. Due to issues such as privacy, confidentiality and lack of ground truth, researchers tend to use synthetic data, which has their limitations [9, 55]. Furthermore, such datasets tend to focus only on detection (monitor phase), containing only events, and lack the additional (symbolic) knowledge which is important in SOCs and for our use cases. An important first step will be to develop synthetic datasets containing both events for detection and necessary knowledge in order to address the use cases. This can either be achieved by extending existing “detection datasets" [58] with the necessary knowledge or by developing new “NeSy datasets" from scratch.

Acknowledgements This work was partly funded by the European Union as part of the European Defence Fund (EDF) project AInception (GA No. 101103385). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union (EU). The EU cannot be held responsible for them.

References

[1] Elastic AI assistant. https://github.com/elastic/kibana/tree/main/x-pack/plugins/elastic_assistant.

[2] RE&CT. https://atc-project.github.io/atc-react/.

[3] Kamal Acharya, Waleed Raza, Carlos Dourado, Alvaro Velasquez, and Houbing Herbert Song. Neurosymbolic reinforcement learning and planning: A survey. *IEEE Transactions on Artificial Intelligence*, pages 1–14, 2023.

[4] Dyuman Aditya, Kaustuv Mukherji, Srikar Balasubramanian, Abhiraj Chaudhary, and Paulo Shakarian. PyReason: software for open world temporal logic. *In Proceedings of 2023 Spring Symposium on Challenges Requiring the Combination of Machine Learning and Knowledge Engineering (AAAI-MAKE 2023); arXiv preprint arXiv:2302.13482*, 2023.

[5] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021.

[6] Leonardo Amado, Ramon Fraga Pereira, and Felipe Meneguzzi. Robust neuro-symbolic goal and plan recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11937–11944, 2023.

[7] Adam Amos-Binks, Joshua Clark, Kirk Weston, Michael Winters, and Khaled Harfoush. Efficient attack plan recognition using automated planning. In *2017 IEEE symposium on computers and communications (ISCC)*, pages 1001–1006. IEEE, 2017.

[8] Andy Applebaum. Finding Dependencies Between Adversary Techniques. Presented at the FIRST 2019 conference. Available at https://www.first.org/resources/papers/conf2019/1100-Applebaum.pdf.

[9] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. Sok: Pragmatic assessment of machine learning for network intrusion detection. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 592–614. IEEE, 2023.

[10] Yaniv Aspis, Krysia Broda, Jorge Lobo, and Alessandra Russo. Embed2Sym: Scalable Neuro-symbolic Reasoning via Clustered Embeddings. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 421–431, 2022.

[11] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18(109):1–67, 2017.

[12] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic Tensor Networks. *Artificial Intelligence*, 303, 2022.

[13] Alice Bizzarri, Brian Jalaian, Fabrizio Riguzzi, and Nathaniel D Bastian. A neuro-symbolic artificial intelligence network intrusion detection system. In *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2024.

[14] Clint Bodungen. *ChatGPT for Cybersecurity Cookbook*. Packt Publishing, 2024.

[15] John R Boyd. The essence of winning and losing. *Unpublished lecture notes*, 12(23):123–125, 1996.

[16] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[17] Bushra A. Alahmadi and Louise Axon. 99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms. In *Proceedings of the 31st USENIX Security Symposium*, 2022.

[18] Pedro Cabalar, Roland Kaminski, Torsten Schaub, and Anna Schuhmann. Temporal answer set programming on finite traces. *Theory and Practice of Logic Programming*, 18(3-4):406–420, 2018.

[19] Robert Chetwyn, Martin Eian, and Audun Jøsang. Onto hunt - a semantic reasoning approach to cyber threat hunting with indicators of behaviour. pages 853–859, 09 2024.

[20] Robert Chetwyn, Martin Eian, and Audun Jøsang. Modelling indicators of behaviour for cyber threat hunting via sysmon. In *To appear in Proceedings of European Interdisciplinary Cybersecurity Conference (EICC 2024)*, pages 327–352, 2024.

[21] Robert Andrew Chetwyn, Martin Eian, and Audun Jøsang. Modelling indicators of behaviour for cyber threat hunting via sysmon. In *Proceedings of the 2024 European Interdisciplinary Cybersecurity Conference*, EICC '24, pages 95–104, New York, NY, USA, 2024. Association for Computing Machinery.

[22] Paul Cichonski, Tom Millar, Tim Grance, Karen Scarfone, et al. Computer security incident handling guide – revision 2. *NIST Special Publication*, 800(61):1–147, 2012.

[23] Cisco. Overview of netflow. Last accessed 03 April 2024.

[24] Djork-Arné Clevert. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[25] Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. Neuro-symbolic learning of answer set programs from raw data. In *International Joint Conference on Artificial Intelligence*, 2023.

[26] Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep symbolic learning: Discovering symbols and rules from perceptions. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23), Main Track*, pages 3597–3605, 2023.

[27] Giuseppe De Giacomo, Moshe Y Vardi, et al. Linear temporal logic and linear dynamic logic on finite traces. In *Ijcai*, volume 13, pages 854–860, 2013.

[28] Ryan Kerr Steven Ding and Li Li Adrian Taylor. Accelerating autonomous cyber operations: A symbolic logic planner guided reinforcement learning approach. In *Proceedings of the International Conference on Computing, Networking and Communications (ICNC 2024)*, pages 641–647, 2024.

[29] BK Done, KD Willett, DW Viel, GW Tally, DF Sterne, and B Benjamin. Towards a capability-based architecture for cyberspace defense. *2016, Concept Paper Approved for Public Release, US Department of Homeland Security, US National Security Agency Information Assurance Directorate, and the Johns Hopkins University Applied Physics Laboratory, AOS-16-0099*, 2016.

[30] Håkon Svee Eriksson and Gudmund Grov. Towards XAI in the SOC – a user centric study of explainable alerts with SHAP and LIME. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2595–2600, 2022.

[31] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[32] European Union Agency for Cybersecurity, Gert-Jan Bruggink, Maarten Toelen, Sergio Carrillo, and Vasileios Mavroeidis. *ENISA Threat Landscape Methodology*. European Union Agency for Cybersecurity, 2023. https://doi.org/10.2824/339396.

[33] European Union Agency for Cybersecurity, S Ntalampiras, C Pascu, M Barros Lourenco, G Misuraca, and P Rossel. *Artificial intelligence and cybersecurity research – ENISA research and innovation Brief*. European Union Agency for Cybersecurity, 2023. https://doi.org/10.2824/808362.

[34] Ulrik Franke, Annika Andreasson, Henrik Artman, Joel Brynielsson, Stefan Varga, and Niklas Vilhelm. Cyber situational awareness issues and challenges. In *Cybersecurity and Cognitive Science*, pages 235–265. Elsevier, 2022.

[35] Francesco Fuggitti and Tathagata Chakraborti. Nl2ltl–a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 16428–16430, 2023.

[36] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. Enabling efficient cyber threat hunting with cyber threat intelligence. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 193–204. IEEE, 2021.

[37] Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic ai: The 3 rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, 2023.

[38] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *CoRR*, abs/1705.09811, 2017.

[39] Renato Lui Geh, Jonas Gonçalves, Igor Cataneo Silveira, Denis Deratani Mauá, and Fabio Gagliardi Cozman. dPASP: A Comprehensive Differentiable Probabilistic Answer Set Programming Environment For Neurosymbolic Learning and Reasoning. *arXiv preprint arXiv:2308.02944*, 2023.

[40] Nirnay Ghosh and Soumya K Ghosh. A planner-based approach to generate and analyze minimal attack graph. *Applied Intelligence*, 36:369–390, 2012.

[41] Alexander Gray. IBM Neuro-Symbolic AI Workshop 23-27 Jan 2023. https://ibm.github.io/neuro-symbolic-ai/blog/nsai-wkshp-2023-blog/. Opening address.

[42] Gudmund Grov, Jonas Halvorsen, Magnus Wiik Eckhoff, Bjørn Jervell Hansen, Martin Eian, and Vasileios Mavroeidis. On the use of neurosymbolic ai for defending against cyber attacks. In *International Conference on Neural-Symbolic Learning and Reasoning*, pages 119–140. Springer, 2024.

[43] Andreas Gylling, Mathias Ekstedt, Zeeshan Afzal, and Per Eliasson. Mapping cyber threat intelligence to probabilistic attack graphs. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 304–311. IEEE, 2021.

[44] Steffen Haas and Mathias Fischer. Gac: graph-based alert correlation for the detection of distributed multi-step attacks. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 979–988, New York, NY, USA, 2018. Association for Computing Machinery.

[45] Kristian Hammond and David Leake. Large language models need symbolic ai. In *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning, La Certosa di Pontignano, Siena, Italy*, volume 3432, pages 204–209, 2023.

[46] Md Ariful Haque, Sachin Shetty, Charles A Kamhoua, and Kimberly Gold. Adversarial technique validation & defense selection using attack graph & att&ck matrix. In *2023 International Conference on Computing, Networking and Communications (ICNC)*, pages 181–187. IEEE, 2023.

[47] Richards J Heuer Jr. Analysis of competing hypotheses. *Psychology of intelligence analysis*, pages 95–110, 1999.

[48] Anna Himmelhuber, Dominik Dold, Stephan Grimm, Sonia Zillner, and Thomas Runkler. Detection, explanation and filtering of cyber attacks combining symbolic and sub-symbolic methods. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 381–388. IEEE, 2022.

[49] Patrick Hohenecker and Thomas Lukasiewicz. Ontology reasoning with deep neural networks. *Journal of Artificial Intelligence Research*, 68:503–540, 2020.

[50] Zhisheng Hu, Minghui Zhu, and Peng Liu. Adaptive cyber defense against multi-stage attacks using learning-based pomdp. *ACM Transactions on Privacy and Security (TOPS)*, 24(1):1–25, 2020.

[51] Brian Jalaian and Nathaniel D. Bastian. Neurosymbolic AI in Cybersecurity: Bridging Pattern Recognition and Symbolic Reasoning. In *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, pages 268–273, 2023.

[52] Pontus Johnson, Robert Lagerström, and Mathias Ekstedt. A meta language for threat modeling and attack simulations. In *Proceedings of the 13th international conference on availability, reliability and security*, pages 1–8, 2018.

[53] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[54] Peter E Kaloroumakis and Michael J Smith. Toward a knowledge graph of cybersecurity countermeasures. *The MITRE Corporation*, 11, 2021.

[55] Anthony Kenyon, Lipika Deka, and David Elizondo. Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets. *Computers & Security*, 99:102022, 2020.

[56] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[57] Elmar Kiesling, Andreas Ekelhart, Kabul Kurniawan, and Fajar Ekaputra. The sepses knowledge graph: an integrated resource for cybersecurity. In *International Semantic Web Conference*, pages 198–214. Springer, 2019.

[58] Ilhan Firat Kilincer, Fatih Ertam, and Abdulkadir Sengur. Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Computer Networks*, 188:107840, 2021.

[59] Igor Kotenko, Diana Gaifulina, and Igor Zelichenok. Systematic literature review of security event correlation methods. *IEEE Access*, 10:43387–43420, 2022.

[60] Kabul Kurniawan, Andreas Ekelhart, and Elmar Kiesling. An att&ck-kg for linking cybersecurity attacks to adversary tactics and techniques. In *International Semantic Web Conference (ISWC) - Posters and Demos*, October 2021.

[61] Max Landauer, Florian Skopik, Maximilian Frank, Wolfgang Hotwagner, Markus Wurzenberger, and Andreas Rauber. Maintainable log datasets for evaluation of intrusion detection systems. *IEEE Transactions on Dependable and Secure Computing*, 20(4):3466–3482, 2022.

[62] Max Landauer, Florian Skopik, and Markus Wurzenberger. Introducing a new alert data set for multi-step attack analysis. In *Proceedings of the 17th Cyber Security Experimentation and Test Workshop*, pages 41–53, 2024.

[63] Jens Lehmann. Dl-learner: learning concepts in description logics. *The Journal of Machine Learning Research*, 10:2639–2642, 2009.

[64] Zhenyuan Li, Jun Zeng, Yan Chen, and Zhenkai Liang. AttacKG: Constructing technique knowledge graph from cyber threat intelligence reports. In *European Symposium on Research in Computer Security*, pages 589–609. Springer, 2022.

[65] Jiehui Liu and Jieyu Zhan. Constructing knowledge graph from cyber threat intelligence using large language model. In *2023 IEEE International Conference on Big Data (BigData)*, pages 516–521. IEEE, 2023.

[66] Kai Liu, Fei Wang, Zhaoyun Ding, Sheng Liang, Zhengfei Yu, and Yun Zhou. A review of knowledge graph application scenarios in cyber security. *arXiv preprint arXiv:2204.04769*, 2022.

[67] Zi Long, Lianzhi Tan, Chaoyang He, and Shengping Zhou. Collecting indicators of compromise from unstructured text of cybersecurity articles using neural-based sequence labelling. *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[68] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

[69] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*. International Conference on Learning Representations, ICLR, 2019.

[70] Francesco Marchiori, Mauro Conti, and Nino Vincenzo Verde. STIXnet: A novel and modular solution for extracting all STIX objects in CTI reports. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, pages 1–11, 2023.

[71] Vasileios Mavroeidis, Ryan Hohimer, Tim Casey, and Audun Jøsang. Threat actor type inference and characterization within cyber threat intelligence. In *2021 13th International Conference on Cyber Conflict (CyCon)*, pages 327–352, 2021.

[72] Vasileios Mavroeidis and Mateusz Zych. Cybersecurity playbook sharing with stix 2.1. *arXiv preprint arXiv:2203.04136*, 2022.

[73] Meta. Introducing LLaMA: A foundational, 65-billion-parameter language model — ai.meta.com. https://ai.meta.com/blog/large-language-model-llama-meta-ai/. [Accessed 06-11-2024].

[74] Doug Miller, Ron Alford, Andy Applebaum, Henry Foster, Caleb Little, and Blake Strom. Automated adversary emulation: A case for planning and acting with unknowns. *MITRE CORP MCLEAN VA MCLEAN*, 2018.
[75] MITRE. Attack flow. https://github.com/center-for-threat-informed-defense/attack-flow.html.
[76] Mitre. Mitre ATT&CK. https://attack.mitre.org/.
[77] Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, Pejman Najafi, Feng Cheng, and Christoph Meinel. Large language models in cybersecurity: State-of-the-art. *arXiv preprint arXiv:2402.00891*, 2024.
[78] Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, Pejman Najafi, Feng Cheng, and Christoph Meinel. Large language models in cybersecurity: State-of-the-art, 2024.
[79] Boubakr Nour, Makan Pourzandi, and Mourad Debbabi. A survey on threat hunting in enterprise networks. *IEEE Communications Surveys & Tutorials*, 2023.
[80] Jakob Nyberg, Pontus Johnson, and András Méhes. Cyber threat response using reinforcement learning in graph-based attack simulations. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–4. IEEE, 2022.
[81] OASIS. Introduction to STIX. https://oasis-open.github.io/cti-documentation/stix/intro.html.
[82] Sean Oesch, Robert Bridges, Jared Smith, Justin Beaver, John Goodall, Kelly Huffer, Craig Miles, and Dan Scofield. An assessment of the usability of machine learning based tools for the security operations center. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 634–641, 2020.
[83] Darian M Onchis, Codruta Istin, and Hogea Eduard-Florin. Advantages of a neuro-symbolic solution for monitoring IT infrastructures alerts. In *2022 24th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 189–194. IEEE, 2022.
[84] Vittorio Orbinato, Mariarosaria Barbaraci, Roberto Natella, and Domenico Cotroneo. Automatic mapping of unstructured cyber threat intelligence: An experimental study:(practical experience report). In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 181–192. IEEE, 2022.
[85] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. Mulval: A logic-based network security analyzer. In *USENIX security symposium*, volume 8, pages 113–128. Baltimore, MD, 2005.
[86] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024.
[87] Filippo Perrina, Francesco Marchiori, Mauro Conti, and Nino Vincenzo Verde. Agir: Automating cyber threat intelligence reporting with natural language generation. In *2023 IEEE International Conference on Big Data (BigData)*, pages 3053–3062, 2023.
[88] Aritran Piplai, Anantaa Kotal, Seyedreza Mohseni, Manas Gaur, Sudip Mittal, and Anupam Joshi. Knowledge-enhanced neurosymbolic artificial intelligence for cybersecurity and privacy. *IEEE Internet Computing*, 27(5):43–48, 2023.
[89] Paul Pols and Jan van den Berg. The unified kill chain. *CSA Thesis, Hague*, pages 1–104, 2017.
[90] Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Yang Wang, and Lise Getoor. Neupsl: Neural probabilistic soft logic. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4145–4153. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Main Track.
[91] Sara Qamar, Zahid Anwar, Mohammad Ashiqur Rahman, Ehab Al-Shaer, and Bei-Tseng Chu. Data-driven analytics for cyber-threat intelligence and information sharing. *Computers & Security*, 67:35–58, 2017.
[92] Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, et al. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. *arXiv preprint arXiv:2310.08559*, 2023. To appear in the Twelfth International Conference on Learning Representations (ICLR 2024).
[93] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar, and Gopal Gupta. Reliable Natural Language Understanding with Large Language Models and Answer Set Programming.
[94] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, and Santosh Srivastava. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020.
[95] Arnaud Rosay, Florent Carlier, and Pascal Leroux. Mlp4nids: An efficient mlp-based network intrusion detection for cicids2017 dataset. In *Machine Learning for Networking: Second IFIP TC 6 International Conference, MLN 2019, Paris, France, December 3–5, 2019, Revised Selected Papers 2*, pages 240–254. Springer, 2020.
[96] Luciano Serafini and Artur S d'Avila Garcez. Learning and reasoning with logic tensor networks. In *Conference of the Italian Association for Artificial Intelligence*, pages 334–348. Springer, 2016.
[97] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.
[98] Dan Shiebler. Writing detection rules with llms, Jul 2024.
[99] Xiaokui Shu, Frederico Araujo, Douglas L Schales, Marc Ph Stoecklin, Jiyong Jang, Heqing Huang, and Josyula R Rao. Threat intelligence computing. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1883–1898, 2018.
[100] Leslie F Sikos. Cybersecurity knowledge graphs. *Knowledge and Information Systems*, 65(9):3511–3531, 2023.
[101] Geir Skjøtskift, Martin Eian, and Siri Bromander. Automated att&ck technique chaining. *Digital Threats*, September 2024. Just Accepted.
[102] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.
[103] Splunk. Splunk RBA. https://splunk.github.io/rba/.

[104] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. Uco: A unified cybersecurity ontology. In *Workshops at the thirtieth AAAI conference on artificial intelligence*, 2016.

[105] Tobias Syvertsen. A comparison of machine learning based approaches for alert aggregation. Master thesis, University of Oslo, 2023. Available from: https://www.duo.uio.no/handle/10852/104437.

[106] PeiYu Tseng, ZihDwo Yeh, Xushu Dai, and Peng Liu. Using llms to automate threat intelligence analysis workflows in security operation centers, 2024.

[107] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[108] Florian Wilkens, Felix Ortmann, Steffen Haas, Matthias Vallentin, and Mathias Fischer. Multi-stage attack detection via kill chain state machines. In *Proceedings of the 3rd Workshop on Cyber-Security Arms Race*, pages 13–24, 2021.

[109] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. DeepStochLog: Neural stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10090–10100, 2022.

[110] Maxime Würsch, Andrei Kucharavy, Dimitri Percia David, and Alain Mermoud. Llms perform poorly at concept extraction in cybersecurity research literature. *arXiv preprint arXiv:2312.07110*, 2023.

[111] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. *arXiv preprint arXiv:2307.07700*, 2023.

[112] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[113] Yizhe You, Jun Jiang, Zhengwei Jiang, Peian Yang, Baoxu Liu, Huamin Feng, Xuren Wang, and Ning Li. Tim: threat context-enhanced ttp intelligence mining on unstructured threat data. *Cybersecurity*, 5(1):3, 2022.

[114] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023.

[115] Zysec. ZySec-AI/SecurityLLM · Hugging Face — huggingface.co. https://huggingface.co/ZySec-AI/SecurityLLM.