An Approach to Capture Attack Descriptions via a Controlled Natural Language

Stamatia Papageorgiou, Luca D'Este✉, Carlos E. Budde, Nicola Dragoni, and Andrea Burattin

DTU Compute, Technical University of Denmark, Kgs. Lyngby, Denmark
{s240146,lucde,cesbu,ndra,andbur}@dtu.dk

Abstract. As modern computing systems continue to evolve, cyberattacks have grown in both volume and complexity. Knowledge of these attacks is extensively documented across many structured knowledge bases that aim to support cybersecurity experts and professionals in threat detection and defense. Successful detection and defense, however, require formalization of this knowledge so that machines can effectively process it. In addition, attacks are typically described in natural language, which is inherently ambiguous and makes it hard to (formally) model the available knowledge, ultimately hindering automated machine processing.
One way to solve this issue is to constrain the language used to describe these attacks such that *(a)* ambiguities are addressed and *(b)* a formal model can be deterministically compiled from the description. This work designs and implements a controlled natural language (CNL) and its translation process, enabling ambiguity resolution, capturing essential attack features, and translating descriptions into a formal representation. The approach is validated by encoding several attack techniques using the devised language into formal models and confirming their complete translation with the assistance of domain experts. The feasibility of the approach is demonstrated through a working prototype that translates these descriptions to their corresponding models end-to-end. All code and models are public and available online.

Keywords: Cybersecurity · Formal Methods · Attack Modeling · Threat Modeling · Controlled Natural Language · Cyber Threat Intelligence

1 Introduction

The cybersecurity community makes great effort to record and study adversary behavior; yet this knowledge is often written in a language that is readable by humans but not directly processable by machines. Even though cyber attacks are continuously cataloged in various structured cybersecurity knowledge bases, such as MITRE ATT&CK [29], KEV [5], OWASP [33], CVE [30], their descriptions are in natural language. While this language is human-readable, systems cannot process and apply it automatically in detection processes.

At the same time, cyber attacks are progressively taking more complicated forms, and their detection demands more advanced methods. System monitoring

is perhaps the most used approach for attack detection [27]: adversary behavior is translated into system behavior by shifting the focus from the attacker (potentially unknown) actions to the *observable* events in the asset system. Then, the expected system behavior is represented, e.g., as a process model, which allows for attack detection by the identification of anomalies [6,25].

Constructing models for attack detection, however, is far from trivial and typically requires manual work and expert assistance to interpret the potential attacks. Moreover, the target modeling language needs to be carefully chosen to be expressive enough to describe systems and attacks, but precise enough to avoid ambiguity and undefined behavior. In that sense, formal methods offer several formalisms to model system/adversary interactions in precise, unambiguous, and machine-processable ways. Starting from semantic concepts and operations, a formal model can describe cyber attacks in ways that allow their automatic detection [6,25].

Characteristics that are significant for detection have already been declared and used in previous attack description languages. For example, the detection part of ADeLe [19] describes these elements as the observable events of an attack, their order and repetition, their time-related constraints, the actor-target interactions linked to each event, and their achieved effects. Following the same characteristics, what distinguishes anomalous and benign system behavior can be specified as a set of *attack features* that are significant to the detection of the attack. These attack features include *(a)* the attacker actions observable by the victim system; *(b)* their execution order and causal links; *(c)* the actions that mark the attack's success; *(d)* the involved system assets, distinguished as actors and affected assets; *(e)* behavioral characteristics such as repetition; and *(f)* time-related specifications, such as the period in which an action occurred.

In this paper, we propose to use a Controlled Natural Language (CNL) approach to describe attacks in a structured way. A CNL is a constructed language based on a certain natural language with restricted grammar and vocabulary, that is expressive enough to preserve the natural properties [16]. Our approach can capture the aforementioned attack features and reduce ambiguity in their expression. A description in such a CNL is deterministically parsed and translated into an intermediate representation that models these features and enables further attack modeling. The translation from description to model is demonstrated in a multi-stage translation pipeline.

These are the contributions of this work:

C1: CNL for describing attacks. A CNL is devised to reduce ambiguities that current attack descriptions contain and to fully express the specified attack features for detection and allow for their deterministic parsing.

C2: Structured attack representation. A structured attack representation is designed to capture and store the attack features parsed from descriptions written in the proposed CNL.

C3: Formal attack model. The attack features derived from the structured attack representation are used to build a formal attack model.

C4: Automated translation pipeline. A fully automated pipeline is implemented to translate CNL descriptions into formal attack models and demonstrate the feasibility of the approach. The pipeline is also available to use as a translation tool through a live web application[1].

The rest of the paper is structured as follows: Sect. 2 presents the background and related work. Sect. 3 describes the overview of the approach. Sect. 4 presents the CNL and its grammar designed for describing attacks. Sect. 5 presents the structured attack representation designed to store the attack. Sect. 6 and 7 describe how the approach is implemented and validated. Lastly, Sect. 8 concludes the paper with a summary and future work. All code and produced models are available online[2].

2 Background and Related Work

The detection of attacks in system behavior requires distinguishing malicious from legitimate system activity. D'Este et al. [6,25] demonstrated how process mining and conformance checking techniques provide an effective solution to detecting both known and zero-day attacks. They injected models of anomalous system behavior (manually created based on MITRE ATT&CK) that indicate the execution of an attack into models of benign system behavior, and proved that conformance-based detection on logs derived from these combined process models successfully identifies each of the attacks. The attack modeling process, however, was constructed manually and required expert interpretation of the attack descriptions to effectively capture the anomalous system behavior. The current work aims to assist in this modeling process, through its contributions, by providing a solution that automatically creates attack models that can be injected into benign models for detection purposes. How this work's contributions are related to the work of D'Este et al. is also shown in Fig. 1.

As an example of a source of attack descriptions, we will use MITRE ATT&CK (v.18)[3], which documents adversarial *Tactics* and *Techniques*: the methods and steps adversaries take during attacks to achieve their goals. This knowledge base is widely used by domain experts and practitioners [22,10], as it is well maintained, up-to-date, and supports adversary behavior analysis and detection. Particularly, a recent addition to the ATT&CK framework is Detection Strategies[4]: behavior-driven strategies that show how adversary actions on a system can be effectively monitored and detected. The majority of ATT&CK's content is written in natural language, which is human-readable, but is ambiguous by nature and lacks structure. Several ambiguities have been found in the Detection Strategies descriptions. For example, the description of the *Brute Force: Password Guessing* Technique (T1110.001) [31] mentions *repeated authentication failures* without

[1] See `https://cnl2fam.streamlit.app/`.
[2] See `https://github.com/processintelligence/cnl2fam`.
[3] See `https://attack.mitre.org/versions/v18/`.
[4] See `https://attack.mitre.org/detectionstrategies/`.

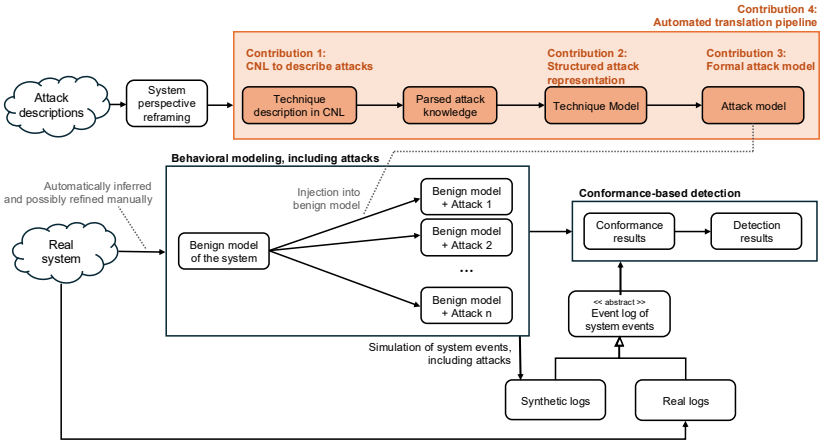


Fig. 1. This work's contributions in relation to D'Este et al.'s MITRE ATT&CK modeling, simulation and detection approach.

specifying the frequency or time interval under which they occur. Similarly, the description of the *Scheduled Task/Job: At* Technique (T1053.002) [32], describes *an anomalous process execution following a scheduled task creation* without clarifying whether the anomalous process is the task itself, a command executed by the task, or a separate execution step altogether. As a result, multiple interpretations exist for the same text, indicating how ambiguous the description is.

Even in more structured representations, such as STIX [15], attack descriptions are stored in natural language and lack procedural attack semantics [7]. This issue is exacerbated by the fact that these descriptions are written at quite a high level of abstraction. As a result, operational attack specifications are left implicit and domain expertise is required for interpretation and attack modeling [14]. This limits the integration of attack descriptions in automated workflows and areas such as real-time threat intelligence, detection, and incident response [26].

Automated extraction of attack knowledge from unstructured Cyber Threat Intelligence (CTI) sources has been demonstrated in several works. Some approaches take advantage of Deep Learning (DL) techniques and, more recently, Large Language Models (LLMs) to extract CTI information and automatically build attack graphs [4,38]. In particular, Cheng et al. [4] use DL methods to reconstruct attack graphs and infer nodes and edges that the source does not state, trading precision with completeness. Because of this, the produced graphs may include artificial information, making them less reliable to use. Zhang et al. [38] use LLMs to fully automate the extraction process from CTI reports. LLMs' behavior, though, is non-deterministic and results are not completely reproducible. In contrast, other works [11,18] rely on Natural Language Processing

and ontology-based methods to build STIX-like outputs more deterministically. Their effectiveness, however, relies on the consistency and explicitness of the inputs, something that is lacking from current attack descriptions, while their output is also not appropriate for modeling attacks as process models.

Attacks can be modeled in various formalisms depending on the information they contain. Attacker's intentions and the consecutive goals that constitute an attack can be expressed using Attack Trees [28]. Jiang et. al [13], for example, used Attack Trees to model energy theft behavior in Advanced Metering Infrastructure. Alternatively, to model the attacker's actions as system behavior that is classified as malicious, Petri Nets [21] or provenance graphs [17] can be used. These have been used in multiple instances to model attacks in different domains and detect anomalous behavior [6,36,35].

One promising solution towards less ambiguous, more explicit and more structured attack descriptions consists of adopting Controlled Natural Languages, subsets of natural languages with restricted vocabulary and grammar [16]. They are designed to reduce ambiguity present in natural language and provide readable and processable text for both humans and machines. While several CNLs have been created for software requirement specification [8,9,34], only a few are specific to Cybersecurity. Chase et al. [3] propose Noam, which is designed to express cybersecurity knowledge and reason over it. It aims to model systems and uncover how adversaries structure their attacks. Yampolskiy et al. [37] introduce the Cyber-Physical Attack Description Language (CP-ADL), built for describing cyber-physical attacks based on the cause-effect chain linking actions. However, Noam remains undeveloped as a usable tool, and CP-ADL is purely descriptive and does not encode entities with structured properties or behavior, limiting its attack semantics to action cause-effect links.

3 Research Questions and General Approach

Existing literature shows that while it is possible to extract and formalize attack knowledge to effectively detect attacks, it is hard to produce reliable attack models automatically. Modeling attacks from the system's perspective as process models, in particular, has only been done manually so far. This is mainly due to the language used to describe the attacks. Attack descriptions often contain ambiguities with important detection-related semantics left implicit. This paper proposes a CNL designed for describing attacks in an explicit and structured format that reduces ambiguities and enables their deterministic parsing and translation into formal process models. It explores the following research questions:

RQ1. How can the attack features be described in a way that reduces ambiguity in their expression and supports deterministic parsing?
RQ2. How can these attack descriptions be translated into formal attack models that preserve the attack features as described?
RQ3. How can the overall feasibility of the approach be demonstrated through a working prototype?

Table 1. Comparison of related approaches against derived requirements.

	R1	R2	R3	R4	R5
CRUcialG [4]	✓	✗	partial**	✗	✓
AttacKG+ [38]	✓	✗	partial**	✗	✓
TTPDrill [11]	✓	partial*	✗	✗	✓
STIXnet [18]	✓	partial*	✗	✓	✓
Noam [3]	✓	✗	✗	✗	✗
CP-ADL [37]	✓	✗	partial**	✓	✗
This work	✓	✓	✓	✓	✓

* intermediate steps rely on Machine Learning/DL models
** contains partially procedural semantics and not execution semantics

To tackle these RQs, the following requirements are identified: **R1.** Ability to capture an unstructured attack description using a structured representation; **R2.** Ability to deterministically translate the textual description into other representations; **R3.** Generated output should capture both procedural and execution semantics of the attack; **R4.** Generated output should be faithful to the source without extra clutter; **R5.** Availability of a working implementation of translation from input description to target representation. Table 1 compares the current work and the related approaches against the identified requirements.

To address the identified gap, a domain-specific CNL and its translation into a formal model are designed and implemented as a multi-stage pipeline. Descriptions of attacks from the system's perspective, written in this language, are deterministically parsed and formatted into an intermediate representation that contains only the attack-related information, discarding filler words that are present in the language for human readability. The intermediate representation is then translated into a Technique Model: a data model of typed entities and fields that maps one-to-one onto the described elements, preserving the attack features in a formal representation. Finally, the Technique Model is translated into a process model. This work targets Petri Nets, though the Technique Model is independent of any particular process modeling formalism.

4 A CNL for Describing Attacks

The proposed CNL is designed to describe an attack as it is observed by the affected system: a series of system asset interactions. Specific structures of the language are heavily inspired by the Gherkin language [34], widely used in Behavior- and Test-Driven Development to define software requirements as explicit and structured scenarios. Its use in this context is described below.

A technique description consists of the *header* with the identity of a technique, the *background* with a declaration of the assets involved in its execution, a series of *events* that depict the attack behavior, and a declaration of attack

completion. Thus, the grammar of the language is structured into the corresponding four main blocks: Header, Background, Event(s), and Completion. To demonstrate how the language is used in practice, the technique "Brute Force: Password Guessing" written in this CNL serves as a running example.

Please note that the actual grammar is expressed using ANTLR4 [23], but, for readability purposes, in the rest we will use a more compact representation, where literals are presented in red and are enclosed in double quotes and rules are in blue. Typical regular expression symbols (e.g., ?, *, |) will be used with the expected meaning. The complete ANTLR4 grammar is available online.[5]

Header. The Header describes the general information of a technique and its associated tactics, their identifiers, and their names, and are used to link to MITRE ATT&CK IDs. The Header block holds the following structure:

```
"Tactic:" TacticID TacticName
"Technique:" TechniqueID TechniqueName
```

Here is the instantiation for the "Brute Force: Password Guessing":

```
1 Tactic: TA0006 Credential_Access
2 Technique: T1110.001 Brute_Force_Password_Guessing
```

Background. This describes which assets are used during the execution of the technique, along with the properties they carry that are observable and play a role in their detection. Assets must be declared with their AssetType and AssetName. Any of their PropertyName and PropertyValue can be optionally defined on the same line. While asset types classify the assets, asset names play an important role in distinguishing them in the description. Assets are declared once in the Background and are referenced throughout the description by their name. If multiple assets are part of a technique, they are declared in the same block connected with the keyword "And", while their properties are separated by commas.

The language supports the complete declaration of a number of assets with properties. These assets are adopted from the MITRE CAR Data Model [20] and MITRE ATT&CK descriptions. The language allows other assets, with their own properties, to be declared as well. Table 2 shows the asset types that are supported by the language, and their origin. The general structure of the Background block, inspired by the Gherkin language's Background structure, is:

```
"Background"
"Given" AssetType "as" AssetName ("with" PropertyName "=" PropertyValue)?
("And" AssetType "as" AssetName ("with" PropertyName "=" PropertyValue)?)*
```

[5] See `https://github.com/processintelligence/cnl2fam/blob/main/src/cnl/grammar/CNL.g4`.

Table 2. Supported asset types and their properties.

Asset type	Property Names	Source
Process	`signed`, `command_line`, `parent_process`	MITRE CAR
File	`path`, `signed`	MITRE CAR
Registry	`path`	MITRE CAR
NetworkConnection	`destination`, `source`, `transport_protocol`	MITRE CAR
Endpoint	`port`, `protocol`, `ip_address`	MITRE CAR
Session	`access_level`	MITRE CAR
Driver	`signed`	MITRE CAR
Module	`path`, `signed`	MITRE CAR
Device	`device_type`	MITRE ATT&CK
Volume	`path`	MITRE ATT&CK
Account	`scope`	MITRE ATT&CK
Message	`data`	MITRE ATT&CK
Handle	`hexadecimal_number`, `target`	MITRE ATT&CK
Directory	`path`	MITRE ATT&CK

Here is the instantiation for the "Brute Force: Password Guessing":

```
Background
Given process as AuthenticationService
And account as TargetAccount
And message as CorrectCredentials
And message as IncorrectCredentials
And session as User
```

Event. Events are defined as a series of blocks, each corresponding to a singular observable action within a technique. The Event block is structured following the Given/When/Then pattern, inspired by Gherkin's scenario structure. The "Given" and "Then" clauses contain the pre-conditions and post-conditions of the action described as asset states. The "When" clause describes the action of the event. The action statement is written in a Subject-Verb-Object pattern, which translates to ActorAsset, ActionVerb, and ObjectAsset. The pre- and post-conditions are expressed as asset-state conditions written in either a passive (e.g. is modified) or an active voice (e.g. exists) (i.e., an AssetName and a StateVerb). Additional information regarding an asset's state condition or action can be added at the end of the clause sentences as Modifiers. Repetition and Timing sentences explicitly describe the event frequency and time frame accordingly. The Event block has this structure:

```
"Event" EventID
"Given" AssetName ("is" StateVerb | StateVerb) Modifier* // Pre-conditions
"When" ActorAsset ActionVerb ObjectAsset Modifier*       // Action
"Then" AssetName ("is" StateVerb | StateVerb) Modifier* // Post-conditions
```

```
("Repeated" X "times" ("within" T TimeUnit)?)?          // Repetition
(("During" | "Outside") TimePeriod)?                    // Timing
```

Modifier defines location, destination, source and trigger of actions or asset states in relation to other assets. They are distinguished by the keywords "in", "to", "from" and "by", followed by the related asset.

In case of multiple pre- and post-conditions, they can be defined in their corresponding clause and be combined according to their logical relation to the action's execution. Pre-conditions can be connected through AND/OR/XOR, while post-conditions can only be connected through AND, since they are treated as deterministic. The corresponding keywords are "And", "Or", and "Xor".

Here are instantiations for the "Brute Force: Password Guessing":

```
Event 1
Given AuthenticationService is awaiting_input
When AuthenticationService receives IncorrectCredentials to TargetAccount
Then IncorrectCredentials is rejected
Repeated 5 times within 1 sec

Event 2
Given IncorrectCredentials is rejected
When AuthenticationService receives CorrectCredentials to TargetAccount
Then CorrectCredentials is accepted And User is active
```

Completion The Completion block describes which events can be considered as final events of the technique that depict its successful end. If more than one event is the final one, then they are written on the same line and separated by AND/OR/XOR. The keywords "And", "Or", and "Xor" show by which events the technique ends. The general structure of the Completion block is:

```
"Completion" "Event" EventID
```

Here is the instantiation for the "Brute Force: Password Guessing":

```
Completion Event 2
```

Although the language's structure is constrained, it is still flexible. Certain grammar positions accept an open vocabulary. These positions are ActionVerb and StateVerb; technique, tactic, and AssetName; PropertyValue; and TimePeriod. Asset types outside of the explicitly-defined set of asset types and their property types are also part of the open vocabulary. This ensures the expressiveness of the language.

5 Technique Model

An attack description expressed following the provided CNL can be parsed to construct a Technique Model, containing all attack-related information. Such a

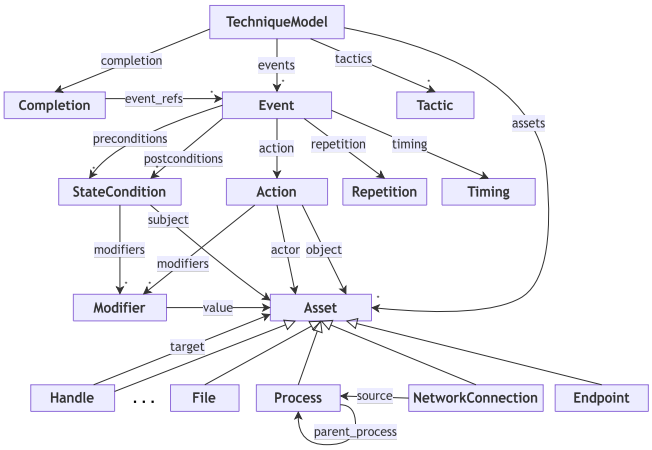


Fig. 2. Simplified UML class diagram of the Technique Model.

Technique Model serves as a typed and structured representation of the attack, carrying the described attack features that are significant towards its detection. As a data model, it constitutes a formal attack representation, and enables further automated processing. A simplified UML class diagram of a Technique Model is shown in Fig. 2, and the complete class diagram can be found online.[2]

The Technique Model consists of a series of events over assets that depict the system behavior under the attack. At its root, it is a single instance identified by its MITRE ATT&CK identifier, name, and tactics it belongs to. This instance also contains the set of assets the technique relies on for its execution, the set of events that constitute it, and the set of events that signal its completion.

Each event, in turn, is composed of an action, a set of pre- and post-conditions combined via logical operators, a repetition frequency, and a timing constraint. Pre- and post-conditions are stored as state conditions. State conditions and actions reference the modifiers, and altogether, they reference the assets involved. Assets share a common base Asset entity and each supported asset type is a derived asset instance with their explicit observable properties.

Following the language's grammar structure, an *action* contains the actor, action verb, object, and modifiers. Similarly, *pre-* and *post-conditions* have a subject asset, state verb, and modifiers. *Modifiers* contain a type and a specific asset. The assets are derived from the base asset, which contains a type, name, and unique identifier. Each supported asset type extends the base with observable properties in its respective type. Assets are created and stored once, at the base of the Technique Model, and each reference links to the same asset instance by following the name as declared in the Background of the description.

Events capture the individual steps of a technique in terms of asset interactions. The specified state conditions in an *event* show the particular states of the participating assets before and after the interaction has occurred. Therefore, the *state conditions*, when examined from a holistic perspective, capture the order

and the causal dependencies among events. *State conditions* also show which assets and asset states enable certain interactions. These are observable elements that are crucial to recognize the attack.

Events of the technique's completion are captured and translated as one *completion* instance. It consists of a set of events and the logical operator that combines them in relation to the end of the technique. These are the final observed asset interactions that reveal the end of the specific technique; or otherwise, the last step(s) the attacker takes to achieve the technique's end goal.

Constrained elements of the language's grammar are converted into the Technique Model's fields after thorough type enforcement. Values of restricted elements that are modeled as enumerations get translated as such as well. This ensures the Technique Model's content is in the appropriate type for future processing. Elements that are part of the open vocabulary are translated as they are described, as raw strings, without further transformations.

Declared assets that are not part of the supported assets listed in Table 2 are parsed, rather than rejected, and are translated as base assets in the Technique Model, carrying only their base attributes. Their properties, however, if explicitly defined, are ignored at the translation layer. This is deliberate: a property is a typed field defined for a specific asset class so it can only be validated relative to a known type and for an unsupported type, there's no place for it to be stored. This allows the language to be future-proof by representing unforeseen assets in events, with the trade-off that they are not completely modeled later on.

6 Translating Attack Descriptions

The implemented translation pipeline consists of 4 transformation stages, each processed by a distinct component: the Parser, which parses the CNL description text and creates the Parse Tree; the Visitor, which traverses the Parse Tree to extract the relevant information; the Translator, which converts the extracted information into the typed Technique Model; and the Attack Model Builder, that uses the Technique Model to build an Attack Model of the described attack. The full architecture of the pipeline is shown in Fig. 3.

The CNL grammar is implemented using ANTLR4. Based on this grammar, ANTLR4 generates a Lexer, Parser, and Visitor for the target language, Python. The Lexer contains the set of grammar rules that the Parser uses to recognize different patterns in the input text and build the Parse Tree. The Parse Tree carries the parsed information, which is later transformed into the Technique Model. ANTLR4 guarantees deterministic parsing of the input description based on the defined grammar rules. Additionally, any syntax errors are caught during parsing, ensuring that only a correctly written description ultimately gets translated. It is also important to mention that ANTLR4's auto-generated base Visitor consists of default methods that the pipeline's Visitor overrides. The overridden methods ensure that only domain-specific information is extracted. This allows the Translator to later apply domain semantics to the extracted words and phrases.

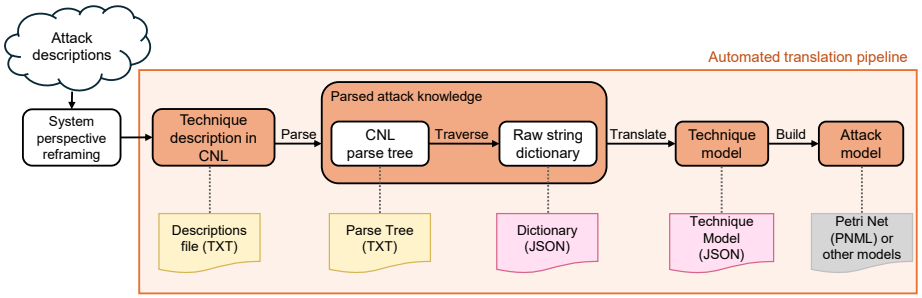


Fig. 3. Translation pipeline architecture

The Technique Model is implemented using Python data classes, where each of its entities is a data class containing its own typed attributes. It is built compositionally and structurally follows the class diagram shown in Fig. 2. The design allows for automated traversal and processing, as well as further translation of the technique into a formal process model, while its structure and content remain independent of any specific downstream modeling formalism.

The Translator inspects the data by the Visitor to populate the Technique Model, enforcing appropriate value types, storing unsupported declared assets as base-class assets and setting undefined optional fields to null. Once the Technique Model object is created, it is possible to use it to generate a behavioral model of the attack, such as a Petri Net. Petri Nets are formalisms widely used for modeling system behavior. They are both machine-processable and human-readable since they can be graphically represented. In this pipeline extension, Petri Nets are created in their simplest form, which consists of transitions, places, and arcs.

As a demonstration, a Petri Net Builder is implemented using the PM4Py library [1]. This Builder automatically generates a Petri Net using actions as transitions, and state conditions as places. Depending on the position of the state conditions, either as pre-conditions or post-conditions, they may form intermediate places between the transitions. As a result, the causal links between events and the different execution paths the technique can take are revealed.

The logical operators connecting the pre- and post-conditions are the factors that produce concurrent and alternative execution paths. AND operators make connected transitions execute concurrently, while XOR operators create alternative paths, with only one taken per execution. OR operators allow any combination of conditions to hold. For the current prototypical demonstration, we interpret OR operators as XOR in the Petri Net, but this can be relaxed in needed.

7 Evaluation

This section presents the results of our evaluation, where some attacks have been modelled and evaluated with the help of experts.

7.1 Selected Techniques for Modeling

We decided to validate our approach against four MITRE ATT&CK Techniques. These techniques were selected with the help of a cybersecurity expert, who considered practical significance, popularity, and potential impact.

1. *OS Credential Dumping: LSASS Memory* (T1003.001). It contains a variety of asset types, and has a simple and comprehensible execution.
2. *Brute Force: Password Guessing* (T1110.001). Its description contains ambiguity regarding the repetitive login attempts. It is a good example to demonstrate how the language resolves this ambiguity.
3. *Content Injection* (T1659). It contains a unique range of assets and asset compositionality. It is also of interest to the cybersecurity community as it is frequently mentioned in the National Vulnerability Database [2].
4. *Scheduled Task/Job: At* (T1053.002). It is assigned to three ATT&CK Tactics and can appear across multiple attack stages. Its description is also ambiguous as to which assets are involved, making it a good demonstration for the disambiguation the CNL provides.

In the following, we present an evaluation on all these attacks, with respect to certain validation criteria and we will also dive into the specific aspects of one attack. We do not claim that our descriptions are complete or fully accurate; we focus more on the language's ability to express the attacks.

7.2 Evaluation on All Attacks

The contributions of this work were evaluated by empirically validating the translation pipeline's artifacts. For encoding the selected techniques, their CNL descriptions were manually written after an expert assisted in interpreting how they are observed from the system's perspective. These descriptions were then used as input to the translation pipeline, which converted them into Technique Models and subsequently into their corresponding Petri Nets. To validate that the content is preserved during the translation, reference Technique Models of the same techniques were created with the help of an expert (independently of the pipeline), and were compared against the ones produced by the CNL. For the validation of the results, we run semi-structured interviews with two domain experts from academia, each with over five years of experience in the field of cybersecurity and formal methods. During the interviews, the experts viewed the manually written CNL descriptions of the selected techniques, and their produced Technique Models and Petri Nets, and we discussed the outcomes.

We considered the requirements mentioned in Sect. 3 and we derived the following validation criteria: **V1.** the technique description (in CNL) expresses the content of the ATT&CK text; **V2.** the technique description (in CNL) does not contain ambiguities; **V3.** the Technique Model captures the described attack features; **V4.** the Petri Net presents the alternative paths the attack can follow based on the described actions, pre- and post-conditions and assets involved. Please note that V1 and V2 assess the extent to which the proposed CNL answers

Table 3. Comparison of results against validation criteria.

	V1 (expressivity)	**V2** (ambiguity)	**V3** (Tech. Model)	**V4** (Petri Net)
T1003.001	✓	✓	✓	✓
T1110.001	✓*	✓	✓	✓
T1659	✓	✓	✓	✓
T1053.002	✓**	✓	✓	✓

* our description additionally contains an explicit repetition count of the login event.
** our description additionally connects the anomalous process to the created task.

RQ1 and satisfies R1, R2, and R4; while V3 and V4 assess the extent to which the translation pipeline answers RQ2 and satisfies R2, R3, and R4. RQ3 is already answered by the working implementation of the CNL and its translation pipeline, which satisfies R5 as well.

Table 3 summarizes our assessment of each translated technique against the validation criteria based on the experts' feedback from the interviews. In addition, a qualitative assessment of each technique is available online.[2]

The comparison between reference and pipeline-produced Technique Models showed no divergence. During the interview phase, the experts concluded that the CNL descriptions of all four techniques satisfy every validation criterion, and noted that two of them explicitly state what the source descriptions leave ambiguous. Namely, an expert observed that "*this solution maintains the natural language and resolves ambiguities, like the repetition, making MITRE more operational*". Both experts emphasized the value the encodings bring to identifying vulnerabilities and technique executions, mentioning "*I can directly see which are the vulnerable assets the attack targets*" and "*I immediately see how the technique plays out*". They also expressed interest in experimenting further with the language and the tool.

7.3 Example of Technique Implementation

In this section we want to showcase how one technique can be implemented following our CNL. We focus on *Content Injection* (T1659), whose translation took 514 ms and allocated 606 KB at peak[6]. Its CNL description is:

```
Tactics: TA0001 Initial_Access, TA0011 Command_and_Control
Technique: T1659 Content_Injection

Background
Given process as browser
And endpoint as server
And network_connection as Browser_conn with source="browser", destination="server"
And process as local_process with parent_process = "browser"
And file as F
```

[6] Run on Intel Core i7-12650H with 16 GB RAM and on Windows 11.

```
And directory as Temp with path = "/temp/"
And directory as AppData with path = "/%AppData%/"

Event 1
Given Browser_conn is activated
When browser receives F from server
Then F is received

Event 2
Given F is received
When browser saves F in AppData
Then F exists in AppData

Event 3
Given F is received
When browser saves F in Temp
Then F exists in Temp

Event 4
Given F exists in AppData Xor F exists in Temp
When local_process executes F
Then F is active

Completion Event 4
```

The produced Technique Model is too large to present in full here; the listing below shows a single asset instance from it, but the complete model is available online[2]:

```
"Browser_conn": {
    "__class__": "NetworkConnection",
    "asset_type": "network_connection",
    "name": "Browser_conn",
    "asset_id": "8f102a9c-23f1-4230-a03b-15eb737009ee",
    "destination": {
        "__class__": "Endpoint",
        "asset_type": "endpoint",
        "name": "server",
        "asset_id": "bfd77dbb-1eea-4141-8f68-849738add3ed",
        "port": null, "protocol": null, "ip_address": null
    },
    "source": {
        "__class__": "Process",
        "asset_type": "process",
        "name": "browser",
        "asset_id": "eb8b7554-42ad-494a-b67a-6491b23c21c3",
        "signed": null, "command_line": null, "parent_process": null
    },
    "transport_protocol": null
}
```

Finally, Fig. 4 reports the generated Petri Net from the CNL description. Please note that places and transitions are labeled in a way that makes them easily relatable to the source CNL attack description.

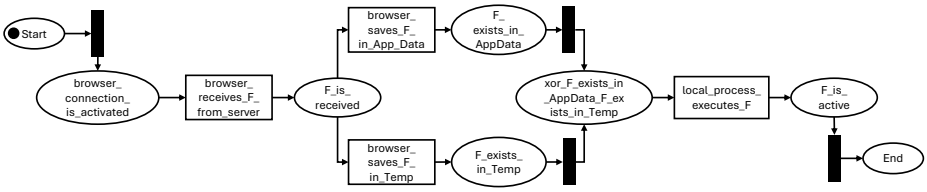


Fig. 4. Petri Net for the Content Injection attack.

8 Conclusion

This work shows how to convert textual descriptions of attacks into formal models that depict the system's behavior under attack, through a controlled language that reduces the ambiguities of natural language. Descriptions in this language are deterministically parsed and translated into attack models (e.g., Petri Nets) through a fully implemented and automated translation pipeline.

Returning to the research questions and contributions, the three RQs mentioned in Sect. 3 are positively answered by the corresponding three contributions. RQ1 is answered through C1: A CNL is designed in a way that addresses the ambiguities found in attack descriptions from MITRE ATT&CK and supports their deterministic parsing. RQ2 is answered through C2: A structured representation, the Technique Model, is built to address the missing procedural semantics for system behavior modeling in existing literature, and fully capture the attack features needed for detection. RQ3 is answered through C3: The full translation pipeline as implemented can translate CNL descriptions to Technique models end-to-end, and it can be used as a command-line tool[2] or as a web application.[1]

While this work provided a solid usable pipeline, several directions remain open for future work. First, the approach requires further validation by extending to a larger set of techniques and experts. Second, since the CNL descriptions are currently written manually, future work could explore leveraging LLM-based tools that assist in the authoring phase. Such automated tools could also extend the validation by generating a large corpus of descriptions. Third, the Technique Model's versatility could be further demonstrated by exploring other formalisms beyond Petri Nets, such as Coloured Petri Nets [12] or DECLARE [24]. Lastly, state conditions and actions could be extended and mapped to concrete system artifacts, such as code or security logs, providing stronger detection indications.

Overall, this work serves as a connection between attack detection knowledge and its operational usability in detection and defense systems, as it can automatically produce attack process models and be potentially integrated in automated processes targeting their detection.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Berti, A., van Zelst, S.J., Schuster, D.: Pm4py: A process mining library for python. Softw. Impacts **17**, 100556 (2023). https://doi.org/10.1016/J.SIMPA.2023.100556
2. Booth, H.: National vulnerability database. https://nvd.nist.gov/ (2015). https://doi.org/10.18434/M3436
3. Chase, L., Mohasseb, A., Aziz, B.: Epistemology for cyber security: A controlled natural language approach. In: Saad, I., Rosenthal-Sabroux, C., Gargouri, F., Chakhar, S., Williams, N.L., Haig, E. (eds.) Advances in Information Systems, Artificial Intelligence and Knowledge Management - 6th International Conference on Information and Knowledge Systems, ICIKS 2023, Portsmouth, UK, June 22-23, 2023, Proceedings. Lecture Notes in Business Information Processing, vol. 486, pp. 267–282. Springer (2023). https://doi.org/10.1007/978-3-031-51664-1_19
4. Cheng, W., Zhu, T., Chen, T., Yuan, Q., Ying, J., Li, H., Xiong, C., Li, M., Lv, M., Chen, Y.: Crucialg: Reconstruct integrated attack scenario graphs by cyber threat intelligence reports. IEEE Trans. Dependable Secur. Comput. **22**(6), 6345–6360 (2025). https://doi.org/10.1109/TDSC.2025.3584826
5. Cybersecurity and Infrastructure Security Agency: Known exploited vulnerabilities catalog, https://www.cisa.gov/known-exploited-vulnerabilities-catalog, accessed: 2026-08-03
6. D'Este, L., Løvstrand Mortensen, E., Pontoppidan Rasmussen, E., Budde, C.E., Dragoni, N., Burattin, A.: From behavior to breach: Modeling, simulating and detecting attacks with process mining. In: 30th International Conference on Enterprise Design, Operations, and Computing (EDOC 2026) (2026), To Appear
7. Ferraz, Á.L.R., Barbieri, S., de Souza, M.E., Júnior, L.A.P.: The procedural semantics gap in structured CTI: A measurement-driven STIX analysis for APT emulation. CoRR **abs/2512.12078** (2025). https://doi.org/10.48550/ARXIV.2512.12078
8. Fuchs, N.E., Schwitter, R.: Attempto controlled english (ACE). CoRR **cmp-lg/9603003** (1996), http://arxiv.org/abs/cmp-lg/9603003
9. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Generation of formal requirements from structured natural language. In: Madhavji, N.H., Pasquale, L., Ferrari, A., Gnesi, S. (eds.) Requirements Engineering: Foundation for Software Quality - 26th International Working Conference, REFSQ 2020, Pisa, Italy, March 24-27, 2020, Proceedings [REFSQ 2020 was postponed]. Lecture Notes in Computer Science, vol. 12045, pp. 19–35. Springer (2020). https://doi.org/10.1007/978-3-030-44429-7_2
10. Golushko, A., Zhukov, V.: Adversary profiling and activity emulation in the process of development and evaluation of information security threat countermeasures. In: International Workshop on Advanced in Information Security Management and Applications. CEUR Workshop Proceedings (2022), https://ceur-ws.org/Vol-3094/paper_8.pdf
11. Husari, G., Al-Shaer, E., Ahmed, M., Chu, B., Niu, X.: Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of CTI sources. In: Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017. pp. 103–115. ACM (2017). https://doi.org/10.1145/3134600.3134646
12. Jensen, K.: Coloured petri nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Petri Nets: Central Models and Their Properties. pp. 248–299. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)

13. Jiang, R., Lu, R., Wang, Y., Luo, J., Shen, C., Shen, X.: Energy-theft detection issues for advanced metering infrastructure in smart grid. Tsinghua Science and Technology **19**(2), 105–120 (2014). https://doi.org/10.1109/TST.2014.6787363
14. Jiang, Y., Meng, Q., Shang, F., Oo, N., Minh, L.T.H., Lim, H.W., Sikdar, B.: MITRE att&ck applications in cybersecurity and the way forward. CoRR **abs/2502.10825** (2025). https://doi.org/10.48550/ARXIV.2502.10825
15. Jordan, B., Piazza, R., Darley, T.: STIX™ Version 2.1. Oasis standard, OASIS Committee Specification (January 2021), https://docs.oasis-open.org/cti/stix/v2.1/stix-v2.1.html, committee Specification 02
16. Kuhn, T.: A survey and classification of controlled natural languages. Comput. Linguistics **40**(1), 121–170 (2014). https://doi.org/10.1162/COLI_A_00168
17. Lv, Y., Qin, S., Zhu, Z., Yu, Z., Li, S., Han, W.: A review of provenance graph based APT attack detection: Applications and developments. In: 7th IEEE International Conference on Data Science in Cyberspace, DSC 2022, Guilin, China, July 11-13, 2022. pp. 498–505. IEEE (2022). https://doi.org/10.1109/DSC55868.2022.00075
18. Marchiori, F., Conti, M., Verde, N.V.: 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, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023. pp. 3:1–3:11. ACM (2023). https://doi.org/10.1145/3600160.3600182
19. Michel, C., Mé, L.: Adele: An attack description language for knowledge-based intrusion detection. In: Dupuy, M., Paradinas, P. (eds.) Trusted Information. pp. 353–368. Springer US, Boston, MA (2001). https://doi.org/10.1007/0-306-46998-7_25
20. MITRE: Data model. Cyber Analytics Repository (CAR) (2015), https://car.mitre.org/data_model/
21. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989). https://doi.org/10.1109/5.24143
22. Oosthoek, K., Doerr, C.: Sok: Att&ck techniques and trends in windows malware. In: Chen, S., Choo, K.R., Fu, X., Lou, W., Mohaisen, A. (eds.) Security and Privacy in Communication Networks - 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23-25, 2019, Proceedings, Part I. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 304, pp. 406–425. Springer (2019). https://doi.org/10.1007/978-3-030-37228-6_20
23. Parr, T.: The Definitive ANTLR 4 Reference. Pragmatic Bookshelf, 2nd edn. (2013)
24. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA. pp. 287–300. IEEE Computer Society (2007). https://doi.org/10.1109/EDOC.2007.14
25. Rasmussen, E.P., Mortensen, E.L.: Modeling and Simulating Attacks with Process Mining for Cybersecurity Research. M.s. thesis, Dept. of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kongens Lyngby, Denmark (Feb 2026), https://findit.dtu.dk/en/catalog/69c335e9e49dd032fcd99903
26. Roy, S., Panaousis, E., Noakes, C., Laszka, A., Panda, S., Loukas, G.: Sok: The MITRE att&ck framework in research and practice (2023). https://doi.org/10.48550/ARXIV.2304.07411

27. Satilmiş, H., Akleylek, S., Tok, Z.Y.: A systematic literature review on host-based intrusion detection systems. IEEE Access **12**, 27237–27266 (2024). https://doi.org/10.1109/ACCESS.2024.3367004
28. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobb's Journal **24**(12), 21–29 (December 1999)
29. Strom, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B.: MITRE ATT&CK: Design and philosophy. Tech. rep., The MITRE Corporation (2018)
30. The CVE Program : CVE (Common Vulnerabilities and Exposures), https://www.cve.org/, accessed: 2026-08-03
31. The MITRE Corporation: Brute force: Password guessing. MITRE ATT&CK®, Sub-technique T1110.001 (2025), https://attack.mitre.org/techniques/T1110/001/, version 1.7
32. The MITRE Corporation: Scheduled task/job: At. MITRE ATT&CK®, Sub-technique T1053.002 (2025), https://attack.mitre.org/techniques/T1053/002/, version 2.4
33. The OWASP Foundation: OWASP Top Ten Web Application Security Risks, https://owasp.org/www-project-top-ten/, accessed: 2026-08-03
34. Wynne, M., Hellesøy, A., Tooke, S.: The Cucumber Book: Behaviour-Driven Development for Testers and Developers. Pragmatic Bookshelf, 2nd edn. (2017)
35. Xiao, Y., Jiang, L., Nie, S., Li, D., Wu, S., Xu, K., Li, Q.: How far should we need to go : Evaluate provenance-based intrusion detection systems in industrial scenarios (2026). https://doi.org/10.48550/ARXIV.2603.22982
36. Xu, D., Nygard, K.E.: Threat-driven modeling and verification of secure software using aspect-oriented petri nets. IEEE Trans. Software Eng. **32**(4), 265–278 (2006). https://doi.org/10.1109/TSE.2006.40
37. Yampolskiy, M., Horváth, P., Koutsoukos, X.D., Xue, Y., Sztipanovits, J.: A language for describing attacks on cyber-physical systems. Int. J. Crit. Infrastructure Prot. **8**, 40–52 (2015). https://doi.org/10.1016/J.IJCIP.2014.09.003
38. Zhang, Y., Du, T., Ma, Y., Wang, X., Xie, Y., Yang, G., Lu, Y., Chang, E.: Attackg+: Boosting attack graph construction with large language models. Comput. Secur. **150**, 104220 (2025). https://doi.org/10.1016/J.COSE.2024.104220