ISA Transactions 141 (2023) 10–19

Contents lists available at ScienceDirect

ISA Transactions

journal homepage: www.elsevier.com/locate/isatrans

Research article

Detecting contradictions from IoT protocol specification documents based on neural generated knowledge graph☆

Xinguo Feng [a,*], Yanjun Zhang [b], Mark Huasong Meng [c,d], Yansong Li [a], Chegne Eu Joe [a], Zhe Wang [e], Guangdong Bai [a]

[a] The University of Queensland, Australia
[b] Cyber Security Research and Innovation (CSRI), Deakin University, Australia
[c] Institute for Infocomm Research, A*STAR, Singapore
[d] National University of Singapore, Singapore
[e] Griffith University, Australia

ARTICLE INFO

Article history:
Received 1 December 2022
Received in revised form 30 March 2023
Accepted 21 April 2023
Available online 29 April 2023

Keywords:
Internet of things
Natural language processing
Web protocol
Contradiction detection
Large language models

ABSTRACT

Due to the boom of Internet of Things (IoT) in recent years, various IoT devices are connected to the Internet and communicate with each other through network protocols such as the Constrained Application Protocol (CoAP). These protocols are typically defined and described in specification documents, such as Request for Comments (RFC), which are written in natural or semi-formal languages. Since developers largely follow the specification documents when implementing web protocols, they have become the *de facto* protocol specifications. Therefore, it must be ensured that the descriptions in them are consistent to avoid technological issues, incompatibility, security risks, or even legal concerns. In this work, we propose Neural RFC Knowledge Graph (NRFCKG), a neural network-generated knowledge graph based contradictions detection tool for IoT protocol specification documents. Our approach can automatically parse the specification documents and construct knowledge graphs from them through entity extraction, relation extraction, and rule extraction with large language models. It then conducts an intra-entity and inter-entity contradiction detection over the generated knowledge graph. We implement NRFCKG and apply it to the most extensively used messaging protocols in IoT, including the main RFC (RFC7252) of CoAP, the specification document of MQTT, and the specification document of AMQP. Our evaluation shows that NRFCKG generalizes well to other specification documents and it manages to detect contradictions from these IoT protocol specification documents.

1. Introduction

The Internet of Things (IoT) is an emerging technology in recent years. It refers to devices and sensors that are uniquely addressable based on their communication protocols, and are adaptable and autonomous with inherent security [2]. Its development is closely connected to many cutting-edge technologies such as blockchain [3,4], smart home [5], and machine learning [6–8]. During the past decade, IoT applications have experienced rapid growth and have been successfully applied on both individual (e.g., e-health, and smart home) and professional levels (e.g., smart supply chain, smart city, and industry IoT) [9,10]. It is estimated that there will be over 500 billion IoT devices connected to the Internet by 2030 [11].

IoT devices communicate through the corresponding *web protocols*, which are defined by the specification documents. Efforts have been made to process specification or specification-like documents using rule-based approaches. Tian et al. [12] build finite state machine for protocol implementation testing by processing RFC documents to extract the states. Andow et al. [13] build ontology to detect the contradictions in application privacy policy documents on the personal information being collected. Xie et al. [14] detect compliance issues on Alexa skills privacy policy documents by extracting keywords mentioning data types being collected by developers. However, the issue of contradictions within the web protocol specification documents is largely unexplored; these contradictions can lead to erroneous implementations and confusion for developers.

Identifying these contradictions from protocol specification documents faces several unique challenges. First, unlike structured specification documents, such as API documentations, protocol specification documents do not have specific fields that

☆ This work extends the preliminary results presented in [1]. It includes a more generic approach for constructing a knowledge graph through utilizing large language models in model pre-training (Section 4.1), entity extraction (Section 4.2), and rule extraction (Section 4.3). It also includes more extensive experiments and evaluation on various specification documents (Section 5).

* Corresponding author.
E-mail address: s.feng@uq.edu.au (X. Feng).

https://doi.org/10.1016/j.isatra.2023.04.025
0019-0578/

contain certain information that can be easily collected with the existing rule-based approaches. Typical information such as the data objects mentioned in the document and the technical requirements are written in unstructured or semi-structured natural languages, which vary in different forms of phrases to describe the functionalities of the protocol, making it hard for a rule-based approach to identify them. Second, the semantics encoding of these requirements demands the preservation of the correct information and sentiment for contradiction detection logic. Requirements that have similar vocabularies but have different or even opposite semantics might be wrongly transformed into similar encodings. Such an error will propagate and affect the contradiction detection process. Third, requirements that are for the same data object can locate in different positions of the document. Inter connection needs to be built to relate them together.

In this work, we propose NRFCKG, an approach for automatically detecting contradictions within a web protocol specification document by constructing a knowledge graph for the document and by formalizing the detection process into a constraint satisfaction problem. We construct the knowledge graph through entity extraction, relation extraction, and rule extraction to structurally represent the document by utilizing state-of-the-art large language models. We formalize the problem to a constraint satisfaction problem and solve it with an SMT solver. We apply NRFCKG to the specification documents of the most extensively used web protocols in IoT, which are RFC7252 of CoAP, MQTT specification document and AMQP specification. NRFCKG detects four contradictions from RFC7252 and one contradiction from MQTT specification document.

We summarize our contributions as follows:

- **An automated approach to detecting contradictions from specification documents**. To our best knowledge, we are the first to propose an approach to detecting contradictions from IoT protocol specification documents. We propose NRFCKG which features automated knowledge graph construction for contradiction detection.
- **Representing specification document as a knowledge graph**. We identify the essential components for constructing a knowledge graph from web protocol specification documents, such as entities, relations, and rules. We utilize various large language models to extract these components and construct the knowledge graph. We evaluate the performance of NRFCKG on these tasks and show that it can generalize well to other specification documents.
- **Formalizing contradiction detection as a constraint satisfaction problem**. We define the types of contradictions and formalize the contradiction detection process as a constraint satisfaction problem, which can be solved with an SMT solver.
- **Revealing contradictions in specification documents**. We demonstrate that NRFCKG can successfully detect contradictions from various real-world specification documents, which shows the soundness and the generalization of our approach. It detects four contradictions from RFC7252 and one contradiction from MQTT specification document. Our findings reveal that contradictions do exist in specification documents, even though they are usually considered to have been thoroughly reviewed before release.

2. Background and related work

In this section, we present the background knowledge to facilitate the understanding of our work, and summarize the related work that inspires this work.

2.1. Background

Protocol specification documents. Web protocol specification documents describe the technical details of a web protocol. They are usually written by engineers or computer scientists to describe the methods, behaviors, or innovations of web protocols in natural languages. Developers who wish to implement the protocol or users who wish to utilize the implementations should always refer to the specification document that defines the protocol.

Knowledge graph. A knowledge graph is a multi-relational graph constructed with *nodes* (entities) and *edges* (relations), where each edge indicates the two entities are connected with a specific relation [15].

Large language models. Since the introduction of the Transformer [16] deep neural network architecture, there has been several significant work that adopts this architecture to train variate large language models with a large number of parameters, such as the BERT family (BERT [17], RoBERTa [18], DeBERTa [19], etc.) as language encoders and the GPT family (GPT [20], GPT-2 [21], GPT-3 [22], etc.) as language decoders. These large language models achieve state-of-the-art performance on several Natural Language Processing (NLP) tasks. Although large language models like BERT and GPT are trained on large-scale datasets, they perform well on downstream tasks with general text data. For encoders like BERT, they are usually further pre-trained with domain specific text data before applying them to domain specific downstream tasks in order to achieve better performance. Rasmy et al. [23] further pre-train BERT with electronic health record data for predicting heart failure and pancreatic cancer. Feng et al. [24] further pre-train BERT with programming code data for natural language code search and code documentation generation. Chalkidis et al. [25] further pre-train BERT with legal documents data to assist with legal NLP research, computational law, and legal technology applications.

2.1.1. Related work

Knowledge graph representation for documents. Li et al. [26] construct a knowledge graph from Android API documents, which can be easily accessed by developers. Mondal et al. [27] propose a way to conduct an end-to-end knowledge graph construction on NLP related papers to describe NLP tasks and their evaluations. Typical tasks for constructing a knowledge graph are entity extraction and relation extraction. Although there exist some tools [28–30] for these tasks, they are usually for general purposes. It is unlikely that they would work well with tasks that are domain-specific without further injecting the domain knowledge.

Rule extraction from specification documents. Rules in specification documents define the functionalities and behaviors of the protocol. The natural language writing style of rules is specified in RFC2119 [31]. In particular, it defines the modal keywords to indicate the requirement levels for rules [12]. Furthermore, RFC8174 [32] emphasizes the usage of uppercase letters for modal keywords defined in RFC2119. Tian et al. [12] extract the rules with keyword matching and use dependency parsing to process the rules. Dependency parsing is also studied in other work such as [33]. It works well with simple sentences but suffers with complicated sentences with multiple objects or multiple subordinate clauses.

Different representations for specifications. Andow et al. [13] construct an ontology on applications' privacy policy documents and check for logical contradictions (e.g. "*Not collecting personal information*" contradicts with "*but collecting email address*"), which is the main inspiration for our work. However, its main focus is on privacy policy documents. The data types they extract

are different from the data types in specification documents. Wang et al. [34] utilize traffic, documents, and configurations of several IoT protocols and construct the finite state machines to evaluate their security. Pacheco et al. [35] also construct finite state machines of protocols from documents to perform attack synthesis. These finite state machine representations only focus on run time dynamics and ignore static properties. However, in a specification documents, there are many static properties of entities that describe technical details. Our work aims to handle specification document related data types (entities, relations) and extract static properties that describe the protocol technical details.

Contradiction detection for specifications. Harabagiu et al. [36] propose an approach to recognize negation, contrast and contradictions for general text. Xie et al. [14] study the privacy policy compliance issue of virtual personal assistant apps. Wang et al. [37] propose a formal analysis framework to detect specification contradictions in single sign-on protocols. Mahadewa et al. [38] explore contradictions in privacy policy on trigger-action integration platforms. Another recent study by Meng et al. [39] proposes a systematic analysis methodology to scrutinize the compliance of mobile operating systems in protecting users' privacy. However, recognizing contradictions in specification documents, such as Requests for comments, has not been well studied.

Vulnerability detection for implementation. Existing vulnerability detection methods focus on the implementation level. In particular, machine learning and deep learning techniques play an important role. Feng et al. [40] present methods for detecting vulnerabilities of IoT devices directly from the firmware. Zhang et al. [41] and Sun et al. [42] present deep learning methods for detecting cyber attacks and incidents. Lin et al. [43] also present deep learning methods for detecting software vulnerabilities. Qiu et al. [44] present machine learning methods for detecting Android malware.

Detection with various data formats. There are also existing detection methods with various data formats for different purposes, but serve as study examples. Mangla et al. [45] propose a misbehavior detection framework for cooperative intelligent transport system. Zhu et al. [46] propose a transformer-based method for machinery diagnosis.

3. Problem definition

Web protocol specification documents are written in unstructured natural languages. NRFCKG parses them and generates knowledge graphs that can be automatically checked for contradictions. In this section, we first define the components of the knowledge graph (Section 3.1), and then we present the types of contradictions we target to detect in this work (Section 3.2).

3.1. Entity, rule and relation

The knowledge graph generated by NRFCKG consists of three components, i.e., *entity*, *rule* and *relation*, where the *entities* and the *rules* are represented as nodes, and the *relations* are represented as edges.

Entity. We refer to an entity as an object in web protocols that has functionalities or behaviors being described. Commonly used entities include "*message*", "*options*", "*token*", etc. We use a single field data structure to represent an entity node in the knowledge graph: Entity(*name*). For example, the entity "*confirmable message*" is represented as Entity(*confirmable message*).

Rule. A rule node consists of a set of atomic rules appearing in the same rule statement, concatenated by logical connective "$\wedge$ (AND)", "$\vee$ (OR)", "$\veebar$ (XOR)" and "$\neg$ (NEGATION)". We define an atomic rule as a four-tuple data structure:

$$(\{variable, operator, value\}, necessity),$$

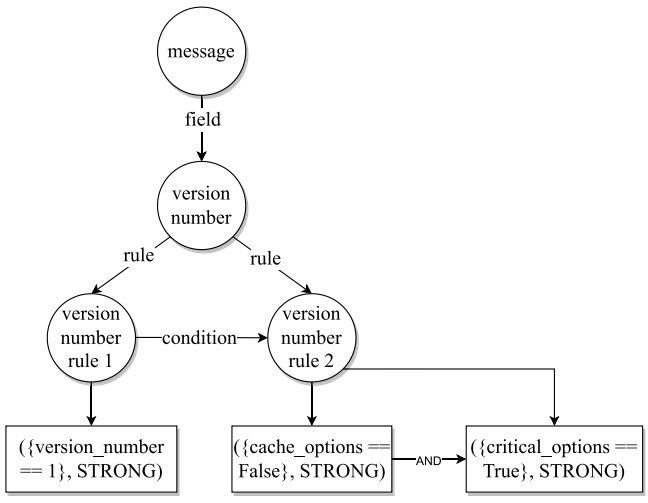

Fig. 1. NRFCKG's knowledge graph representation of an example statement in RFC: "If the version number of a message is set to 1, the options of the message MUST NOT be cached and the options MUST be critical".

in which $\{variable, operator, value\}$ represents the rule content, and the *necessity* represents the requirement level, including "STRONG" and "WEAK", where "STRONG" indicates an absolute requirement level such as "MUST", "REQUIRED", "SHALL", "MUST NOT", and "WEAK" indicates an optional requirement such as "NOT RECOMMENDED", "MAY" and "OPTIONAL". For example, in the statement: *"Message version number MUST be set to 1 and the options of the message MUST be cached"*, the extracted rule is:

$$(\{version\_number == 1\}, STRONG)$$
$$\wedge (\{cached\_options == TRUE\}, STRONG)$$

Relation. The relations NRFCKG targets to extract from RFCs include (1) the relation between an entity and an entity, e.g., "A version number [entity] is a field of [relation] a message [entity]", (2) the relation between an entity and a rule, e.g., ($\{version\_number == 1\}$, *STRONG*) [rule] is a rule of [relation] Confirmable message [entity], and (3) the conditional relation between two rules. For example, in the statement *"If the version number of a message is not set to 1, the options of the message MUST NOT be cached"*, ($\{version\_number! = 1\}$, *STRONG*) [rule] is a condition of [relation] ($\{cached\_options == FALSE\}$, *STRONG*) [rule]. The former is the antecedent rule and the latter is the consequent rule.

Fig. 1 illustrates the KG representation of the rule statement "*If the version number of a message is set to 1, the options of the message MUST NOT be cached and the options MUST be critical*".

3.2. Contradictions

The core idea of NRFCKG is to represent an unstructured specification document under analysis as a structured knowledge graph, to formalize this as a constraint satisfaction problem, and then to solve it with an SMT solver. If the constraint is unsatisfiable, NRFCKG reports a contradiction. In particular, we define two types of contradictions as follows:

- **Direct contradiction.** This occurs when different rules of the same entity e - denoted as $\{r_1, \ldots, r_n\}_e$ contradict each other. That is, the conjunction of rules is evaluated as unsatisfiable, i.e.,

$$\bigwedge_{i=1}^{n} \{r_i\}_e == unsat$$

A direct contradiction is regarded as an erroneous contradiction of a specification document which may lead to implementation issues. This contradiction captures the following three scenarios.

(1) Contradiction among plain rules. A plain rule refers to a rule that is not an antecedent rule or consequent rule. For example, consider the following rule statements *"The version number of a message MUST be set to 1"* and *"Message version number MUST be 0"*. The rules for these rule statements are ({*version_number* == 1}, STRONG) and ({*version_number* == 0}, STRONG). These are plain rules as they are not antecedent rules or consequent rules. We concatenate them and see that they evaluate as unsatisfiable.

$$(version\_number == 1) \wedge (version\_number == 0) == unsat$$

(2) Contradictions between plain rule and consequent rule. It occurs when a plain rule states "*A must be True*", while a consequent of a conditional rule states "*A is False*". For example, consider the following rule statements *"The version number of a message MUST be set to 1"* and *"If the options of the message are cached, the version number of the message MUST be set to 0"*. The rule extraction and evaluation are the same as in the example above, but the contradiction is between a plain rule and a consequent rule.

(3) Contradictions between the consequent of conditional rules. For example, two conditional rules state the same antecedent while implying a contradicted consequent. For example, consider the following rule statements *"If the options of the message are not cached, the version number of the message MUST be set to 1"* and *"If the options of the message are not cached, the version number of the message MUST be set to 0"*. The rule extraction and evaluation are the same as in the example above, but the contradiction is between two consequent rules that serve the same antecedent rule.

- **Conditional contradiction.** This occurs when the antecedent of conditional rules contradicts plain rules. Denoting c_i as an antecedent rule of a conditional proposition, NRFCKG reports a conditional contradiction if

$$\bigwedge_{i=1}^{n} \{r_i, c_i\}_e == unsat.$$

For example, consider the following rule statements *"Message version number SHOULD be set to 1"* and *"If the version number of a message is not 1, the options MUST be cached"*. The rule for the first statement is ({*version_number* == 1}, STRONG), which is a plain rule. The antecedent rule for the second statement is ({*version_number* != 1}, STRONG). We concatenate them and the combined constraint is evaluated as unsatisfiable.

$$(version\_number == 1) \wedge (version\_number\ != 1) == unsat$$

The conditional contradictions extracted by NRFCKG can highlight the instruction of error handling in specification document, which is likely ignored by the developers especially when such statements appear in different places in the document.

4. Approach

We design NRFCKG as a four-phase approach that consists of *language model pre-training*, *entity recognition and relation extraction*, *rule extraction*, and *contradiction detection*. The overall approach is shown in Fig. 2.

4.1. Language model pre-training

We first pre-train BERT with the specification documents of the three most extensively used IoT protocols, which are CoAP [47–50], MQTT [51] and AMQP [52]. We split the documents into individual sentences using the NLP tool `Natural Language Toolkit (NLTK)` [53], and pre-train the case-sensitive BERT base model with its original pre-training tasks, which are MLM and NSP. Through this, the pre-trained BERT model can produce IoT domain specific contextual embeddings. We refer to this pre-trained model as IoT-BERT.

4.2. Entity recognition and relation extraction

In this phase, we aim to train a model to construct a skeleton knowledge graph to represent the protocol described in the specification document. There are two main components, which are *entity recognition* and *relation extraction*. Note that this phase only uses RFC7252 as training data.

We refer to an entity in a web protocol as an object that has some functionalities or some properties in the web protocol, such as "*message*", "*token*", "*version number*", and so on. To prepare the data for training the model, we first pre-process RFC7252 by removing irrelevant parts of the document, such as the abstract, copyright notice, table of content, figures, tables, and references. We then use `NLTK` [53] to split the document into individual sentences. We use the BERT Tokenizer from `Hugging Face` [54] to tokenize each sentence. We use the BIO Tagging approach [55] to conduct data labeling on a token level. Similar to any Name Entity Recognition task, for each token in a sentence, we label it as either the beginning of an entity (class 0), the inside of an entity (class 1), the outside of an entity (class 2) or the padding token (class 3). For example, consider the tokenized sentence "*These messages are called Confirmable messages PAD PAD PAD*". To simplify the demonstration, this example assumes that each word is one token, although this is usually not the case with a WordPiece tokenizer [17], such as the one BERT uses. The labels we give to this sentence are [2, 0, 2, 2, 0, 1, 3, 3, 3], as the first "*messages*" token is the beginning of an entity and it is a complete entity by itself. The "*Confirmable*" token is the beginning of an entity, and the second "*message*" token is the inside of an entity. The rest are either the outside of any entity or padding tokens. We then take the BERT for token classification model from `Hugging Face` [54], which is a BERT model with an extra token classification layer, and substitute the layers of the BERT model with the layers of the pre-trained IoT BERT model. We then use the annotated data to train an entity extractor for extracting entities from web protocol specification documents. After we extract the entities, we examine each possible pair of extracted entities, and identify the relation for them from the predefined list: *no relation, equivalent, field, type*, and *feature*. For example, *Confirmable message* is a *type* of *message*.

4.3. Rule extraction

An entity has properties that serve as the functionalities of the web protocols. In this phase, we extract these properties from the rule statements and represent them as rules. There are three main steps, which are *rule statement extraction*, *condition split* and *property identification*.

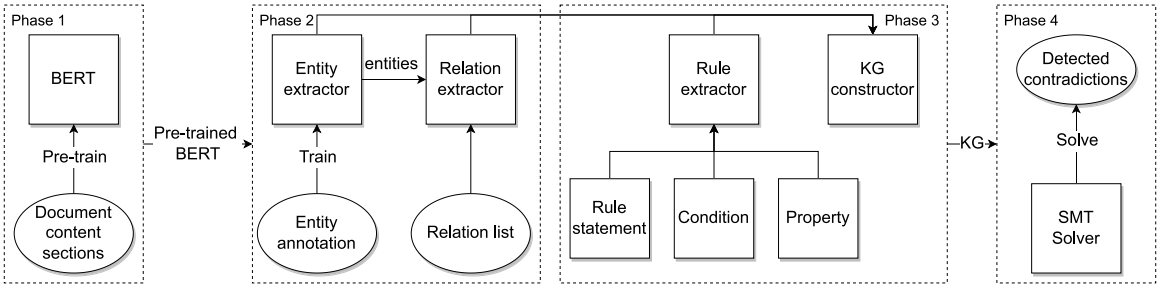

Fig. 2. Overview of approach showing four phases.

Table 1
Modal keywords for extracting rule statements.

STRONG keywords	WEAK keywords
MUST, REQUIRED, SHALL, MUST NOT, SHALL NOT	SHOULD, RECOMMENDED, SHOULD NOT, NOT RECOMMENDED, MAY, OPTIONAL

4.3.1. Rule statement extraction

RFC2119 [31] states that in many standards track documents, several keywords are used to signify the requirements in the specification. We refer to the statements that contain these keywords as rule statements, as they are the requirements that need to be followed. Since the release of RFC2119 (which specifies the standard for keywords usage in RFCs to indicate requirement levels, but also generalizes to all web protocol specification documents) in 1997 [31], specification documents (released after 1997) enforce the use of capitalized modal keywords (such as "MUST", "MAY" and etc.) to indicate the requirement level of a rule in the specification. We therefore examine the capitalized modal keywords used in RFC7252 and identify the sentences that contain those capitalized modal keywords as rule statements. These rule statements are the base sentences for extracting and constructing the rules. More specifically, we extract the strong statements and weak statements based on the modal keywords as shown in Table 1, following the definition from a related study [12]. Algorithm 1 in Appendix demonstrates this process.

4.3.2. Condition split

In a rule statement, there can exist a part of it that serves as the antecedent and another part that serves as the consequent. Consider the following rule statement "*If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned*", where "*If the request to the destination times out*" is the antecedent, and "*then a 5.04 (Gateway Timeout) response MUST be returned*" is the consequent. We note that there also can be a conditional relation in a rule statement that is not specified with explicit keywords such as "*if*" and "*when*". For example, consider the following rule statement "*Messages with unknown version numbers MUST be silently ignored*". The implicit antecedent in this rule sentence is "*Messages with unknown version numbers*" and the consequent is "*MUST be silently ignored*". As such, NRFCKG needs to identify both the explicit conditional relation and the implicit conditional relation in the rule statements.

To this end, we take use of the GPT-2-xl model [21] from `Hugging Face` [54], which is the original GPT-2 model for text generation. Given a rule statement, we fine-tune this model to generate the underlying antecedent and consequent for this rule statement, if there exists any. For each of the rule statements from RFC7252, we construct the training text data as a string with three mandatory fields, which are "*Statement*", "*Antecedent*" and "*Consequent*". For example, for the rule statement "*Messages with unknown version numbers MUST be silently ignored*", the constructed string would be:

> *Statement: Messages with unknown version numbers MUST be silently ignored*
> *Antecedent: Messages with unknown version numbers*
> *Consequent: MUST be silently ignored*

Note that this is one complete string and there is an explicit line separator after each field. If there is no conditional relation in a rule statement, we simply put "*Not applicable*" after the "*Antecedent*" field and the "*Consequent*" field.

We then use this constructed text data to fine tune the model with the Next Token Prediction task as an antecedent and consequent extractor that can identify the underlying conditional relation within a rule statement. Given an unseen rule statement, we construct a text string with the "*Statement*" field filled with the rule statement, but leave the part after the "*Antecedent*" field blank as a prompt for the model to generate the rest. Notice that we only construct the prompt up the "*Antecedent*" field, as the prompt is fed into the model as a string. After the model generates the antecedent, it will generate the consequent of the statement starting with a "*Consequent*" keyword.

4.3.3. Property identification

In this step, we extract the properties of an entity described in a rule statement. For example, for the antecedent rule statement "*Messages with unknown version numbers*", the property is "*unknown version numbers*". For the consequent rule statement "*MUST be silently ignored*", the property is "*be silently ignored*". These properties are usually described as phrases in the rule statement, but their forms vary and it is difficult to define syntactical patterns to extract them. Also, we extract these properties and point them to the corresponding entities. For the above example, the property "*unknown version numbers*" should be pointed to the entity "*version number*", and the property "*be silently ignored*" should be pointed to the entity "*message*".

We use the GPT-2-xl model [21] from `Hugging Face` [54] and fine-tune this model to generate the underlying properties in a rule statement. The training task is Next Token Prediction we fine-tune this model to recognize our defined properties pattern. For a rule statement, we first backtrack five sentences in the original document and append them to the front of the statement to construct a context. If the rule statement is an antecedent or consequent after being processed from the last step, we also append the original rule statement to the context. This is useful as we also want to assign the properties to entities. It is common in natural languages to use pronouns or phrases starting with

keywords such as "this", "that", and so on, to refer to something that has been mentioned earlier. In some cases, given a single rule statement, it is not sufficient to decide which entities the properties should be pointed to.

For each of these rule statement contexts, we then construct the text string with three mandatory fields, which are "*Context*", "*Statement*", and "*Properties*". For example, for the antecedent rule statement "*Messages with unknown version numbers*", the constructed string would be:

> *Context: (Five sentences before rule statement, plus the original rule statement before condition split)*
> *Statement: Messages with unknown version numbers*
> *Properties: version number @ unknown version number = True;*

For the consequent rule statement "*MUST be silently ignored*", the constructed string would be:

> *Context: (Five sentences before rule statement, plus the original rule statement before condition split)*
> *Statement: MUST be silently ignored*
> *Properties: message @ be silently ignored = True;*

We then use this constructed text data to fine-tune the model with Next Token Prediction task as a property extractor that can extract the underlying properties and point them to corresponding entities. Given an unseen context and statement, we construct a text string with the "*Context*" field filled with the context and the "*Statement*" field filled with the rule statement, but leave the part after the "*Properties*" field blank as a prompt for the model to generate the rest. If it generates any property that does not match that specific format, we filter the generated property out.

Note that the entity extractor and the property extractor are different. The entity we generate with the property extractor is usually the subject that the property is describing and is a candidate entity. We add an extra step here to match it to the entities that the entity extractor extracts as they are specifically for the task and should be more accurate. For an entity–property pair generated by the property extractor, we take the candidate entity part and pass it through the pre-trained Phrase-BERT model [56], which is a BERT model with dedicated pre-training tasks to generate phrase embedding, to get an embedding vector for the candidate entity. We also pass all the entities extracted by the entity extractor through Phrase-BERT to get their embedding vectors. We then calculate the cosine similarities between them and select the most similar one as the entity we point the property to. If there is no such entity that has a similarity larger than 0.9, we leave it as it is.

For an entity, there could be many properties in variant forms of phrases. We need to further normalize them as some of them can be actually describing the same behaviors and therefore should be normalized as one property. We pass the extracted properties into Phrase-BERT, calculate the cosine similarities, and group them as one if it is larger than the 0.9 threshold. There exist situations where two properties phrases are similar in the embedding vector space, but they actually have different meanings. Consider the property phrases "*unknown version number = True*" and "*set the version number to 1 = True*", where the first property states that the version number is unknown and the second property states that the version number is one, which has totally different meaning. However, they have very high similarity in the vector space and can be grouped as one property easily. To address this, we examine the properties we annotate for rule statements in RFC7252 and define a list of patterns to further process the extracted properties. We define two common properties for an entity, which are "*value*" and "*error*". The "*value*" property describes the entity value and the "*error*" property describes the error handling case for an entity. For the example above, after further processed by the patterns, they will be transformed as "*value = unknown*" and "*value = 1*", both being pointed to the entity "*version number*". For the rest of the properties phrases that do not fall into these patterns, we leave them as dynamic properties.

With all the steps combined, we have a property extractor that can generate the underlying candidate entities and the properties, match the candidate entities to the more accurate entities, and normalize the properties. After this, we can now construct the rule nodes in the knowledge graph. Recall that we define an atomic rule as a four-tuple data structure:

$$(\{variable, operator, value\}, necessity)$$

For every property, we take the property phrase as the *variable*, the operator as the *operator*, and the value as *value*. The *necessity* comes from the requirement level of the original rule statement. If the rule statement is a STRONG statement, then its necessity is STRONG. Otherwise, it is WEAK. For each rule statement, we connect all the atomic rules with logical connective "$\wedge$ (AND)" or "$\vee$ (OR)". We determine this by searching if there is an "or" keyword in the original rule statement. If there is, then the logical connective is set as "$\vee$ (OR)". Otherwise, it is set as "$\wedge$ (AND)". After this, we have all the rule nodes constructed. We then connect them to the entities they are pointed to with an edge with a "rule" relation. For the rule nodes that have a conditional relation, we connect them with an edge with a "condition" relation. The knowledge graph is complete at this stage and it is ready to be used for contradiction detection.

4.4. Contradiction detection

We now describe the contradictions detection step with the knowledge graph constructed above. We do contradictions detection on the entity level. Recall that each atomic rule within a Rule node is constructed with the *variable*, the *operator* and the *value*. We first iterate through all the atomic rules under the same entity and use the Z3 solver [57] to create a literal for each unique variable. We then also transformed each unique value to a unique real number (if the unique value is not a real number originally, we choose a seed then generate the transformed value from the seed), as the solver can only accept literals with the same type. We then encode these variable and value pairs as constraints according to their operators. For these constraints within the same Rule node, we connect them with their corresponding logical connective "AND" $\wedge$ or "OR" $\vee$.

Recall that there are three types of direct contradictions. For the first type of direct contradiction, for each entity node, we extract all the plain rules. We enumerate all the plain rule pairs and connect the constraints within them with the logical connective "AND" $\wedge$, and see if the result is evaluated as satisfactory or not. For the second type and third type of direct contradictions, we simply substitute the rule pairs to plain–consequent rule pairs and consequent rule pairs under the same antecedent rule.

For conditional contradictions, the process is also similar. For each entity node, we extract all the plain rules and the antecedent rules. We enumerate all the plain–antecedent rule pairs and connect the constraints within them with the logical connective "AND" $\wedge$, and see if the results are evaluated as satisfactory or not.

Algorithm 2 in Appendix demonstrates the process for detecting direct contradiction type 1 as an example.

Table 2
Statistics of sentences and rule statements used for experiments from three specification documents, where 20% of the rule statements from MQTT (63) and AMQP (67) are used.

	CoAP	MQTT	AMQP
Sentence	1280	1668	1323
Rule Statement	217	63/317	67/334

5. Evaluation

We evaluate the performance of our approach in two phases. The first phase mainly focuses on entity recognition, condition split, and property identification tasks for CoAP, MQTT and AMQP specification documents. We randomly sample 20% of the rule statements from the latter two specification documents and perform a benchmark evaluation on these tasks. The second phase focuses on detecting potential contradictions in all three specification documents.

The three specification documents we choose are for the most commonly and extensively used IoT protocols. For most of the other existing IoT protocols specification documents, they are either very outdated or not commonly used. They are not representative specification documents and we exclude them in the experiment.

5.1. Experiment setup

We split RFC7252 into 1280 sentences, and extract 217 rule statements from them. For the entity recognition task, we annotate all 1280 sentences. For the condition split task, we annotate the 217 rule statements to indicate if there is a condition relation in the statements. For the property identification task, we first further split the antecedent and consequent statements, resulting in 355 statements, and annotate them to indicate the entities and the properties in the statements.

We have randomly sampled 90% of rule statements from CoAP as training data. We then take the rest of the rule statements from CoAP and 20% of the rule statements from MQTT and AMQP as benchmarks to evaluate the three tasks mentioned above. Table 2 summarizes the statistics for the sentences and rule statements used in the experiment. For the contradiction detection task, we manually examine the original sentences associated with the contradicted rules reported by NRFCKG and determine if it is a true positive. All experiments are performed on a ThinkStation P620 with two NVIDIA RTX A6000 GPUs running Ubuntu OS.

5.2. Benchmark results

Table 3 summarizes the results of entity recognition, and Table 4 summarizes the results of condition split and property identification. For entity recognition, we report the accuracy and the macro F1 score. For condition split and property identification, as they are generative tasks, we report the BLEU score, which is a metric for evaluating the quality of the generated text with a value between 0 and 1. The formula to calculate BLEU score is as follows:

$$BLEU(R; C) = e^{max(0, \frac{r}{c} - 1)} \cdot exp(\sum_{n=1}^{\infty} w_n \ln p_n(R; C)),$$

where R denotes the reference text, C denotes the candidate (generated) text, r denotes the length of the reference text, c denotes the length of the candidate text, w denotes the weight for an n-gram of tokens, and p denotes the precision for an n-gram of tokens.

Table 3
Entity Recognition performance of NRFCKG.

	Accuracy	F1 Score
CoAP	0.99	0.94
MQTT	0.98	0.85
AMQP	0.99	0.78

Table 4
BLEU Score of Condition Split and Property Identification.

Task	Protocol	BLEU
Condition Split	CoAP	0.69
	MQTT	0.71
	AMQP	0.73
Property Identification	CoAP	0.71
	MQTT	0.75
	AMQP	0.69

Overall, the results show that the models we train on RFC7252 generalize well to other specification documents, with either high accuracy and F1 score, or high BLEU score, depending on the tasks.

Our benchmark results demonstrate the effectiveness of NRFCKG. First, the pre-training of BERT allows the model to adapt to the contextual semantics of this domain specific lingo environment prior to any actual training. Second, the entities in different protocols are similar as they follow some general patterns. For example, in CoAP, "*Confirmable message*" is an entity, while in MQTT, "*Subscriber message*" is an entity. The overlapping part "*message*" is already a big indicator that this phrase could be an entity. Also, the model predicts if these tokens are an entity considering the semantics of different tokens in the same sentence. Given what is being described in the sentence, such as a functionality of a type of message, it provides more information for the model to make a prediction.

5.3. Contradiction detection results

Table 5 summarizes the number of true positive and false positive contradictions detected from RFC7252, MQTT and AMQP. Table 6 shows the original contradicted sentences for the true positive cases from the specification documents.

We detected four true positive contradictions out of eight detected contradictions for CoAP, one true positive contradiction out of 11 detected contradictions for MQTT, and zero true positive contradiction out of two detected contradictions for AMQP. For the direct contradiction detected in RFC7252, the original rule statements are "*Implementations SHOULD also support longer length identifiers and MAY support shorter lengths*" and "*Note that the shorter lengths provide less security against attacks, and their use is NOT RECOMMENDED*". In the first statement, it states that the shorter lengths identifier MAY be supported. But in the second statement, it states that it is NOT RECOMMENDED. According to RFC2119 [31], these two keywords are at the same requirement level. It is confusing for the developers to decide if they should implement this functionality or not. If one implementation supports it and another does not, it might cause communication issues for the devices and the system might become vulnerable. For the direct contradiction detected in MQTT specification document, the original rule statements are "*The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection*" and "*If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT packet except AUTH packets*". In the first statement, it states that the server MUST process the second CONNECT packet as an error and close the connection, while the second statement

Table 5
Number of detected contradictions from three specification documents.

Protocol	Contradiction	True Positive	False Positive	Document Total
CoAP	Direct Contradiction	1	3	4
	Conditional Contradiction	3	1	4
MQTT	Direct Contradiction	1	4	5
	Conditional Contradiction	0	6	6
AMQP	Direct Contradiction	0	0	0
	Conditional Contradiction	0	2	2
Total	–	5	16	21

Table 6
Original rule statements for detected contradictions. "RE": requirement level, "D": direct contradiction, "C": conditional contradiction, "W": weak contradiction, "S": strong contradiction.

Protocol	Rule 1	Rule 2	Type	RE
CoAP	Implementations SHOULD also support longer length identifiers and MAY support shorter lengths	Note that the shorter lengths provide less security against attacks, and their use is NOT RECOMMENDED	D	W
	The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field	If there are any bytes, they MUST be processed as a message format error	C	S
	The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field	Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error	C	S
	Implementations of this specification MUST set this field to 1 (01 binary)	Messages with unknown version numbers MUST be silently ignored	C	S
MQTT	The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection	If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT packet except AUTH packets	D	S

states that the server MUST NOT process any data except the AUTH packets. Although the contradicted part in the second rule statement has a condition and it is the consequent part, the first statement is a plain rule and it is supposed to be satisfactory at all times.

For the conditional contradictions detected, all of them are discussing the error handling situation. For example, the original rule statements for a conditional contradiction detected in RFC7252 are "*The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field*" and "*If there are any bytes, they MUST be processed as a message format error*". The first statement is a plain rule and it states that there MUST NOT be any data after the Message ID field. The second statement states that if there is any data after the Message ID field, it MUST be processed as an error. These conditional contradictions worth being highlighted as an alert for the developers to pay attention to in order to avoid any potential implementation issues.

For the false positive cases, we investigate them and find all of them are caused by the incorrectly grouped variables, i.e., two variables that do not mean the same are grouped as one. Recall that we group variables by getting their vector representations from Phrase-BERT and comparing their similarities. The similarity threshold is a hyper-parameter which we can tune. Increasing the similarity threshold can lower the false positive rate because only very similar properties will be considered as one variable. But this would also make the tool miss the real same properties in different expressions without a very high similarity score. We choose 0.9 as the similarity threshold after several runs as it achieves a good trade off of revealing true positive cases and limiting the false positive cases. However, the model is limited to producing different vectors for variables that have different meanings but have similar semantics. We therefore recommend that further confirmation should be conducted on the detected positive cases when NRFCKG is used in practice. A possible direction to address this problem is to further pre-train this model with more protocol documents so that it can adapt to this domain specific contextual environment and further produce better vector representation for the variables to reduce the error rate.

We notice that there is a drop in the amount of true positive contradictions being detected from MQTT and even none from AMQP. We investigate this issue and find that MQTT and AMQP

Algorithm 1 Extracting rule statements from specification document

Input: Preprocessed spec document $\mathcal{S}$
Output: strong and weak rules statements sets

```
modal_keywords ← ["MUST", ..., "MAY", "OPTIONAL"]
strong_modal_keywords ← ["MUST", ..., "SHALL"]
strong_rules_statements ← [ ]
weak_rules_statements ← [ ]
rfc_sentences ← NLTK.split_sentences(S)
for i ← 0, length(rfc_sentences) do
    sentence ← rfc_sentences[i]
    for j ← 0, length(modal_keywords) do
        keyword ← modal_keywords[j]
        if keyword in sentence then
            if keyword in strong_modal_keywords then
                strong_rules_statements.append(sentence)
            else
                weak_rules_statements.append(sentence)
                break
return strong_rules_statements, weak_rules_statements
```

Algorithm 2 Check Direct Contradiction Type 1

Input: entity_rules_set, solver
Output: direct_contradictions_set
for $i \leftarrow 0$, $length(entity\_rules\_set)$ **do**
  $entity\_rule\_a \leftarrow entity\_rules\_set[i]$
  $constraint\_expression\_a \leftarrow solver.create\_constraint(entity\_rule\_a)$
  for $j \leftarrow 0$, $length(entity\_rules\_set)$ **do**
    $entity\_rule\_b \leftarrow entity\_rules\_set[j]$
    $constraint\_expression\_b \leftarrow solver.create\_constraint(entity\_rule\_b)$
    $constraints = constraint\_expression\_a \wedge constraint\_expression\_b$
    if $solver.solve(constraints) == unsat$ **then**
      $direct\_contradictions\_set.append((entity\_rule\_a, entity\_rule\_b))$
    $solver.reset()$
return $direct\_contradictions\_set$

specification documents are maintained and updated more frequently. The specification document for MQTT we use is version 5.0 and there are three previous versions. The specification document for AMQP we use is version 1.0 and there are four previous versions. It is likely that the original contradictions have already been noticed and fixed. Yet, our approach still manages to find one true positive contradiction from MQTT.

In terms of true negative and false negative cases, we cannot report them as we do not know if there is any contradiction in the specifications in advance. Our tool reports the cases that cannot be solved by the solver, then we evaluate the reported cases manually to see if they are real contradictions. Reporting true negative and false negative cases will need ground truth contradictions in advance, which will require extensive human labor. Our work serves as a warning tool that report suspicious cases.

6. Discussion

We now discuss the limitations of this work and some possible future directions.

Relation extraction. In this work, we manually examine each extracted entity pair and assign a relation from the defined list of relations. We try training a relation extractor by annotating more than 10,000 sentences from enumerating all the entity pairs from RFC7252. Given an entity pair and a sentence that contains them, the classifier needs to predict what is the underlying relation between. However, more than 80% of the data have a "*No relation*" relation, resulting in poor generalization of the classifier to other specification documents. Furthermore, relation extraction itself is a hard downstream task for NLP, as the classifier needs more information than just a sentence that contains the entity pair to predict the correct relation especially in technical documents and it requires intensive labor for annotation. This task requires more study and empirical experiments to justify what is the right approach to address the problem. Some possible future directions might be constructing a context environment by backtracking more sentences before the sentence that contains the entity pair to provide more information to the classifier to make the right prediction, or through some few-shot learning approach that does not require intensive data annotation.

Variable normalization. Most of the false positive contractions being detected are caused by the incorrectly normalized variables. We used Phrase-BERT [56] as an embedding extractor to compute the vector representation of the variables. However, this model is not further pre-trained. Further pre-training this model can potentially improve the embedding representations of the variables, such that the false positive rate can be decreased. Hu et al. [58] presents state of the art methods that utilizes texture features to detect events on social media platforms, which potentially can be used for extracting more complicated variables.

Different downstream tasks. Constructing a knowledge graph for a specification document can serve different downstream tasks. For example, with a complete knowledge graph of a protocol and a number of different implementations of the protocol, we can apply fuzzing techniques [59] on the implementations and see if they perform the same. If these is any discrepancy, we can locate which part is different through traversing the knowledge graph and further analyze if the implementations are implemented correctly according to the embedded facts in the knowledge graph.

7. Conclusion

In this work, we propose NRFCKG, a neural generated knowledge graph based contradictions detection tool for IoT protocols specification documents. We train it on RFC7252 of CoAP and further apply it to MQTT and AMQP specification documents. We evaluate its performance and show the generalizing ability of this tool. We have successfully detected one direct contradiction and three conditional contradictions from CoAP, and one direct contradiction from MQTT. We analyze the detected contradictions and demonstrate that NRFCKG can serve as a general framework for this task. We further propose a future direction for better variable normalization by further pre-training Phrase-BERT with more specification documents.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

This work is funded by the university research fund of the University of Queensland, Australia.

Appendix. Rule statement extraction algorithm and direct contradiction type 1 detection algorithm

See Algorithm 1 and 2.

References

[1] Feng X, Zhang Y, Meng MH, Teo SG. Detecting contradictions from CoAP RFC based on knowledge graph. In: Proceddings of the 16th international conference on network and system security. Springer; 2022.
[2] Chegini H, Naha RK, Mahanti A, Thulasiraman P. Process automation in an IoT–fog–cloud ecosystem: A survey and taxonomy. IoT 2021;2(1):92–118.
[3] Huh S, Cho S, Kim S. Managing IoT devices using blockchain platform. In: 2017 19th international conference on advanced communication technology. IEEE; 2017, p. 464–7.
[4] Le D-P, Meng H, Su L, Yeo SL, Thing V. BIFF: A blockchain-based IoT forensics framework with identity privacy. In: TENCON 2018-2018 IEEE region 10 conference. IEEE; 2018, p. 2372–7.

[5] Mahadewa K, Wang K, Bai G, Shi L, Liu Y, Dong JS, et al. Scrutinizing implementations of smart home integrations. IEEE Trans Softw Eng 2019.
[6] Khan LU, Saad W, Han Z, Hossain E, Hong CS. Federated learning for Internet of Things: Recent advances, taxonomy, and open challenges. IEEE Commun Surv Tutor 2021.
[7] Shanthamallu US, Spanias A, Tepedelenlioglu C, Stanley M. A brief survey of machine learning methods and their sensor and IoT applications. In: 2017 8th international conference on information, intelligence, systems & applications. IEEE; 2017, p. 1–8.
[8] Xiao L, Wan X, Lu X, Zhang Y, Wu D. IoT security techniques based on machine learning: How do IoT devices use AI to enhance security? IEEE Signal Process Mag 2018;35(5):41–9.
[9] Lynggaard P, Skouby KE. Complex IoT systems as enablers for smart homes in a smart city vision. Sensors 2016;16(11):1840.
[10] Uddin H, Gibson M, Safdar GA, Kalsoom T, Ramzan N, Ur-Rehman M, et al. IoT for 5G/B5G applications in smart homes, smart cities, wearables and connected cars. In: 2019 IEEE 24th international workshop on computer aided modeling and design of communication links and networks. IEEE; 2019, p. 1–5.
[11] Singh AK. We will be surrounded by 500 billion connected devices by 2030, says anter virk of SubCom. 2022, https://opportunityindia.franchiseindia.com/article/we-will-be-surrounded-by-500-billion-connected-devices-by-2030-says-anter-virk-of-subcom-35012. [Accessed 28 August 2022].
[12] Tian C, Chen C, Duan Z, Zhao L. Differential testing of certificate validation in SSL/TLS implementations: An RFC-guided approach. ACM Trans Softw Eng Methodol 2019;28(4). http://dx.doi.org/10.1145/3355048.
[13] Andow B, Mahmud SY, Wang W, Whitaker J, Enck W, Reaves B, et al. {PolicyLint}: Investigating internal privacy policy contradictions on google play. In: 28th USENIX security symposium. 2019, p. 585–602.
[14] Xie F, Zhang Y, Yan C, Li S, Bu L, Chen K, et al. Scrutinizing privacy policy compliance of virtual personal assistant apps. In: Proceedings of the 37th IEEE/ACM international conference on automated software engineering. 2022.
[15] Wang Q, Mao Z, Wang B, Guo L. Knowledge graph embedding: A survey of approaches and applications. IEEE Trans Knowl Data Eng 2017;29(12):2724–43.
[16] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Proceedings of the 31st international conference on neural information processing systems. Red Hook, NY, USA: Curran Associates Inc.; 2017, p. 6000–10.
[17] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the North American chapter of the Association for Computational Linguistics: Human language technologies, volume 1 (Long and short papers). Minneapolis, Minnesota: Association for Computational Linguistics; 2019, p. 4171–86. http://dx.doi.org/10.18653/v1/N19-1423, URL: https://aclanthology.org/N19-1423.
[18] Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V. RoBERTa: A robustly optimized BERT pretraining approach, CoRR abs/1907.11692. 2019, arXiv:1907.11692.
[19] He P, Liu X, Gao J, Chen W. DeBERTa: Decoding-enhanced BERT with disentangled attention. 2020, http://dx.doi.org/10.48550/ARXIV.2006.03654, URL: https://arxiv.org/abs/2006.03654.
[20] Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training. 2018.
[21] Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. 2018.
[22] Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language models are few-shot learners. 2020, http://dx.doi.org/10.48550/ARXIV.2005.14165, URL: https://arxiv.org/abs/2005.14165.
[23] Rasmy L, Xiang Y, Xie Z, Tao C, Zhi D. Med-BERT: pre-trained contextualized embeddings on large-scale structured electronic health records for disease prediction. 2020, http://dx.doi.org/10.48550/ARXIV.2005.12833, URL: https://arxiv.org/abs/2005.12833.
[24] Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, et al. CodeBERT: A pre-trained model for programming and natural languages. 2020, http://dx.doi.org/10.48550/ARXIV.2002.08155, URL: https://arxiv.org/abs/2002.08155.
[25] Chalkidis I, Fergadiotis M, Malakasiotis P, Aletras N, Androutsopoulos I. LEGAL-BERT: The Muppets straight out of Law School. 2020, http://dx.doi.org/10.48550/ARXIV.2010.02559, URL: https://arxiv.org/abs/2010.02559.
[26] Li H, Li S, Sun J, Xing Z, Peng X, Liu M, et al. Improving api caveats accessibility by mining api caveats knowledge graph. In: 2018 IEEE international conference on software maintenance and evolution. IEEE; 2018, p. 183–93.
[27] Mondal I, Hou Y, Jochim C. End-to-end NLP knowledge graph construction. 2021, arXiv preprint arXiv:2106.01167.
[28] Honnibal M, Johnson M. An improved non-monotonic transition system for dependency parsing. In: Proceedings of the 2015 conference on empirical methods in natural language processing. 2015, p. 1373–8.
[29] Soares LB, FitzGerald N, Ling J, Kwiatkowski T. Matching the blanks: Distributional similarity for relation learning. 2019, arXiv preprint arXiv:1906.03158.
[30] Zhang B, Xu Y, Li J, Wang S, Ren B, Gao S. SMDM: Tackling zero-shot relation extraction with semantic max-divergence metric learning. Appl Intell 2022. http://dx.doi.org/10.1007/s10489-022-03596-z.
[31] Bradner S. Key words for use in RFCs to indicate requirement levels. 1997, https://datatracker.ietf.org/doc/html/rfc2119.
[32] Leiba B. Ambiguity of uppercase vs lowercase in RFC 2119 key words. 2017, https://datatracker.ietf.org/doc/html/rfc8174. [Assessed 04 August 2022].
[33] Xie D, Li Y, Kim M, Pham HV, Tan L, Zhang X, et al. DocTer: documentation-guided fuzzing for testing deep learning API functions. In: Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis. 2022, p. 176–88.
[34] Wang Q, Ji S, Tian Y, Zhang X, Zhao B, Kan Y, et al. {MPInspector}: A systematic and automatic approach for evaluating the security of {IoT} messaging protocols. In: 30th USENIX security symposium. 2021, p. 4205–22.
[35] Pacheco ML, von Hippel M, Weintraub B, Goldwasser D, Nita-Rotaru C. Automated attack synthesis by extracting finite state machines from protocol specification documents. 2022, arXiv preprint arXiv:2202.09470.
[36] Harabagiu S, Hickl A, Lacatusu F. Negation, contrast and contradiction in text processing. In: AAAI, vol. 6. 2006, p. 755–62.
[37] Wang K, Bai G, Dong N, Dong JS. A framework for formal analysis of privacy on SSO protocols. In: International conference on security and privacy in communication systems. Springer; 2017, p. 763–77.
[38] Mahadewa K, Zhang Y, Bai G, Bu L, Zuo Z, Fernando D, et al. Identifying privacy weaknesses from multi-party trigger-action integration platforms. In: Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis. 2021, p. 2–15.
[39] Meng MH, Zhang Q, Xia G, Zheng Y, Zhang Y, Bai G, et al. Post-GDPR threat hunting on android phones: Dissecting OS-level safeguards of user-unresettable identifiers. In: The network and distributed system security symposium. 2023.
[40] Feng X, Zhu X, Han QL, Zhou W, Wen S, Xiang Y. Detecting vulnerability on IoT device firmware: A survey. IEEE/CAA J Autom Sin 2022;10(1):25–41.
[41] Zhang J, Pan L, Han QL, Chen C, Wen S, Xiang Y. Deep learning based attack detection for cyber-physical system cybersecurity: A survey. IEEE/CAA J Autom Sin 2021;9(3):377–91.
[42] Sun N, Zhang J, Rimba P, Gao S, Zhang LY, Xiang Y. Data-driven cybersecurity incident prediction: A survey. IEEE Commun Surv Tutor 2018;21(2):1744–72.
[43] Lin G, Wen S, Han Q-L, Zhang J, Xiang Y. Software vulnerability detection using deep neural networks: a survey. Proc IEEE 2020;108(10):1825–48.
[44] Qiu J, Zhang J, Luo W, Pan L, Nepal S, Xiang Y. A survey of android malware detection with deep neural models. ACM Comput Surv 2020;53(6):1–36.
[45] Mangla C, Rani S, Herencsar N. A misbehavior detection framework for cooperative intelligent transport systems. ISA Trans 2023;132:52–60.
[46] Zhu S, Liao B, Hua Y, Zhang C, Wan F, Qing X. A transformer model with enhanced feature learning and its application in rotating machinery diagnosis. ISA Trans 2023;133:1–12.
[47] Hartke K, Richardson M. Extended tokens and stateless clients in the constrained application protocol (CoAP). 2021, https://datatracker.ietf.org/doc/rfc8974/.
[48] Selander G, Mattsson JP, Palombini F, Seitz L. Object security for constrained RESTful environments (OSCORE). 2019, https://datatracker.ietf.org/doc/rfc8613/.
[49] Shelby Z, Bormann C. Block-wise transfers in the constrained application protocol (CoAP). 2016, https://datatracker.ietf.org/doc/rfc7959/.
[50] Shelby Z, Hartke K, Bormann C. The constrained application protocol (CoAP). 2014, https://datatracker.ietf.org/doc/html/rfc7252.
[51] Coppen R, Banks A, Briggs E, Borgendale K, Gupta R. MQTT version 5.0. 2019, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf.
[52] Jeyaraman R, Telfer A, Godfrey R, Ingham D, Schloming R. OASIS advanced message queuing protocol (AMQP) version 1.0. 2012.
[53] Bird S, Klein E, Loper E. Natural language processing with Python: Analyzing text with the natural language toolkit. O'Reilly Media, Inc.; 2009.
[54] Huggingface. Huggingface. 2022, https://huggingface.co. [Accessed 30 November 2022].
[55] Ramshaw LA, Marcus MP. Text chunking using transformation-based learning. 1995, http://dx.doi.org/10.48550/ARXIV.CMP-LG/9505040, URL: https://arxiv.org/abs/cmp-lg/9505040.
[56] Wang S, Thompson L, Iyyer M. Phrase-BERT: Improved phrase embeddings from BERT with an application to corpus exploration. 2021, http://dx.doi.org/10.48550/ARXIV.2109.06304, URL: https://arxiv.org/abs/2109.06304.
[57] Moura Ld, Bjørner N. Z3: An efficient SMT solver. In: International conference on tools and algorithms for the construction and analysis of systems. Springer; 2008, p. 337–40.
[58] Hu X, Ma W, Chen C, Wen S, Zhang J, Xiang Y, et al. Event detection in online social network: Methodologies, state-of-art, and evolution. Comp Sci Rev 2022;46:100500.
[59] Zhu X, Wen S, Camtepe S, Xiang Y. Fuzzing: a survey for roadmap. ACM Comput Surv 2022;54(11s):1–36.