Checking Network Security Policy Violations via Natural Language Questions

Pinyi Shi, Yongwook Song, Zongming Fei, James Griffioen
Laboratory for Advanced Networking, University of Kentucky
Lexington, Kentucky 40506-0495, USA
Emails: {pinyishi,ywsong2,fei,griff}@netlab.uky.edu

***Abstract*—Network security policies provide high-level directives regarding acceptable and unacceptable use of the network. Organizations specify these high-level directives in policy documents written using human-readable natural language. The challenge is to convert these natural language policies to the network configurations/specifications needed to enforce the policy. Network administrators, who are responsible for enforcing the policies, typically translate the policies manually, which is a challenging and error-prone process. As a result, network operators (as well as the policy authors) often want to verify that network policies are being correctly enforced.**

In this paper, we propose *Network Policy Conversation Engine (NPCE)*, a system designed to help network operators (or policy writers) interact with the network using natural language (similar to the language used in the network policy statements themselves) to understand whether policies are being correctly enforced. The system leverages emerging big data collection and analysis techniques to record flow and packet level activity throughout the network that can be used to answer users policy questions. The system also takes advantage of recent advances in *Natural Language Processing (NLP)* to translate natural language policy questions into the corresponding network queries. To evaluate our system, we demonstrate a wide range of policy questions – inspired by actual networks policies posted on university websites – that can be asked of the system to determine if a policy violation has occurred.

I. INTRODUCTION

An organization's computer network, whether it be a small company network, a medium-size campus/enterprise network, or a large-scale ISP network, are governed by organization-specific policies that define acceptable and unacceptable uses of the network. Organizations typically establish *Policy Committee*s that are charged with writing the policies for the various networks used by the organization. The network policies produced by Policy Committees take the form of documents written in human-readable natural language that have little resemblance to the network configurations needed to enforce these policies. Historically the translation from human-readable natural language policies to low-level network configurations that enforce the policies has been done in a manual, error-prone, fashion by experienced network security staff members. Errors in translation can occur for any number of reasons ranging from misinterpretation of the policy, to human error while translating, to the inability to fully implement the policy given the network capabilities, to network complexities that result in unexpected behavior.

Recent attempts to partially automate the translation process help reduce the potential for errors by automating portions of the translation and configuration process. However, they still require a human-(expert)-in-the-loop to interpret the natural language policies and convert them to a format that can be automatically processed [1]. Related efforts to specify policy using intent-based networking [2] specifications can also simplify a network operator's job by translating from natural language policies to intents that are carried out by the intent-based network. However, such systems do not eliminate the human-in-the-loop, and the specified intents do not always map correctly onto the desired effect.

While progress has been made to improve the translation of natural language network policies into low-level network configurations that enforce those policies, policy violations are still possible. But how does one find such violations? Designing software to automatically identify and detect errors in the translation process is difficult [3], [4]. Moreover, analyzing the low-level network implementation and mapping it back to the natural language policies to see if it is correctly enforcing the policies is subject to the same types of problems as the forward translation from natural language to network configuration. The problem is further complicated by the reality that violations are sometimes allowed – i.e., policy exceptions [5]. In addition, Policy Committees can make mistakes when writing policies, mistakes that often do not become apparent until one examines the network traffic and observes unexpected behavior.

To address this problem, we describe an approach based on interactive interrogation of the network using natural language (that can be performed by both network experts and Policy Committee members) to determine if a policy violation has occurred – an approach designed to discover violations that have occurred rather than detect and/or prevent violations from occurring. For example, consider the natural language network policy "Cleartext Telnet and FTP traffic are prohibited". A network operator (or policy writer) might interactively question the network about this policy by asking any one of the following questions of the network "Is there any FTP or Telnet traffic in the network?" or more specific variations "Is there any FTP traffic in the network?" or more generally "Is there any cleartext traffic in the network?". Such questions would be posed to a database of captured network traffic and packet trace to determine whether any such traffic has

been seen by the network. Such information can help both those writing the policies and those translating them into configurations. Moreover, it leverages information about the network's actual behavior (e.g., traffic) in addition to the network's configuration.

Our approach is based on recent advances in big data analysis techniques that allow detailed information about network behavior, including the collection of flow and/or packet-level information over time, to be used to answer questions about network policy violations. Furthermore, it leverages emerging interactive conversation systems – driven by recent advances in natural language processing (NLP) and artificial intelligence (AI) – to read and respond to natural language questions from network operators or network policy writers.

Our *Network Policy Conversation Engine (NPCE)* continuously collects detailed traffic information from the network and then allows users to interactively issue policy questions, written in human-readable natural language, to the network. Given a question, the *NPCE* system automatically generates a corresponding query over the collected network traffic looking for behavior that may represent a violation.

NPCE defines a *mapping layer* that functions as the intermediary between the natural language questions and the low-level database queries. When interacting with the user, it helps parse and understand the natural language questions, but can also provide hints and instructions about useful information that would be helpful in searching for policy violations. When interacting with the underlying traffic databases, the mapping layer helps with the translation by identifying the specific information that needs to be explored and will need to be included in the generated low-level database query. To assist in this process, *NPCE* maintains an *alias file* that maps between the high-level human terms used in questions to the corresponding low-level database fields, allowing the system to be easily extended to handle new terms and policies. The system utilizes Google DialogFlow [6] to implement a conversational user interface where users can ask their questions and receive answers using natural language. *NPCE* outputs answers in a simple human-readable format that easily shows whether a policy is being violated or not, but could be enhanced to output more descriptive answers using approaches such as those used in Net2Text [7]. *NPCE* leverages common tools such as *NetFlow* [8] and *Tcpdump* [9] in combination with big data analysis system such as Logstash and Elasticsearch [10] to scalably obtain detailed network traffic information at both the flow and packet level over time from large portions of the network, thereby enabling policy analysis not possible with current systems.

To illustrate the potential of a system like *NPCE*, we collected many different types of campus network policies currently posted on various university websites, and developed natural language questions that users (operators or policy writers) might interactively ask to explore whether policy violations are occurring. We show how *NPCE* extracts the useful information from the questions, and how the information is used to generate the lower-level database queries, showing whether network security policies have been violated or not.

The rest of paper is organized as follows. Section II describes the mapping layer in *Network Policy Conversation Engine*, which focuses on the extraction of useful information in the questions. Section III illustrates the architecture of the system and how each component of the system works. Section IV presents network security policies found on various university websites and demonstrates how to use our system to answer the natural language questions with regard to these policies to check policy violations. Section V provides a literature review of related work. Section VI summarizes our findings and concluding thoughts.

II. The NPCE Mapping Layer

The role of the NPCE *mapping layer* is to translate user questions (written in human-readable natural language) into corresponding database queries (written in low-level query languages) that can identify and extract the information needed to answer the questions.

A. Design Principles

As an intermediary between the user interface and the underlying database, the mapping layer should meet the following requirements.

(1) **The mapping layer should be capable of extracting *all* the useful information in the questions related to policy violations.** Users may ask the same question in different ways. Network administrators tend to use low-level details to describe the network traffic (e.g., port 22) while other users may use high-level terms (e.g., ssh) in their network policy questions. It is vital that the mapping be able to recognize *all* these identifiers no matter whether the identifiers are high-level or low-level.

(2) **The mapping layer should clearly categorize the extracted information so that the translation from the mapping to the database queries is verifiable and straightforward**. The identification of keywords and the removal of redundant information not only simplifies the interpretation of the questions, but also make the translation process straightforward since only the relevant information needs to be placed in the corresponding fields in a query to complete the translation. The mapping itself should also be flexible enough to be translated to *any* lower-level query language based on the database.

B. Identifiers and Entities of the Mapping

The mapping layer in *NPCE* was developed to parse and categorize useful information contained in the natural language questions. We start by analyzing the format of the natural language questions to obtain the facts and events being asked about in the network.

Questions can be categorized into three groups:

(1) Questions asking about the value of an identifier that satisfies certain conditions. For example, "Which IP addresses sent packets to 10.10.1.1?" In this question, the user

Identifiers	Example Values
Object	Traffic, Packet, Flow, Byte
Device	Host, Switch, Router, Firewall, Server, Printer
Timestamp	'in the last' 'number':value (minute, hour, day, month)
Attribute	<Protocol>, 'IP':value, 'Port':value
Protocol	HTTP, FTP, BitTorrent, Telnet, TFTP, Rlogin, DNS, DHCP
Traffic Direction	incoming, outgoing, bi-directional
Comparison Operator	equal to, greater than (or equal), not equal to, less than (or equal)
Aggregation	max, min, average, count, unique count
Special Terms	the Internet, non-standard port, port scanning, IP source routing, campus <Device>, authorized <Protocol>servers and etc.
Answer	(True, False), the number of <Object>, <Attribute>
Object Description	<Attribute>, <Traffic Direction>, <Comparison Operator>, <Aggregation>, <Special Terms>
Device Description	<Attribute>, <Comparison Operator>, <Aggregation>, <Special Terms>

TABLE I
IDENTIFIERS AND EXAMPLE VALUES OF THE MAPPING LAYER

is asking about the value of the IP address that satisfies the condition "sent packets to 10.10.1.1".

(2) Questions asking about the correctness of a statement. For example, "Does host A have IP address 10.10.1.1?" In this question, the user is asking about the correctness of the statement "host A have IP address 10.10.1.1".

(3) Questions asking about the number of identifiers that meet certain conditions. There are various ways to ask this type of questions. For example, "How many packets did host A send?" and "Is there any packet sent by host A?" In the above examples, the user is interested in getting the number of packets that are sent by host A. From the perspective of queries, we need to count the number of packets, and the condition is that the packets are sent by host A. The only difference between the answers to these questions is that we should add "Yes/No" based on the value of the count of packets for the second example.

The *Five W's* is a concept to gather information about a natural language sentence [11]. In particular, it represents the following five keywords *(who, when, what, where, why)*. We only chose *what, where, when* to collect information about network traffic. The questions we should ask include "What information are we gathering?" and "When and where does the network event happen?".

Besides the three W's we selected, other information can be thought of as the descriptions. In Table I, we list the identifiers and entities for the mapping.

The *Object* identifier covers the subject of the question, representing "*what*". The *Device* identifier specifies the place in which network events happened, representing "*where*", and the *Timestamp* identifier regulates the time period, representing "*when*". When people query the status of network traffic, they may also use the low-level network identifiers, to describe the traffic as well as the network devices. These descriptions are covered by the *Object Description* identifier and the *Device Description* identifier. Both identifiers involve low-level network identifiers such as the *protocol*, the *IP address* and the *port number*. The mapping also includes identifiers that deal with traffic direction, value comparison and aggregation to increase the expressiveness. The *Special Terms* identifier represents the high-level networking terms which can also be described using the corresponding low-level network identifiers. In the body of database queries, only the low-level identifiers will appear. With the assistance of *alias files* where the low-level details of the special terms are recorded, this translation process can be automated. The information in the *alias file* is consulted each time the special terms are recognized. As have been noted earlier, the questions are categorized into different groups. The *Answer* identifier reflects the type of the answer to be returned to the users. As a result, we only need to find the answer that matches the descriptions given in the *Object Description* and the *Device Description* identifiers.

III. SYSTEM ARCHITECTURE

Network Policy Conversation Engine (NPCE), a system that takes advantage of the recent advances in NLP and modern database solutions to answer natural language questions, is shown in Fig. 1. The main components of the system are the natural language processing module, the query generation module and the big data storage and analysis database used to store detailed information about network traffic.

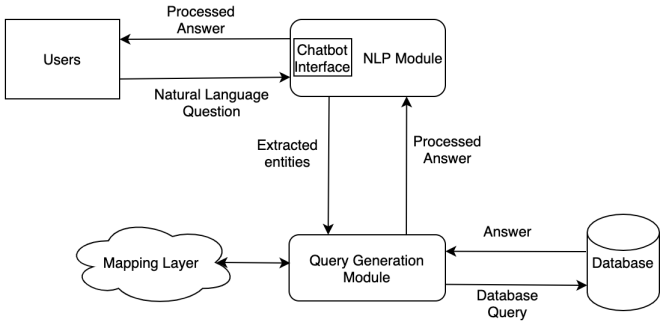

Fig. 1. NPCE System Architecture

A. The NLP Module and Entity Extraction

Entity extraction is a NLP technique to extract and classify important words in the text (entities) into pre-defined entity sets from the unstructured text. It is the first step toward understanding the user question, where the useful information is extracted and the redundant information is ignored. The accuracy of this step is dependent on the completeness and correctness of the pre-defined entity sets. In our system, the pre-defined entity sets are based on the identifiers specified in the mapping layer. We collect a list of synonyms for each entity type. When the user asks a question which contains keywords that are exactly the same as, or similar to, the pre-defined entities, the NLP module is able to detect and

recognize them. The output of this module is a list of entities consisting of an entity name and the resolved value.

B. The Query Generation Module

Another important part of the system is the query generation module whose job is to take as input the output from the NLP module and process it to generate the database query. First, the module must determine how to use the detected low-level details. For example, the module takes into account the information of the *Traffic Direction* identifier and the order in which the IP addresses or port numbers appear to determine the correct fields the detailed information should be placed in.

Second, the module must pass the information – e.g. the 5-tuple values – to the query and issue REST API calls to the database. The basic query syntax for such calls involves a simple match statement. More advanced queries involve comparison operators and aggregation functions, and enable queries to be generated for a wide range of potential natural language questions.

C. Capturing and Storing Network Traffic

A number of tools and services have been developed to capture network traffic. In general, they focus on capturing network traffic at either the flow-level or the packet-level. For example, *NetFlow* is designed to capture network traffic at the flow level by aggregating the packets with the same 5-tuple into compact records describing the observed flows. Tools such as *Tcpdump* can be used to inspect the header information of every packet. Both approaches have their own advantages since organizations typically select the tools based on their needs. As a proof-of-concept, we assume that both packet-level and flow-level traffic capturing tools are available for us to use and the database provides adequate space to store the information of the captured traffic.

The challenge is to collect flow and packet-level data at scale. Fortunately, emerging big data collection and analysis systems offer services capable of collecting, filtering, compacting, storing, and later searching or analyzing flow and packet-level data at scale. NPCE leverages the ELK stack [10] for this task. At the heart of ELK is Elasticsearch, a search and analytic engine that provides excellent performance and scalability and is built on Apache Lucene [12]. The syntax of the Elasticsearch query DSL (Domain Specific Language) is human-readable, which makes it easy to generate the queries from the entities returned by the NLP module. Moreover, it supports several "beats" that can efficiently collect data about packets and flows, ranging from Packetbeat [13] that collect NetFlow-style data (or we can use Netflow directly if the device supports it) as well as filebeats that can read and parse TCPdump files. Collected data is efficiently imported into the Elasticsearch database where it can be queried and analyzed (possibly passing through Logstash if filtering is required).

As an aside, note that *network intrusion detection systems (NIDS)*, designed to detect attacks, serve a different purpose. As such, IDS that uses signature-based detection mechanism, such as Snort [14] and Suricata [15], can generate real-time alerts once the captured traffic matches attack traffic signatures. IDS systems based on anomaly detection such as Zeek [16] can analyze packet capture files and create log files categorized by the protocol name and look for anomalies. Such information could also be ingested into NPCE and could be useful for finding network policy violations, but the data being collected is usually specified in such a way that attacks or anomalous behavior can be identified, not necessarily policy violations.

IV. Prototype Implementation

In this section, we present the test topology we set up and the network security policies found on various university websites to demonstrate the capabilities of *NPCE*. We show for each policy the questions that can be asked to check the violations and how they are translated to the *Elasticsearch queries*.

As shown in Figure 2, we set up the topology in the Global Environment for Network Innovations (GENI) platform [17], where people can conduct networking research at scale. We include an OpenVSwitch (OVS) in the topology to use its Netflow module to capture flow information of network traffic. The other components of the prototype implementation are the same as have been discussed in the previous section. In particular, we took advantage of Google Dialogflow to train and extract the entities. The reason we picked Dialogflow out of various natural language understanding platforms is that it can be integrated with many popular messaging applications so that people can ask questions in the applications they use daily. We wrote a web server using Flask [18] to work as the query generation module. For the selection of the database, our choice was Elasticsearch, which is built on Apache Lucene. The log files of *NetFlow* and *Tcpdump* were sent to the Elasticsearch server for analysis.

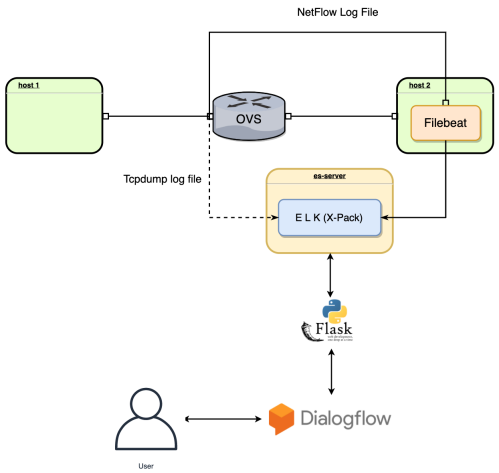

Fig. 2. Test Topology for Prototype Implementation

The entire workflow starts from the natural language questions asked by the user. Dialogflow takes the questions and outputs a list of entities. These entities are categorized and sent

to the web server to be translated into the corresponding *Elasticsearch queries*. Once these queries are issued to the database through REST API calls, the user will get the answers in the responses. The remaining part of this section demonstrates the capabilities of *NPCE* with real-world network security policies found on various university and organization websites.

A. Example Network Policies and Use Cases

We show how the natural language questions that check policy violations can be translated to the *Elasticsearch queries* step by step. Due to space limitation, we will only discuss the related fields in an *Elasticsearch query*.

1) Policies regarding Insecure Application Layer Protocols: The vulnerabilities in the application layer protocols are often exploited by hackers to perform security attacks. As a result, network security policies always require disabling these insecure protocols. We list below the network security policies on the websites of the University of Missouri at St. Louis, University of Birmingham and Indiana University:

> *"Applications which transmit sensitive information over the network in clear text, such as telnet and ftp, are prohibited and will be blocked."* [19]

> *"The University Wireless Network should not be used inappropriately; in particular you should not use the network to: run peer-to-peer (P2P) file sharing software, e.g. BitTorrent."* [20]

> *"For a computer system to be managed securely, functional unit technicians must: Disable or secure remote access from system-to-system (e.g., rlogin)."* [21]

For these types of policies, users may ask questions in a similar way to check violations. Taking the first policy for example:

"Is there any ftp or telnet traffic in the network?"

In the question, only the protocol names are used to describe the traffic. If device and timestamp are not mentioned in the question, the query generation module assumes that the user wants to query the data from *all* the switches in the network at *any* time. The query is performed on all the collected network traffic data to examine whether any policy violation has *ever* happened in the network. For each application layer protocol, an *alias file* records the corresponding low-level details such as the transport layer protocol and destination port numbers. They are used when the module generates the corresponding Elasticsearch query.

Figure 3 shows the question and the *alias file* to check the existence of FTP or Telnet traffic in the network. In the *alias file*, the corresponding destination port numbers and transport layer protocols are recorded. The extracted information indicates what *Answer* will be returned to the user and what filters to use when generating the database query.

Here the protocols and destination port numbers in the *alias file* are used to build the *Elasticsearch query* as shown in Figure 4.

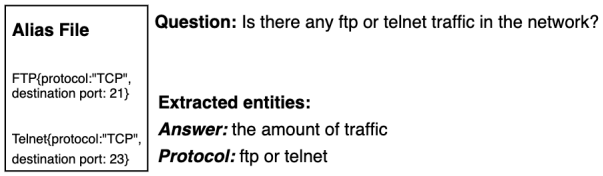

Fig. 3. Question and Extracted Entities for FTP and Telnet Traffic

```
"match_phrase": {
  "network.transport": "tcp"
}
"bool": {
  "should": [
    {
      "match_phrase": {
        "destination.port": "21"
      }
    },
    {
      "match_phrase": {
        "destination.port": "23"
      }
    }
  ],
}
```

Fig. 4. Elasticsearch Query to Check FTP and Telnet Traffic

2) Policies regarding Prohibited Services: Some types of traffic are prohibited everywhere in the network while others are only disallowed in a selected area of the network. We found network policies about rogue servers and non-standard ports on the websites of Villanova University and Salem State University:

> *"Network usage judged appropriate by the University is permitted. Some activities deemed inappropriate include, but are not limited to: Attaching unauthorized network devices, including but not limited to wireless routers, gateways DHCP or DNS servers; or a computer set up to act like such a device."* [22]

> *"Most network services through non-standard ports are not supported. Services through non-standard ports may be restricted to a limited number of subnets or hosts. For example, WWW access via the standard HTTP port will be permitted, but via some other arbitrary port number may not be permitted."* [23]

The question and extracted entities for the first policy is shown in Figure 5. In this question, the comparison operator and the IP addresses are the additional information. The authorized server is recognized as a *special term* so that the IP addresses of the servers are fetched from the alias file.

Based on the direction of the traffic, the IP addresses of the authorized servers are used as the destination IP addresses. Because of the existence of "not" in the question, the destination IP addresses is placed in the "must_not" field to generate the *Elasticsearch query* as shown in Figure 6.

For the second policy, users may ask the following question:

"Is there any campus server that has incoming traffic on

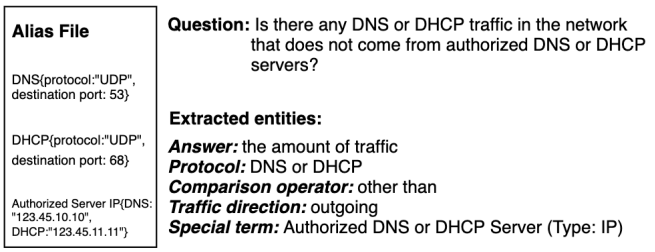

Fig. 5. Questions and Extracted Entities for the Policy about Rogue Servers

```
"match_phrase": {
   "network.transport": "udp"
 }
 "bool": {
    "should": [
      {
        "match_phrase": {
          "destination.port": "53"
        }
      },
      {
        "match_phrase": {
          "destination.port": "68"
        }
      }
    ],
  }
"must_not": [
{
  "bool": {
    "should": [
      {
        "match_phrase": {
          "source.ip": "123.45.10.10"
        }
      },
      {
        "match_phrase": {
          "source.ip": "123.45.11.11"
        }
      }
    ],
  }
 }
]
```

Fig. 6. Elasticsearch Query to Check the Existence of Rogue Servers

non-standard ports?"

In the alias file, the IP addresses of the campus servers and the standard port of the application protocols can be found. Based on the direction of the traffic, the IP addresses of the servers are used as the destination IP addresses and the port numbers are used as the destination port numbers. Additionally, the comparison operator determines that the destination port numbers are placed in the "must_not" field in the *Elasticsearch query*.

3) Policy regarding Device Access Control: Access control policies regulate whether some network devices are accessible by other devices. Here, we list the policies regarding the accessibility of campus printers found on the website of UC Berkeley:

> *"Campus printers should not be exposed to the public Internet."* [24]

We show the question and the extracted entities in Figure 7. This question is trying to figure out the amount of traffic between two IP ranges. The *alias file* records the IP addresses of the campus printers as well as the campus IP range. The Internet represents all the IP addresses *other than* those specified in the campus IP range. The comparison operator "other than" is used to get the IP range of the Internet. The direction of the traffic also determines the IP range of the Internet that is used as the source IP address.

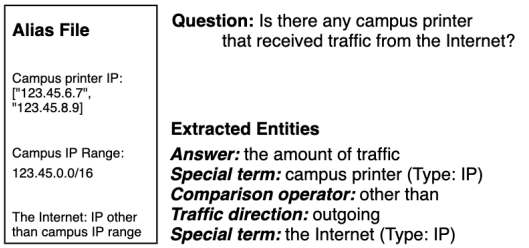

Fig. 7. Question and Extracted Entities for the Policy about Campus Printer

The IP addresses of the campus printers are placed in the "should" field for the match of destinations. We used regular expressions to show the IP range in the CIDR notation. The expression is placed in the "must_not" field which reflects the comparison operator "other than". The related *Elasticsearch query* is shown in Figure 8.

```
      "must_not": {
        "bool": {
          "should": [
            {
              "query_string": {
                "fields": [
                  "source.ip"
                ],
                "query": "\\1\\2\\3\\.\\4\\5\\.*"
              }
            }
          ],
        }
      }
{
  "bool": {
    "should": [
      {
        "match_phrase": {
          "destination.ip": "123.45.6.7"
        }
      },
      {
        "match_phrase": {
          "destination.ip": "123.45.8.9"
        }
      }
    ],
  }
}
```

Fig. 8. Elasticsearch Query for Campus Printer Access Policy

4) Policy regarding Port Scanning: Port scanning is a method of detecting the open ports on network devices. Based on the characteristic of the generated network traffic, port scanning can be described as *a connection that attempts to send traffic to a large number of ports within a time period*. On the website of University of Louisiana at Lafayette, we found a policy regarding port scanning:

> *"Port scanning or security scanning is expressly prohibited unless prior notification to Information Technology Security is made."* [25]

There are various ways to ask questions that check violations against this policy. The reason lies in the fact that port scanning is a high-level term and the traffic generated by port scanning has the corresponding low-level details. People who do not understand the details of port scanning may directly

ask questions as *"Is there any port scanning traffic in the network?"*. On the other hand, people who understand port scanning may ask questions using the low-level details, such as *"Did any host in the network send traffic to more than 500 different ports?"*. To deal with both types of questions, we exploit the use of the *alias file* where the definition of port scanning is saved. The number of different ports and the time period are defined by the organization based on the network topology and requirements. The question along with the extracted entities can be found in Figure 9.

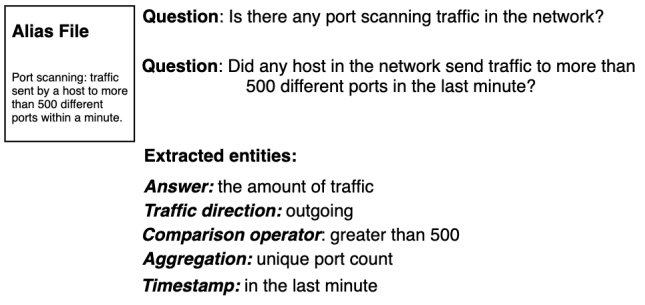

Fig. 9. Question and Extracted Entities for the Policy about Port Scanning

Based on the direction of the traffic, we are trying to find the source IP address that sent traffic to more than 500 different destination port numbers. In Elasticsearch, we use *cardinality* for the unique count, and *bucketselector* to express the comparison relationship in the *agg* field. The corresponding Elasticsearch query is shown in Figure 10.

```
"aggs": {
  "traffic": {
    "terms": {
      "field": "source.ip.keyword"
    },
    "aggs": {
      "count_dst_port": {
        "cardinality": {
          "field": "destination.port"
        }
      },
      "comparison": {
        "bucket_selector": {
          "buckets_path": {
            "dstport_count":"count_dst_port"
          },
          "script": "params.dstport_count>500"
        }
      }
    }
  }
}
```

Fig. 10. Elasticsearch Query Example for Port Scanning Traffic

5) Policy regarding IP Source Routing: Policy violations shown in the previous sections can be checked using the data collected by *NetFlow* since only the fields in the 5-tuple are used as the filter to generate the query. However, checking some policy violations requires analyzing the specific header information of each packet. *IP source routing* is an approach where people can specify the route that the packets will take to reach the destination. Using this approach, attackers can get the information of the network in the response. We found a policy that requires disabling *IP source routing* on the network devices from the website of the SANS Institute as follows:

> *"The following services or features must be disabled: All source routing and switching."* [26]

To identify the *IP source routing* traffic, the *IP Option field* in the packet header needs to be checked. We collected the data using *Tcpdump* and convert the .pcap file into a .json file which has additional information showing the value of the *IP Option field*. Then the .json file is imported to *Elasticsearch* for the queries. The alias file records the details of IP source routing and the values are the codes for the two types of source routing, namely *Loose Source Routing (LSR)* and *Strict Source Routing (SSR)* in the IP Option field as shown in Figure 11. The corresponding *Elasticsearch query* is shown in Figure 12.

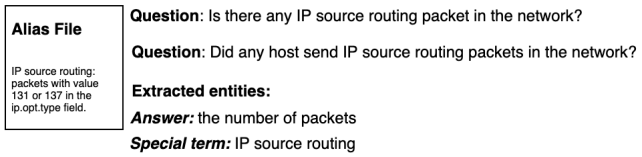

Fig. 11. Question and Extracted Entities for the policy regarding IP source routing

```
"bool": {
  "should": [
    {
      "match_phrase": {
        "layers.ip.ip_options_route_ip_opt_type": "131"
      }
    },
    {
      "match_phrase": {
        "layers.ip.ip_options_route_ip_opt_type": "137"
      }
    }
  ],
}
```

Fig. 12. Elasticsearch Query for IP Source Routing Packets

V. Related Work

Various abstractions and intent definition languages have been proposed to simplify network policy enforcement. Jinjing, an intent framework proposed for the accurate configuration of Access Control List (ACL), comes along with the intent definition language, namely LAI (Language for ACL Intents) [27]. This intent language uses different primitives to define the actions to take on various devices for the ACL. We argue that *NPCE* is more general. It provides users with a natural language interface so that the users do not need to learn the syntax of the intermediate intent definition language. Nile, an intent definition language proposed in [28], has a powerful template that reflects the intents of the various network policies. Similar to our work, their proposed refinement process also starts with natural language. However, Nile only focuses on the intent refinement process while *NPCE* not only takes the natural language as input but also generates the database

queries, which makes the entire process automated. Policy Graph Abstraction (PGA) [29] provides an approach where network administrators can specify network policies using graphs. The focus of the paper is on how to solve the conflicts among graphs when people in different departments compose graphs simultaneously. The state of the network needs to be considered when network administrators enforce the policies. Unlike policy enforcement, checking policy violations can be done by multiple people at any time simultaneously.

Other work deals with fetching network information using natural language questions. Net2Text [7] helps the network operators find the routing and forwarding behaviors of network traffic, while in *NPCE*, we focus on whether network policies are violated. We derive the special terms from policy documents and find the traffic described by these terms. The work described in [30] presents an abstraction layer that can be used to either enforce a network policy or get answers to network queries. The queries of the users are first mapped to various abstraction tasks and then fulfilled using the SDN controller. However, we argue that *NPCE* allows the users to look at the history of the network traffic data to determine whether any policy violation has *ever* happened, while using SDN controller one can only know the status of the network at that instant.

Also related is the work focusing on translating the natural language questions into the corresponding SQL queries [31], [32]. Various machine learning models such as reinforcement learning are applied to achieve high translation accuracy. We argue that *NPCE* fits the context of checking network policy violations better since it contains domain specific knowledge in the networking area that cannot be processed by other general purpose translation schemes. The existence of the mapping layer also provides flexibility on the tools an organization can use to store network traffic data.

VI. Conclusion

In this paper, we proposed an approach to use natural language queries to check network security policy violations. The proposed *Network Policy Conversation Engine* system provides users with a friendly chatbot interface to ask questions in a natural language without knowing the details of the queries. It translates users natural language questions into network traffic database queries on behalf of the users so that even the people who do not have sufficient query programming skills can still use the system to check policy violations. The mapping layer, which works as an intermediate agent, ensures that users questions can be translated into various database queries . To evaluate *NPCE*, we took network security policies from various University websites and asked questions related to these policies. The results showed that useful information in the natural language questions could be smoothly extracted to build the *Elasticsearch queries* that fetch answers to the questions from the database, so that we can check whether these network policies have been violated or not.

VII. Acknowledgment

This work was supported in part by the National Science Foundation under Grant ACI-1642134.

References

[1] S. Rivera, Z. Fei, and J. Griffioen, "POLANCO: Enforcing Natural Language Network Policies," in *Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020.

[2] "Cisco whitepaper: Intent-based Networking, Building the bridge between business and IT," https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/digital-network-architecture/nb-09-intent-networking-wp-cte-en.pdf.

[3] D. Farrar, J. Huffman Hayes, G. Adkins, J. Griffioen, and C. Bumgardner, "NetSecOps and Policy Checking An Application of Traceability Techniques," in *In Proceedings of Grand Challenges of Traceability 2017*, 2017.

[4] J. H. Hayes, "Towards Improved Network Security Requirements and Policy: Domain-Specific Completeness Analysis via Topic Modeling," in *2020 IEEE 28th International Requirements Engineering Conference Workshops (REW)*, 2020.

[5] J. Griffioen, Z. Fei, S. Rivera, J. Chappell, M. Hayashida, P. Shi, C. Carpenter, Y. Song, B. Chitre, H. Nasir *et al.*, "Leveraging SDN to Enable Short-term On-demand Security Exceptions," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 13–18.

[6] "Dialogflow, Create Conversational Experiences Across Devices and Platforms," https://cloud.google.com/dialogflow.

[7] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Net2text: Query-guided Summarization of Network Forwarding Behaviors," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018, pp. 609–623.

[8] "Introduction to Cisco IOS NetFlow - A Technical Overview," https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html.

[9] "Tcpdump/Libpcap Public Repository," https://www.tcpdump.org/.

[10] "What is The ELK Stack?" https://www.elastic.co/what-is/elk-stack.

[11] Z. Zhang, B. Wang, F. Ahmed, I. Ramakrishnan, R. Zhao, A. Viccellio, and K. Mueller, "The Five Ws for Information Visualization with Application to Healthcare Informatics," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 11, pp. 1895–1910, 2013.

[12] "Ultra-fast Search Library: Lucene," https://lucene.apache.org/core/.

[13] "Lightweight Shipper for Network Data," https://www.elastic.co/beats/packetbeat/.

[14] "What is Snort?" https://www.snort.org/.

[15] "Suricata, Open Source IDS/IPS/NSM Engine," https://suricata-ids.org/.

[16] "Zeek Manual," https://docs.zeek.org/en/master/.

[17] "What is Geni?" https://www.geni.net/about-geni/what-is-geni/.

[18] "Flask, Web Development One Drop At a Time," https://flask.palletsprojects.com/en/1.1.x/.

[19] "UMSL Network Policies," https://www.umsl.edu/technology/networking/networkpolicy.html.

[20] "University of Birmingham Wireless Network Policy," https://intranet.birmingham.ac.uk/it/teams/infrastructure/core/wireless/help/policy.aspx.

[21] "Indiana University Security of Information Technology Resources," https://policies.iu.edu/policies/it-12-security-it-resources/index.html.

[22] "Villanova University Network Security Policy," https://www1.villanova.edu/villanova/unit/policies/AcceptableUse/security.html.

[23] "Salem State University Network Security Policy," https://records.salemstate.edu/sites/records/files/policiesNetwork20Security20Policy.pdf.

[24] "UC Berkeley Network Printer Security Best Practices," https://security.berkeley.edu/education-awareness/best-practices-how-articles/system-application-security/network-printer-security.

[25] "University of Louisiana at Lafayette Security and Accetable Use Policies," https://helpdesk.louisiana.edu/sites/helpdesk/files/IT_Security_Policy.pdf.

[26] "The Sans Institute Router and Switch Security Policy," https://assets.contentstack.io/v3/assets/blt36c2e63521272fdc/blt6cbaf88421cd16f6/5e9dfac0674ec260f325c430/router_and_switch_security_policy.pdf.

[27] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang *et al.*, "Safely and automatically updating in-network acl configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 214–226.

[28] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, 2018, pp. 15–21.

[29] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using Graphs to Express and Automatically Reconcile Network Policies," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 29–42, 2015.

[30] A. Alsudais and E. Keller, "Hey Network, Can You Understand Me?" in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2017, pp. 193–198.

[31] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating Structured Queries from Natural Language using Reinforcement Learning," *arXiv preprint arXiv:1709.00103*, 2017.

[32] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev, "Typesql: Knowledge-based Type-aware Neural Text-to-sql Generation," *arXiv preprint arXiv:1804.09769*, 2018.