Automated Synthesis of Access Control Lists

Xiao Liu, Brett Holden, and Dinghao Wu
College of Information Sciences and Technology
The Pennsylvania State University, University Park, PA 16802, USA
{*xvl5190, bah5423, dwu*}*@ist.psu.edu*

***Abstract*—Network configuration remains time-consuming and error-prone with the current configuration command system. To create access control lists (ACLs) with commands containing many options is still considered as a difficult task. In light of this, we aim to develop a comprehensible way to the ACL construction. Based on Eliza, a prototype of Artificial Intelligence, we propose a new design called EASYACL that synthesizes ACL rules automatically from natural language descriptions. EASYACL demonstrates the effectiveness of domain-specific program synthesis. Through the use of natural language, ACL rules can be constructed without using an excessive number of options or rigid syntax. By introducing the batch processing, we make it possible for users to apply configurations to a range of IP addresses rather than tediously repeating commands. EASYACL supports multi-platform by an intermediate representation which may be ported to the commands for both Cisco and Juniper devices. The comprehensible commands are friendly for encapsulation as well as reuse. EASYACL enables end-users with no prior programming experience to construct ACL in a natural way which lowers the bar for security management training and also reduces the errors in network administration.**

1. Introduction

An Access Control List (ACL), with respect to the network system, is a list of permissions attached to a certain network [1]. Configuration mistakes can cause network outages, degradation in performance, and security vulnerabilities. For example, installing the wrong packet filter, filtering valid routes, advertising an incorrect block of IP addresses, or assigning the same IP address to multiple pieces of equipment, can all lead to reachability problems [2].

With an empirical analysis of the configuration process plus a thorough survey with network administrators in our department, a few complications are the main troublemakers. The misuse of configuration options is one of the most issues that were brought up. For ACL configuration commands, options are critical which enlarge the semantic sets with succinct syntax. However, the large number of abbreviated options are disturbing for network administrators, especially those who are not apprenticed. Take the IP permit command as an example, there are in total six options that it would accept. To permit the network flows of a specific type needs a few options filled out by the administrator who is required to read the specification carefully to avoid possible mistakes. Listing 1 demonstrates the IP permit command syntax [3] and the details of each option is described in Table 1. To write a correct **permit** commands are difficult with these complex options to consider. Additionally, the command is case-sensitive. Therefore, it is very likely for a careless administrator to misuse an option or two.

```
permit {gre | icmp | tcp | udp | ip | proto−num} {
source−ip [wildcard] | host source−ip | any} {dest−ip
[wildcard] | host dest−ip | any}
```

On the other hand, the platform dependency of syntax makes the case more complicated. Network companies hold their own operating systems and with the design of platform-directed syntaxes, the ACL configurations are entirely different. Cisco and Juniper are two main network device manufacturers and they have their own configuration syntaxes respectively. If one wants to permit *HTTP* traffic from IP *172.21.1.1* to IP *172.21.1.15*, the Cisco configuration and Juniper configuration are as follows:

```
# Cisco command
permit tcp host 172.21.1.1 host 172.21.1.15 eq 443

# Juniper command
filter 1 {
  term T1 {
    from {
      source−address {
        172.21.1.1/32; }
      destination−address {
        172.21.1.15/32; }
      protocol tcp; destination−port 443;
    } then {
    accept; }
  }
}
```

As demonstrated in this example, the syntax of Cisco ACL configuration commands is distinct from what Juniper system adopts. While Cisco commands are designed to be imperative, Juniper's are object-oriented [4]. Because of these distinctions, people have to learn different syntaxes when they change platforms which brings more hurdles to networking engineering training process.

Formal language synthesis is discussed over a few applications, including spreadsheet commands [5], SQL queries [6], and even input grammars [7]. In this study, to lower the bar of entry for network administration training and reduce the configuration complexity, we propose EASYACL, a tool that synthesizes ACL configuration commands for different platforms directly from natural language descriptions. To overcome the challenge from option specifications, we introduce a natural language interpretation system which accepts descriptions in a considerable flexibility and

TABLE 1: Details for IP **permit** command syntax

Keywords	Details
host	Matches the following IP address.
any	Matches any IP address.
gre	Matches packets using the Generic Routing Encapsulation protocol.
ip	Matches all IP packets.
tcp	Matches packets using the TCP protocol.
udp	Matches packets using the UDP protocol.
dest-ip	Destination IP address.
icmp	Matches ICMP packets.
proto-num (Optional)	IP protocol number.
icmp-type (Optional)	Matches by ICMP message type (0255).
code (Optional)	Used with icmp-type to further match by ICMP code type(0255).
operator (Optional)	Operator to use with specified ports.
port (Optional)	Port, using a number (065535) or a keyword.
established (Optional)	Matches TCP packets with the acknowledgment or reset bits set.
icmp-msg (Optional)	Matches by a combination of ICMP message type and code types.

synthesize the target commands directly by extracting the semantics with a rule-based natural language processing method. To avoid redundant training process, EasyACL can port the synthesized commands to different platforms, namely, Cisco and Juniper, for the current stage. We also demonstrate the practical usage of troubleshooting existing ACL configuration errors through a common mistake among network engineers. Our tool is the *first* that synthesizes ACL rules from *natural language descriptions* and proposes a *unified* interpretation system that connects different platforms. We also provide a *rangelist* function that allows users to perform batch updates for the network configuration. This function remarkably reduced the workload when same configurations for a range IP addresses are performed.

The rest of this paper is organized as follows. We describe the overview with a clear problem statement and our basic idea in Section 2. We also provide a running example for demonstrating the basic idea. We detail the designs of each module in our method and explain the rationales together with how we implement in practice in Section 4. Then, we discuss the usefulness and usability of the proposed tool with real-world use cases in Section 5. We conduct discussions over the current implementation in Section 6 and we draw the conclusion in Section 7.

2. Overview

We are proposing a tool that synthesizes commands P in the ACL configuration syntaxes from natural language descriptions N. Essentially, our goal is to 1) extract the semantics of natural language described commands and interpret them into an abstracted intermediate language; 2) port the intermediate representation to a specific platform with a target syntax. By design, there are two main challenges: natural language processing and semantic abstraction.

To be more specific, the first step of the proposed tool is to understand natural language descriptions. *Natural Language Processing* is widely discussed by researchers with kinds of solutions, either statistical-based [8], or rule-based [9], [10]. Recently, researchers are prone to adopt statistic-based methods when there are adequate data to train and test. However, the rule-based method is more efficient in specific domains since it does not require many computational resources, and error analysis is more natural to perform. In our scenario, it requires the system to understand human descriptions over a specific domain. Therefore, the variations of descriptions are limited. Due to the lack of training data and targeting on boosting a fast solution, we initialize our idea with a rule-based natural language processing system. The extracted semantics will be marked with an IR (intermediate representation) and then ported to commands in the target syntax specified by the user. We will use a running example to show the entire working flow.

3. A Running Example

We have proposed a tool that synthesizes ACL configuration commands from natural language descriptions. Additionally, the synthesized commands should be actively ported to different platforms with a corresponding syntax. For example, if we want to create a list of ACL rules:
Permit the RDP traffic for port 80 from IP address 192.168.0.11 to all the others.

In the first step, our tool should understand that, this is a permit rule and the kind of traffic it wants to permit is RDP traffic. The permission is performed over port 80 and the source IP address is 192.168.0.11. All the other IP addresses are specified as the destination. Therefore, our tool will generate the commands, such that:

```
type: permit
traffic: RDP
port: 80
source: 192.168.0.11
destination: Others
```

We need a sound and complete interpretation of the natural language description which can later be precisely ported to a specific platform, either Cisco or Juniper in our current design. Thus, we have

```
# Cisco
permit tcp 192.168.0.11 any eq 80
#Juniper
filter 1 {
  term T1 {
    from {
      protocol rdp;
      destination−port 80;
    }
    then {
      permit;
    }
  }
}
```

If there is an ambiguity in the natural language description, our tool will actively inquire for confirmation before generating the target commands. By providing a simple way that network engineers to configure access control lists, we lower the entry bar for the training process and create a method for cross-platform configurations. It will potentially reduce the error rate for network security engineering.

4. Design

EasyACL is simply a system for creating network access control lists. EasyACL operates by receiving natural language inputs, interpreting such inputs, and responding with two sequential and distinct outputs: a natural language response which indicates understanding and initiation of the end-user's desired action, and the synthesis of the correctly formatted command for such action. In this section, we will detail the techniques leveraged in building EasyACL. As demonstrated in the running example, EasyACL extracts the semantics from commands in natural language and interprets into system commands for different platforms. To better support different platforms, we incorporated an *IR* (Intermediate Representation) into the system. In addition, we also leverage a *rangelist* for batch updates which makes the system more user-friendly.

4.1. Natural Language Processing

To extract the semantics from natural language descriptions, EasyACL utilizes a rule-based method originated from Eliza [11], a primitive AI prototype. The system is built upon the assumption that structures of natural language descriptions in a specific domain are limited. We leverage this heuristic for constructing rules that can extract the intrinsic semantics of sentences. There are two types of rules in Eliza, the decomposition rules and the reassemble rules. As shown in the following example, the decomposition rules are used to decompose the complete sentences into tokens. We defined four kinds of tokens: *Number Token*, *Predicate Token*, *Option Token*, and *Redundant Token* to match different parts in a sentence. We in total construct 73 decomposition rules for EasyACL which accepts most descriptions that one network engineer may say when configuring the access control list. In addition to the decomposition rules, we also adopt the reassemble rules for interaction purpose. Eliza leverages the reassemble rules in its system for composing interactive responses. Essentially, the reassemble rules are some incorporated templates that can be based on to synthesize the natural language feedbacks. There are some dynamic changing parts in the templates which can be replaced by extracted tokens from the input sentences. For instance, if Eliza heard "I love dogs" from a user, it will apply the reassemble rule "What are (Token)" together with the extracted token "dog" and synthesize the response "What are dogs". To generate interactive feedbacks for command confirmation, we incorporate 73 reassemble rules in the system. To make things simpler, we build such rules

TABLE 2: Context-free Grammar for ACL language

command	:=	command—command command
command	:=	type option source destination
type	:=	permit—deny
option	:=	port|protocol
source	:=	ip|range|any
destination	:=	ip|range|any
port	:=	number
protocol	:=	TCP|IP|ICMP|...
ip	:=	[0-255].[0-255].[0-255].[0-255] [wildcard]
range	:=	ip-ip
any	:=	[0.0.0.0-255.255.255.255]

one-to-one corresponding to the decomposition rules which is different from the original Eliza framework.

Decomposition Rules
Permit the RDP traffic for port 80
(Predicate)(Redundant)(Option)(Redundant)(Option)+(Number)
from IP address 192.168.0.11 to all the others.
(Predicate)(Option)(Number)(Predicate)(Number)

4.2. Intermediate Representation

As described, we adopt a rule-based method to extract the semantics from the natural language descriptions. An important feature which sets distinguishes EasyACL is the generation of cross-platform output; this being both the Cisco IOS and Juniper Junos syntaxes for the current system. This enables users who are using a very wide range of networking devices, and helps users who may need to convert from one syntax to another. This includes helping a user to utilize his or her preexisting knowledge of a syntax to learn another syntax or to adapt their existing ACL rules.

We incorporate an *IR* (Intermediate Representation) in EasyACL, which is a context-free language as shown in Table 2. It is a superset of the complete sets of Cisco and Juniper commands which include in total two predicate types (*permit*, *deny*). We first extract the semantics from the natural language descriptions and then synthesize the IR commands. The IR commands are then ported to specific platforms requested by the user.

4.3. Rangelist

In practice, engineers commonly complain that router interfaces do not support network ranges. A crucial feature of EasyACL is its ability to handle ranges of IP addresses, network summarizations and using them in ACL configuration generation. The depending size of the network range and its specific summarization, often results in the output of several lines of rules. The capability of EasyACL to handle ranges of IP addresses in its interpretation of natural language is hugely beneficial to the user. This feature empowers end-users who have limited computer networking knowledge and would otherwise be thwarted by the need to create summaries.

A concise representation of an IP address refers to the number of groups used to list addresses. For example, while the program could simply produce one line or rather one

ACL configuration rule for each address listed in the range, this is inefficient for the program, and tedious for the end-user if he or she needs to add the rules to an access control list. Furthermore, it is important for ACLs to be as concise as possible in order to remain efficient and not exceed the maximum allowed length.

The range handling of EASYACL is contained within the `rangelist` function. This component takes a range of end-user inputted IP addresses and converts them into appropriately grouped and formatted networks for the synthesis of access control lists. Ranges of IP addresses can be most appropriately represented by being both precise and concise. It is important that IP address ranges are precise so that only the specified addresses are included in the synthesized output. In simple terms, this means that if a user enters the address range "192.168.1.0 - 192.168.1.10", this must only include those addresses, not for instance, another address in that networks subnet such as "192.168.1.15".

The `rangelist` function accomplishes the necessary precision and conciseness using Variable Length Subnet Masking (VLSM). The result in the above example would be the summarization of "192.168.1.0 - 192.168.1.10" as "192.168.1.0/29, 192.168.1.8/31, 192.138.1.10/32". In addition to calculating the VLSM summarization of the end-user inputted IP address range, it also calculates the inverse of the subnet mask, and converts the CIDR notation to an ordinary four-byte mask, as required by IOS syntax for access control lists. The result of the "rangelist" function is an IP address range in a usable format for ACL synthesis. This component can be used to implement many abstract actions in a batch across the desired range of IP addresses.

5. Case Study

Normally, Access Control Lists allow or deny packets according to the source address, destination address, type of packet, or any combination of these requirements. In this section, we perform a case study over practical ACL configurations. To demonstrate the ability to synthesize standard ACL rules and conduct rangelist summary, we have collected two practical Access Control Lists from the CCNA Lab materials [12]. In addition, we also demonstrate the ability to troubleshoot ACL errors with a practical implementation error.

5.1. Commands Synthesis

We propose EASYACL, a tool that synthesizes ACL rules from natural language descriptions, to lower the entrance bar for the network engineering training process. In this section, we perform the case studies over two examples from CCNA practice. The first example is a standard network permit and the second one has a rangelist.

Example 1. The networks 172.16.0.0/16 and 172.17.0.0/16 should be permitted access, with all others denied.

This example is a standard Access Control List with two permit commands. By providing EASYACL with this natural language described commands, our tool will understand that only packets from two sources are permitted. Therefore, it will synthesize the IR,

```
type: permit
traffic: ANY
port: ANY
source: 172.16.0.0/16 172.17.0.0/16
destination: ANY.
```

By specifying the Cisco syntax as the target, EASYACL will first interpret the subnet mask as 0.0.255.255 and then generates the commands in Juniper syntax,

```
filter 1 {
  term T1 {
    from {
      destination 172.16.0.0 0.0.255.255;
    }
    then {
      permit;
    }
    from {
      destination 172.17.0.0 0.0.255.255;
    }
    then {
      permit;
    }
  }
}
```

Example 2. All hosts within the network 192.168.50.0/23 should be denied access, except for hosts 192.168.50.128–255. All other hosts should be permitted access.

This example shows the case of interpreting rangelist. EASYACL will first extract the semantics from the natural language descriptions including (1) permit the hosts 192.168.50.128-192.168.50.255; (2) deny the hosts 192.168.50.0/23; (3) permit all others. Therefore, three commands are synthesized,

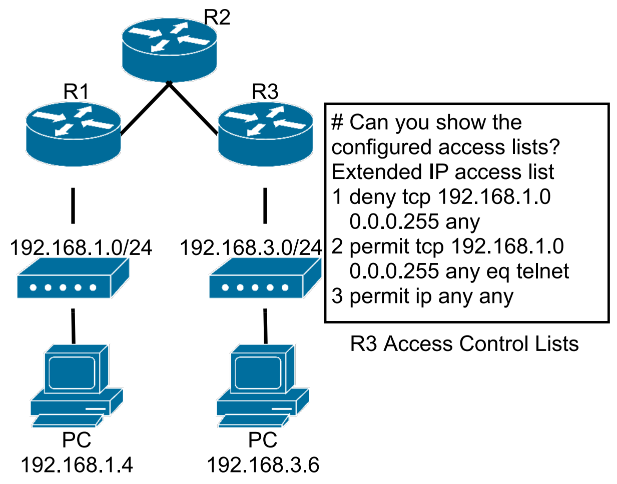
```
# command 1
type: permit
traffic: ANY
port: ANY
source: 192.168.50.128−192.168.50.255
destination: ANY
# command 2
type: permit
traffic: ANY
port: ANY
source: 192.168.50.0/23
destination: ANY
# command 3
type: permit
traffic: ANY
port: ANY
source: ANY
destination: ANY.
```

Since there is a rangelist specified by the user in the first command, our system will perform the summary calculation over it. After the calculation, it outputs the following commands in the Cisco syntax:

```
access−list 2 permit 192.168.50.128 0.0.1.127
access−list 2 deny 192.168.50.0 0.0.1.255
access−list 2 permit any.
```

5.2. Troubleshooting

Troubleshooting is another main motivation we propose EASYACL. This tool will handle natural language descriptions with no ambiguity and then make interpretations.



```
# Can you show the
configured access lists?
Extended IP access list
1 deny tcp 192.168.1.0
  0.0.0.255 any
2 permit tcp 192.168.1.0
  0.0.0.255 any eq telnet
3 permit ip any any
```

Figure 1: Troubleshooting network errors

Therefore, we can leverage this feature for troubleshooting. To be more specific, when we find any errors in the network, such as no connectivity, we can describe the requirements in natural language and re-implement an access control list quickly. Problems should be resolved if it is simply an implementation error; otherwise, it should be a design problem. In this section, we present an analysis of a practical implementation error [13].

Example 3. The configuration requires: any tcp traffic from hosts 192.168.1.0/8 should be permited through the telnet port; all ip traffic are permited.

As shown in Figure 1, there is an error in the access control list implementation: Host 192.168.1.4 has no telnet connectivity with 192.168.3.6. It is a common mistake that many network engineers may encounter, because the router processes ACLs from the top down, statement 1 denies host 192.168.1.4, so statement 20 does not get processed. To troubleshoot this problem, we asked a network administrator to try with our proposed method.

He tried with feeding our system with the natural language descriptions one sentence after the other. And EASYACL synthesized the commands:

```
1 permit tcp 192.168.1.0
  0.0.0.255 any eq telnet
2 deny tcp 192.168.1.0
  0.0.0.255 any
3 permit ip any any
```

Comparing the synthesized commands and the original implementation, statement 1 and 2 are reversed. The last line allows all other non-TCP traffic that falls under IP (ICMP, UDP, and so on). The network administrator corrected the implementation right away and claimed it a helpful tool for network troubleshooting.

6. Discussion

Although tutorial style training software for network configuration, such as Cisco Packet Tracer [14] and Graphical Network Simulator-3 (GNS3) [15], and the natural language programming of EASYACL are similar, EASYACL has some key advantages. The use of a natural language system is advantageous over training software because it conveys the meaning of concepts in plain language. Alternatively, tutorials lead the end-user to mimic patterns which provide desired results, which can be an important intermediate step for learning. However, the end-user cannot learn a programming language if they cannot assign semantics to commands. Without such an understanding end-users are less able to remember syntax or make adaptations.

In addition to ensuring that the end-user understands the meaning of tasks in a programming language, the use of natural language translation also reduces the time it takes to learn. The casual and conversational dialogue included in EASYACL allows the end-user to transition from English to programming syntax at his or her own pace. Furthermore, because EASYACL responses in natural language and program syntax, the end-user is always able to easily refer between the two, should they become confused.

Synthesizing programming language from natural language descriptions has been discussed widely. Applications are constructed by researchers for different purposes. PiE [16] is a framework that automatically generates programs from natural language descriptions using a rule-based method. Similar approaches are also presented in Natural Shell [17], both of which are targeting for tutoring purposes. While natural language translation is largely advantageous over tutorial or training software as a method of programming education, there are some drawbacks of natural translation, as well as some instances where a tutorial-model is more effective. If an end-user is not sufficiently attentive, a natural language translation and program synthesis system, such as EASYACL, has the potential to be too overly flexible, such that it prohibits learning. For example, if an end-user can use a wide range of natural language commands to synthesize a target program, and he or she is not attentive to recognize the associated program syntax, this may inhibit their learning by not directly forcing them to learn the syntax. Because tutorial software most often attempts to mimic real programming scenario, it does not typically have this issue.

7. Conclusion

In this paper, we developed a system called EASYACL in order to synthesize access control lists from natural language inputs. EASYACL has a simple system architecture in which the user provides a natural language description to Eliza, who replies with an accurate ACL rule and natural language response to maintain a conversation. EASYACL operates on a rule-based intelligent system, which is capable of receiving ranges of IP addresses, and supports multi-platform outputs, i.e., Cisco and Juniper. We demonstrated the functionalities of EASYACL through three case studies. It was shown that the code can be conveniently modified by even novice programmers to expand functionality, increase the flexibility of users' natural language inputs, or troubleshoot a problematic configuration.

References

[1] R. Shirey, “Internet Security Glossary, Version 2,” Internet Requests for Comments, RFC Editor, RFC 4949, 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4949

[2] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, “The cutting EDGE of IP router configuration,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 21–26, 2004.

[3] Y. Bhaiji, *Network Security Technologies and Solutions (CCIE Professional Development Series)*. Pearson Education, 2008.

[4] J. Davies, P. Comerford, V. Grout, N. Rvachova, and O. Korkh, “An investigation into the effect of rule complexity in access control list,” 2012. [Online]. Available: https://www.khai.edu/csp/nauchportal/Arhiv/REKS/2012/REKS512/Davies.pdf

[5] S. Gulwani, “Automating string processing in spreadsheets using input-output examples,” in *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL ’11. New York, NY, USA: ACM, 2011, pp. 317–330.

[6] C. Wang, A. Cheung, and R. Bodik, “Synthesizing highly expressive SQL queries from input-output examples,” in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2017, pp. 452–466.

[7] O. Bastani, R. Sharma, A. Aiken, and P. Liang, “Synthesizing program input grammars,” in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2017, pp. 95–110.

[8] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[9] R. Vlas and W. N. Robinson, “A rule-based natural language technique for requirements discovery and classification in open-source software development projects,” in *Proceedings of the 44th Hawaii Int’l Conf. on System Sciences (HICSS)*, 2011.

[10] A. Ranta, “A multilingual natural-language interface to regular expressions,” in *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*. Association for Computational Linguistics, 1998, pp. 79–90.

[11] J. Weizenbaum, “ELIZA—a computer program for the study of natural language communication between man and machine,” *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, Jan. 1966.

[12] C. Practice, “Standard ACLs Access Control Lists,” 2014. [Online]. Available: http://www.ccnapractice.com/acls/standard-acls

[13] Orbitco, “What is ACLs Error ? Solutions to ACLs errors Examples,” 2015. [Online]. Available: http://www.orbit-computer-solutions.com/network-troubleshooting-access-control-lists-errors/

[14] J. Janitor, F. Jakab, and K. Kniewald, “Visual learning tools for teaching/learning computer networks: Cisco networking academy and packet tracer,” in *Proceedings of the Sixth International Conference on Networking and Services (ICNS)*. IEEE, 2010, pp. 351–355.

[15] C. Welsh, *GNS3 Network Simulation Guide*. Packt Publishing, 2013.

[16] X. Liu and D. Wu, “PiE: Programming in Eliza,” in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2014, pp. 695–700.

[17] X. Liu, Y. Jiang, L. Wu, and D. Wu, “Natural Shell: An assistant for end-user scripting,” *International Journal of People-Oriented Programming (IJPOP)*, vol. 5, no. 1, pp. 1–18, 2016.