

THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

COLLEGE OF INFORMATION SCIENCES AND TECHNOLOGY

ELIZA FOR ACCESS CONTROL LISTS

BRETT ALEXANDER HOLDEN
SPRING 2015

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Security and Risk Analysis and Russian
with honors in Security and Risk Analysis

Reviewed and approved* by the following:

Dinghao Wu
Assistant Professor of Information Sciences and Technology
Thesis Supervisor and Honors Adviser

Patrick Shih
Research Associate and Lecturer of Information Sciences and Technology
Thesis Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

The network configuration languages used to create access control lists (ACLs) have a complex syntax of commands with tens of options, which make both construction and revision difficult. For this reason, ACLs have become a dark art. In light of this, we should find a comprehensible way to redesign the philosophy of ACL construction. Based on Eliza, the prototype of Artificial Intelligence and the subsequent work, Programming in Eliza (PiE), we propose a specific implementation of PiE for natural language programming of ACLs. PiE-ACL extends from PiE and PiE-LOGO to demonstrate 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. In addition to ACL syntax for Cisco, PiE-ACL also supports syntax for Juniper devices as a reference for those working with multiple platforms. Furthermore, by introducing a “range” feature, we make it possible for users to apply configurations to a range of IP addresses in an abstracted batch, rather than tediously repeating commands. Lastly, PiE-ACL has reusable program code, making it easy to improve or implement in a related system. PiE-ACL enables end-users with no prior programming experience, and can shorten the learning period for experienced programmers as well.

TABLE OF CONTENTS

LIST OF FIGURES	iii
ACKNOWLEDGEMENTS	iv
Chapter 1 Introduction	1
Chapter 2 System Architecture	9
Chapter 3 System Techniques.....	12
Chapter 4 Demonstration	16
Chapter 5 Discussion	31
Chapter 6 Conclusion.....	33
Appendix A Program Syntax and Examples	34
BIBLIOGRAPHY.....	42

LIST OF FIGURES

Figure 1 System Architecture9

ACKNOWLEDGEMENTS

I will take this opportunity to thank all of the individuals who supported and guided me throughout my thesis work. I would like to thank Dr. Dinghao Wu and Xiao Liu for introducing me to and inviting me to continue their work on the “Programming in Eliza” project. Their help was invaluable to creating my written thesis as well as the completion of a fully functional program for creating access control lists. I would also like to thank Dr. Patrick Shih for serving as my faculty reader. Lastly, I would like to thank Edward Chan for his help with debugging and at times allowing me to use him as a rubber ducky.

Chapter 1

Introduction

It is safe to say that we have entered a new era in which computer programming is on the rise. Programming skills are needed by virtually every industry including Information Technology, Business, Consulting, and Education. Even seemingly “untechnical” industries require or can benefit from some method of data analysis. Although there are only about 3 million professional programmers in the United States, the number of people who do some quantity of programming at work is over 12 million, and over 50 million people use spreadsheets, which constitutes simple programming.

While programming is a very valuable and needed skill, it can also be rather difficult to learn. The learning curve for beginning programmers is very steep. Programming comes naturally almost immediately to a very small subset of people. For a much larger set of programmers, it takes a great deal of time before they have what some call a “light bulb” moment, or rather a sudden revelation, after which they progress very quickly. For an entirely different group, programming never becomes enjoyable or natural. Another problem for even the most experienced of programmers is switching between programming languages. Although an experienced programmer may thoroughly understand the common logic between languages, it is inevitable that it will take him or her a considerable amount of time to adapt to a new syntax. From a computer science and programming education viewpoint, the goal of Programming in Eliza is to reduce the time that it takes programmers of any skill level to learn to program or master a specific language or application.

The PiE-ACL system described in this thesis is based directly on the preceding paper "Eliza - A Computer Program For the Study of Natural Language Communication Between Man And Machine" (Weizenbaum, 1966), and its revival and expansion in "Programming in Eliza" (Liu & Wu, 2014). As these papers serve as the foundation for this thesis, their contents will be briefly explained.

Eliza is a program which enables natural language conversation with a computer through the use of a rule based method. These rules include decomposition rules which are used to break down user input, and associated reassembly rules which are used to formulate an appropriate response. More specifically, the sentences that users input are analyzed according to the decompositions rules which are triggered by key words. For example, a very simple decomposition rule might exist which triggers on the word "hello". A reassembly rule could correspond to this "hello" decomposition rule which maps a response of "Hello there!" This primitive rule mapping allows the computer to understand, so to speak, that the user has said hello, and to greet him or her back to portray understanding and respond appropriately. Programming in Eliza (PiE) proposed the use of Eliza for the novel purpose of program synthesis, and applied it to synthesize programs in the LOGO programming language with successful results.

PiE recognized that natural language translation is founded on the unattainable principle of "AI Complete", which means that that a machine is required to understand every natural language description. This state has not been achieved, however it is possible for a machine to partially understand people, and thus PiE asserted and demonstrated that in a subdomain, natural language programming is viable.

PiE-ACL extends the work of PiE and PiE-LOGO to another field to further demonstrate the effectiveness of the PiE method of natural language programming in such specific subdomains (Liu & Wu, 2014; Weizenbaum, 1966).

The creation of access control lists can become quite complex and thus a unique and fitting application of PiE to a domain specific language. For example, a network engineer may

need to create an ACL rule used to block Remote Desktop Protocol traffic from the first ten hosts (192.168.1.1-192.168.1.10) on his network to a host computer (192.168.1.15) which contains sensitive information, as these first ten hosts are operated by unprivileged users. He has in mind that even if one of these unprivileged users discovers the password to the sensitive computer, they will still be unable to initiate an unwanted Remote Desktop Protocol session. To create such an ACL rule the network engineer will need actually a minimum of five rules or rather lines. These lines would be:

```
deny tcp 192.168.1.1 0.0.0.0 host 192.168.1.15 eq 3389
deny tcp 192.168.1.2 0.0.0.1 host 192.168.1.15 eq 3389
deny tcp 192.168.1.4 0.0.0.3 host 192.168.1.15 eq 3389
deny tcp 192.168.1.8 0.0.0.1 host 192.168.1.15 eq 3389
deny tcp 192.168.1.10 0.0.0.0 host 192.168.1.15 eq 3389
```

Here, in each line, “deny” indicates that the traffic will be blocked, “tcp” refers to the Transport Layer Protocol of Remote Desktop Protocol, “3389” refers to the registered port number of remote desktop protocol, and 192.168.1.15 refers to the receiving host. The following part of the input refers to the most concise summarization of the sending hosts. Cisco IOS syntax requires that this include a four-byte inverse of the subnet mask, sometimes called a “wildcard mask” of the Variable Length Subnet Mask (VLSM).

```
192.168.1.1 0.0.0.0
192.168.1.2 0.0.0.1
192.168.1.4 0.0.0.3
192.168.1.8 0.0.0.1
192.168.1.10 0.0.0.0
```


Above the first line represents the summary of 192.168.1.1, the second line represents 192.168.1.2 and 192.168.1.3, the third line represents 192.168.1.4, 192.168.1.5, 192.168.1.6, 192.168.1.7, the fourth line represents 192.168.1.8, 192.168.1.9, and the last line represents 192.168.1.10.

Consider also a larger range, 192.168.1.1-192.168.1.100. Here, the most concise summarization of the range in Cisco IOS, which once again uses wildcard masks, is the following:

```
192.168.1.1 0.0.0.0
192.168.1.2 0.0.0.1
192.168.1.4 0.0.0.3
192.168.1.8 0.0.0.7
192.168.1.16 0.0.0.15
192.168.1.32 0.0.0.31
192.168.1.64 0.0.0.31
192.168.1.96 0.0.0.3
192.168.1.100 0.0.0.0
```

These examples demonstrate that although it often requires several lines to summarize an IP address range for access control lists, as the range gets larger, the summary becomes more beneficial. In the first example of a 10 IP address range, there is a five-line summary, whereas in the second example there is a 100 IP address range, which can be summarized in nine lines.

It seems very counterintuitive that it requires several lines to summarize a range, but this stems from the fact that a range of IP addresses must be exact when dealing with access control lists. What this means conceptually, is that when an access list permits or denies traffic to a

range of addresses, it must do so to exactly those specified addresses. It is not acceptable to budget too few or too many addresses in the summary. This idea contrasts with traditional subnetting procedures, in which network engineers will choose a subnet which may contain a few additional addresses than the desired number.

For example, if I have the address space from 192.168.1.0 to 192.168.1.255, and I would like to create a subnetwork which contains 50 hosts, I would create the network “192.168.1.0 255.255.255.192”. For reference, this network in wildcard notation would be “192.168.1.0 0.0.0.63”. This network would have useable host addresses from 192.168.1.1 to 192.168.1.62. In this example, in the last octet of the address scheme, there are two bits used for the network portion of the mask and six bits used for the host portion. These six host bits provide two to the sixth power hosts, or rather 64 hosts, which converts to 62 useable hosts, as the first and last address 192.168.1.0 and 192.168.1.64, are used as the network and broadcast address respectively. Effectively, the range 192.168.1.51 -192.168.1.63 is extra addresses.

The specific reason why ACLs cannot summarize networks in such a fashion of rough estimates by powers of two is because it could be insecure or dysfunctional to permit or deny traffic to these additional addresses, 192.168.1.51-192.168.1.63.

Thus instead, ACLs uses multiple lines with wildcard masks to precisely summarize a range. In a wildcard mask “0” indicates that the byte must be matched exactly, whereas a greater number corresponds with the number of bits which may vary.

The primary reason why IP range summarizations often require several lines is because with a low starting IP range such as the second example (192.168.1.1-192.168.100), the number of hosts required consumes more bits than the starting address. This is most easily seen in the very first line “192.168.1.1 0.0.0.0”. This line must be summarized as such, because any other

value in the last octet would result in an imprecise specification. For example, “0.0.0.1” would include the address 192.168.1.0, as would “0.0.0.3”, or any larger mask. Once again, this additional address is not acceptable in the context of security.

This example can be contrasted with another range, such as 192.168.1.64-192.168.1.79. This range could be summarized in just one line as the following:

```
192.168.1.64 0.0.0.15
```

This is an example of a somewhat rare case in which the starting address and the total length of the range fit nicely into powers of two which do not overlap. Specifically, the starting address 64 is a power of two, the range is 16 addresses which is also a power of two, and 64 is greater than 16 which prevents the number of hosts required from distorting the starting address.

To put things in perspective, these examples represent a simple action that a user may want to put in ACL rules, but it requires rather complicated syntax and calculation. With a wider range of networks or protocols this action would require even more work and caution of the network engineer.

Access control lists are the first and main defense for securing a network infrastructure. At a high level, access control lists are lists of rules that a router uses to decide whether to deliver information based on its source and destination, among other attributes.

More specifically, access lists filter network traffic by controlling whether routed packets are forwarded or blocked at the router’s interfaces. Based on the rules that are provided by the end-user, a router examines each packet to determine if it should forward or drop it. Although lack of authentication allows sophisticated users to evade access control lists using data encapsulation or other forms of traffic manipulation, they are still largely effective and remain

the first line of defense for any enterprise network. Without the use of access control lists all packets passing through a router could be allowed to all parts of the network (Cisco, 2007).

Beyond their prevalence in network infrastructure, access control lists are also a fitting subdomain for the application of PiE because they have a unique programming language. In addition, this programming language varies across platforms. The majority of networks have Cisco brand routers, which use Cisco IOS (Internetwork Operating System). End-users manipulate IOS using a command line interface which consists of multiple-word commands. While IOS and similar platforms can be considered relatively simple, they are also limited and not intuitive to manipulate based on traditional programming experience. PiE-ACL is capable of generating ACL rules in IOS as well as the Junos operating system syntax for Juniper networking devices. Cross-platform support can allow developers to avoid the need to master several syntaxes for different platforms.

A final reason why access control lists are a fitting subdomain for natural language programming is that it is very easy to make an access control list error, very damaging to make such an error, and very difficult to realize such an error. PiE-ACL can help ease users' construction of ACLs to avoid such critical errors. The accidental writing of a "permit" in place of a "deny" could allow the very traffic which a network administrator was trying to block, and the reverse could halt essential traffic on a network. Furthermore if a "deny" statement was meant to apply to a protocol such as HTTPS and mistakenly specified TCP instead of UDP, the rule would have no effect, and the error would not be realized until damage was already dealt to the network. Lastly, as demonstrated above it can be very difficult and easy to make an error in network summarization for ACLs. Errors in access control lists often go unnoticed because they are often tedious or not possible to test or sanity check.

This is where PiE-ACL can come in to empower users learn to program, check for and completely avoid errors with natural language syntax, perform IP range handling, provide cross-platform support, and reusable code.

Chapter 2

System Architecture

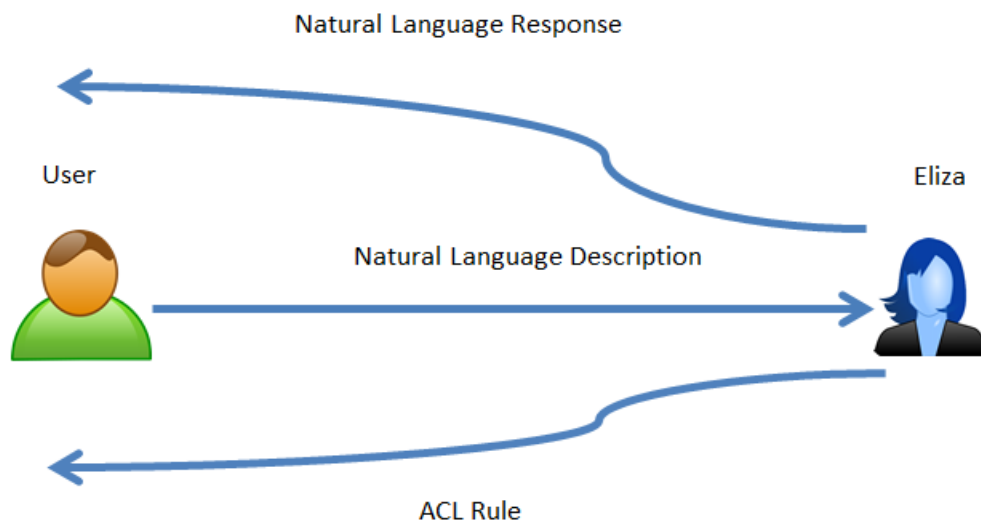


Figure 1 System Architecture

Above all else, PiE-ACL is simply a system, or rather a computer program, for creating access control lists. This computer program operates by receiving natural language input, interpreting such input, 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.

There are two entities which are responsible for communicating with one another: the system user and the Eliza engine. The system user is responsible for entering natural language which details what ACL rule the user would like to create. In practice, the user only needs to describe what network action or activity he or she would like to allow or prevent, one at a time. Each one of these desired or undesired “actions” can be converted into an ACL rule.

The system user output of natural language is then fed into Eliza as an input, whereupon Eliza produces two outputs. The first output is a natural language response to the user’s description, which reiterates their request and indicates that it is being carried out. This maintains a conversational dialogue to promote learning and maintain program usability. Furthermore, this allows the user to see that meaning of their natural language description was correctly interpreted. The second output is a syntactically and semantically correct ACL rule. This ACL rule can be outputted in the syntaxes for both Cisco and Juniper devices. The user can use this output to learn how to form their own ACL rules, or even implement it directly into a preexisting network configuration.

In fact, the Eliza engine can be further broken down into two parts which correspond with its outputs: PiE which maintains natural language conversation with the user, and ACL OS which generates rules. In this context the system actually consists of three entities: the User, PiE, and ACL OS.

It is noteworthy that the system’s flexibility through the use of pattern recognition in interpreting users’ natural language descriptions creates a many-to-one relationship, in which an infinite number of descriptions could be translated in to a single rule. In addition, there is a one-to-one relationship between the rule and natural language description that Eliza responds with. In other words, Eliza only offers one natural language description per semantically equivalent

description. Once again these responses are formed using pattern recognition and are designed to be conversational and easily understood.

Chapter 3

System Techniques

In its current state PiE-ACL consists of three python files which each perform a major function of the program. These are the “interface”, “rules”, and “rangelist” files.

The “interface” file, as its name suggests, contains the code which manages the program interface, which is Eliza. It is responsible for receiving user input and delivering the appropriate output. This file is also responsible for writing the “right.txt” and “wrong.txt” files which keep track of correct and uninterpretable end-user input. This file also sets up the regular expression mapping scheme which is used by “rules”. Lastly, “interface” catches a variety of errors in order to allow for and maintain a stream of natural language conversation between the end user and Eliza.

While the “interface” file contains the backbone of the program that is Eliza, a conversational console, the “rules” file governs how Eliza interprets such a conversation. “Rules” contains the main response table in which each element has three sub-elements: a regular expression to recognize user input, the associated IOS syntax command output, and a natural language output. The purpose of the regular expression is to recognize the desired action of the end-user, and to define the parts of this input which need to be used in order to comprise the output.

For example, suppose the end-user input is “Hello, I would like to deny the IP 192.168.1.13”. The associated regular expression needs to recognize that the words “deny” and “192.168.1.13” in order to form the IOS syntax command “deny ip 192.168.1.13 255.255.255.0”. Furthermore, these keywords are needed in order to form the natural language output “Okay, let’s block all traffic from 192.168.1.13 255.255.255.0!”.

The natural language output is delivered first so that the end-user understands that Eliza has appropriately interpreted their input. This is followed by the correctly formatted IOS syntax command for the natural language description. In simple terms, this is the exact command which the end-user would need to enter into their router's terminal in order to achieve their desired action.

A crucial feature of PiE-ACL is its ability to handle ranges of IP addresses, by calculating network summarizations and using them in ACL rule generation. Depending size of the network range and its specific summarization, this often results in output of several lines of rules.

The capability of PiE-ACL to handle ranges of IP addresses in its interpretation of natural language is extremely beneficial to the user. Calculating and appropriately formatting network summarization to represent a range of addresses is a tedious and difficult process. This feature empowers end-users who have limited computer networking knowledge and would otherwise be thwarted by the need to create summaries. In fact, many users commonly complain that router interfaces do not support network ranges. A common computer programmer likely would not know off-hand how to perform network summary. Likewise, even someone well-versed in networking may seldom use this skill.

The range handling of PiE-ACL is contained within the "rangelist" file. 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 "192.168.1.1 – 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”.

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 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 “rangelist” file accomplishes the necessary precision and conciseness using Variable-Length Subnet Masking (VLSM). The result in the above example would be the summarization of “192.168.1.1 – 192.168.1.10” as “192.168.1.1/32, 192.168.1.2/31, 192.168.1.4/30, 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, the range file 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 final result of the example is then “192.168.1.1 0.0.0.0, 192.168.1.2 0.0.0.1, 192.168.1.4 0.0.0.3, 192.168.1.8 0.0.0.1, 192.168.1.10 0.0.0.0”.

The result of the “rangelist” file is now an IP address range in a usable format for ACL synthesis. This component can be used to implement many preexisting actions defined in “rules” commands across a desired range of IP addresses.

A final feature which sets distinguishes PiE-ACL, in addition to the handling of IP address ranges, is the generation of cross-platform output; this being both the Cisco IOS and Juniper Junos syntaxes. 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, or to adapt their existing ACL rules.

Chapter 4

Demonstration

This section contains demonstrations which show a number of strengths of the PiE-ACL system for the synthesis of access control lists. The first key strength of the system is its flexibility and robustness in interpreting the natural language descriptions provided by users. The system can correctly interpret a wide range of semantically equivalent natural language descriptions into the same ACL rule. As previously mentioned, the system also provides a natural language response to the user in order to maintain a conversation.

The following examples (I and II) are used to demonstrate that the system is capable of accurately interpreting the meaning of common natural language descriptions. In each example there are several possible human inputs which are describing the same desired action. The inputs in are interpreted the same, and result in an equivalent ACL rule and natural language response. The examples contain several typical inputs by a human user, many of which are extremely conversational and even contain grammatical errors.

Example I – User Input Flexibility

User Natural Language Descriptions:

>Eliza, I would like to permit HTTPS protocol from ip 172.21.1.1 to ip 172.21.1.15

>Can you please permit https traffic from 172.21.1.1 to 172.21.1.15

>Hmmm, do me a favor and permit Https from ip 172.21.1.1 to ip 172.21.1.15

>I can't figure out how to permit https traffic from 172.21.1.1 to 172.21.1.15 ! Can you help me Eliza?

>What rule can I use? I would like to permit https traffic from ip 172.21.1.1 to ip 172.21.1.15
what is the proper command?

>Eliza, I would like to make a rule but I cannot remember the port number for the HTTPS protocol! Please help me permit https traffic from 172.21.1.1 to 172.21.1.15

>hi computer program. i need to permit https from 172.21.1.1 to 172.21.1.15 okay?

>Hi Eliza, it is to meet you. I need you to do me a favor : permit https traffic from 172.21.1.1 to 172.21.1.15 . How does that sound?

Eliza's Natural Language Response:

Okay, let's permit all HTTPS traffic from 172.21.1.1 to 172.21.1.15

Eliza's ACL Rule:

```
permit tcp host 172.21.1.1 host 172.21.1.15 eq 443
```

Example II – User Input Flexibility Cont'd

User Natural Language Descriptions:

- > I need to avoid the risks of remote access on the network. Eliza, please deny all rdp traffic to or from any host on our network.
- > DENY ALL RDP NOW!
- > There may be a security breach. Eliza please deny each remote desktop protocol connection on my network.
- > Eliza, I forgot the port number but I need to deny any rdp connections.
- > Please deny every remote desktop session
- > \.\\I need to deny all remote desktop protocol conectionss. Can you help me?
- > I can't remember whether it uses tcp or udp. Eliza, how can I deny all rdp traffic?

Eliza's Natural Language Response:

Okay, let's deny all RDP traffic!

Eliza's ACL Rule:

deny tcp any any eq 3389

Another very powerful characteristic, shown in example III below, of the PiE-ACL system is its high level of abstraction. This is most apparent in the system's ability to handle ranges of IP addresses in users' natural language descriptions. This feature is extremely useful for novice and experienced users because network summarization for access control lists can be both difficult and very tedious. In each case, the user is able to focus on learning ACL syntax. The examples below demonstrate the high level abstraction PiE-ACL is capable of.

Example III – Range Commands

> Eliza, I need to permit tcp from ip in range 192.168.1.1-192.168.1.10 to ip 192.168.1.15

Okay, let's permit all TCP traffic from range 192.168.1.1-192.168.1.10 to 192.168.1.15!

```
permit tcp 192.168.1.1 0.0.0.0 host 192.168.1.15
```

```
permit tcp 192.168.1.2 0.0.0.1 host 192.168.1.15
```

```
permit tcp 192.168.1.4 0.0.0.3 host 192.168.1.15
```

```
permit tcp 192.168.1.8 0.0.0.1 host 192.168.1.15
```

```
permit tcp 192.168.1.10 0.0.0.0 host 192.168.1.15
```

>Sorry Eliza, I actually made a mistake I would like to DENY tcp from ip in range 192.168.1.1-192.168.1.10 to ip 192.168.1.15

Okay, let's deny all TCP traffic from range 192.168.1.1-192.168.1.10 to 192.168.1.15!

```
deny tcp 192.168.1.1 0.0.0.0 host 192.168.1.15
```

```
deny tcp 192.168.1.2 0.0.0.1 host 192.168.1.15
```

```
deny tcp 192.168.1.4 0.0.0.3 host 192.168.1.15
```

```
deny tcp 192.168.1.8 0.0.0.1 host 192.168.1.15
```

```
deny tcp 192.168.1.10 0.0.0.0 host 192.168.1.15
```

>Please also give me the command to deny udp from ip in range 192.168.1.1-192.168.1.10 to ip 192.168.1.15

Okay, let's deny all UDP traffic from range 192.168.1.1-192.168.1.10 to 192.168.1.15!

```
deny udp 192.168.1.1 0.0.0.0 host 192.168.1.15
```

```
deny udp 192.168.1.2 0.0.0.1 host 192.168.1.15
```

```
deny udp 192.168.1.4 0.0.0.3 host 192.168.1.15
```

```
deny udp 192.168.1.8 0.0.0.1 host 192.168.1.15
```

```
deny udp 192.168.1.10 0.0.0.0 host 192.168.1.15
```

>However, I do need to add remote access to this gateway for some hosts. Let's permit ssh from ip in range 192.168.1.1-192.168.1.10 to ip 192.168.1.15

Okay, let's permit all SSH traffic from range 192.168.1.1-192.168.1.10 to 192.168.1.15!

```
permit tcp 192.168.1.1 0.0.0.0 host 192.168.1.15 eq 22
```

```
permit tcp 192.168.1.2 0.0.0.1 host 192.168.1.15 eq 22
```

```
permit tcp 192.168.1.4 0.0.0.3 host 192.168.1.15 eq 22
```

```
permit tcp 192.168.1.8 0.0.0.1 host 192.168.1.15 eq 22
```

```
permit tcp 192.168.1.10 0.0.0.0 host 192.168.1.15 eq 22
```

>Hmmm, why don't I do a different set. Let's permit rdp from ip in range 192.168.1.1-192.168.1.100 to ip 192.168.1.115

Okay, let's permit all RDP traffic from range 192.168.1.1-192.168.1.100 to 192.168.1.115!

```
permit tcp 192.168.1.1 0.0.0.0 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.2 0.0.0.1 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.4 0.0.0.3 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.8 0.0.0.7 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.16 0.0.0.15 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.32 0.0.0.31 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.64 0.0.0.31 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.96 0.0.0.3 host 192.168.1.115 eq 3389
```

```
permit tcp 192.168.1.100 0.0.0.0 host 192.168.1.115 eq 3389
```

>Again, why don't I do a different set. Let's permit udp from ip in range 192.168.1.64-192.168.1.79 to ip 192.168.1.99

Okay, let's permit all UDP traffic from range 192.168.1.64-192.168.1.79 to 192.168.1.99
permit udp 192.168.1.64 0.0.0.15 host 192.168.1.99

Another major strength of our system is that its structure is very convenient for reuse and revision. The "rules" file in particular, can be easily manipulated by a novice programmer. In order to add a new feature, for instance, adding support for another protocol, one simply needs to create a new rule which follows the same structure of the existing rules. The program code could also be easily revised to accept even more natural language descriptions by making some minor modifications the regular expressions in the existing rules, or even more easily by creating new rules to match the descriptions.

Lastly, PiE-ACL has yet another feature which is extremely convenient for users: it is capable of producing output for multiple platforms. Currently the system supports output in for Cisco IOS and Junos OS for Juniper devices. Example IV below shows several interactions where Eliza converts the user's natural language description into both IOS and Junos syntax for handy reference.

Example IV – Cross-Platform Support

>What rule can I use? I would like to permit https traffic from ip 172.21.1.1 to ip 172.21.1.15

what is the proper command?

Okay, let's permit all HTTPS traffic from 172.21.1.1 to 172.21.1.15

```
permit tcp host 172.21.1.1 host 172.21.1.15 eq 443
```

```
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;  
    }  
  }  
}
```

> There may be a security breach. Eliza please deny each remote desktop protocol connection on my network.

Okay, let's deny all RDP traffic!

deny tcp any any eq 3389

```
filter 1 {  
  term T1 {  
    from {  
      protocol tcp;  
      destination-port 3389;  
    }  
    then {  
      reject;  
    }  
  }  
}
```

> Eliza, I need to permit tcp from ip in range 192.168.1.1-192.168.1.10 to ip 192.168.1.15

Okay, let's permit all TCP traffic from range 192.168.1.1-192.168.1.10 to 192.168.1.15!

```
permit tcp 192.168.1.1 0.0.0.0 host 192.168.1.15
```

```
permit tcp 192.168.1.2 0.0.0.1 host 192.168.1.15
```

```
permit tcp 192.168.1.4 0.0.0.3 host 192.168.1.15
```

```
permit tcp 192.168.1.8 0.0.0.1 host 192.168.1.15
```

```
permit tcp 192.168.1.10 0.0.0.0 host 192.168.1.15
```

```
filter 1 {
```

```
  term T1 {
```

```
    from {
```

```
      source-address {
```

```
        192.168.1.1/32;
```

```
      }
```

```
      destination-address {
```

```
        192.168.1.15/32;
```

```
      }
```

```
      protocol tcp;
```

```
    }
```

```
  then {
```

```
    accept;
```

```
  }
```

```
}
```

```
}
```

```
filter 1 {
```

```
  term T1 {
```

```
    from {
```

```
      source-address {
```

```
        192.168.1.2/31;
```

```
      }
```

```
      destination-address {
```

```
        192.168.1.15/32;
```

```
      }
```

```
      protocol tcp;
```

```
    }
```

```
  then {
```

```
    accept;
```

```
  }
```

```
}
```



```
filter 1 {  
  term T1 {  
    from {  
      source-address {  
        192.168.1.4/30;  
      }  
      destination-address {  
        192.168.1.15/32;  
      }  
      protocol tcp;  
    }  
    then {  
      accept;  
    }  
  }  
}
```

```
filter 1 {  
  term T1 {  
    from {  
      source-address {  
        192.168.1.8/31;  
      }  
      destination-address {  
        192.168.1.15/32;  
      }  
      protocol tcp;  
    }  
    then {  
      accept;  
    }  
  }  
}
```

```
filter 1 {  
  term T1 {  
    from {  
      source-address {  
        192.168.1.10/32;  
      }  
      destination-address {  
        192.168.1.15/32;  
      }  
      protocol tcp;  
    }  
    then {  
      accept;  
    }  
  }  
}
```

Chapter 5

Discussion

Training Software vs. Natural Language Programming

Although tutorial style training software for network configuration, such as Cisco Packet Tracer and Graphical Network Simulator-3 (GNS3), and the natural language programming of PiE-ACL are similar, PiE-ACL 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, the use of a tutorial at times leads the end-user to mimic patterns which provide desired results. Pattern recognition can be an important intermediate step for learning, however ultimately the end-user cannot learn a programming language if they are unable to assign meaning to commands. Without such an understanding end-users are less able to remember syntax or use it novel ways.

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 PiE-ACL allows the end-user to transition from English to programming syntax at his or her own pace. Furthermore, because the PiE-ACL responses always contain the natural language command and program syntax, the end-user is always able to easily refer between the two, should they become confused.

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 PiE-ACL, has the potential to be overly flexible, such that it prohibits learning. For example, if an end-user is able to use a wide range of natural language commands to synthesize a desired 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.

Perhaps the most fitting solution to this problem is the use of a system which at first uses natural language translation to gently introduce end-users to programming syntax, and then later acts as a tutorial as end-users progress to ensure attentiveness.

Chapter 6

Conclusion

In this thesis, we developed a system called Programming in Eliza for Access Control Lists (PiE-ACL) in order to synthesize access control lists from natural language input. PiE-ACL 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 conversation. PiE-ACL's engine operates on a rule based system, is capable of receiving ranges of IP addresses, and supports multi-platform output. Furthermore, the system's code can be conveniently modified by even novice programmers to expand functionality, increase the flexibility of users' natural language input, or implement in a new system. In summary, the PiE-ACL system has demonstrated the effectiveness of Programming in Eliza (PiE) to a domain specific language for an improved method of computer science education.

Appendix A

Program Syntax and Examples

This section gives the syntax for performing all of the functions of the PiE-ACL. Examples of end-user input and the associated program response are also given. Many functions include a minimalistic as well as a more natural or conversational input in order to demonstrate the flexibility of the program. The simplified and conversational outputs result in the same program response.

The shown syntax and examples are not exhaustive. The purpose here is to provide a general idea of how the program works. The exact specification of the accepted syntax is defined by the regular expressions in the “Rules” file, which can be found in Appendix C.

Instructions

Description:

This command will provide instructions to help get started with ACL configuration.

Syntax:

(create|make|start|begin|design|do) IP ACL

Example:

“create an IP ACL”

“I would like to make an ACL!”

Response:

“Okay, let’s create an IP ACL, you have to 1) enter privileged exec mode, 2) enter configuration mode, 3) create the IP ACL and enter IP ACL configuration mode, 4) Create a rule in the IP ACL”

Privileged EXEC Mode

Description:

To configure an ACL, first enter Privileged EXEC mode

Syntax:

(enable|privileged|exec)

Example:

“enable”

“Let’s go ahead and enter privileged exec mode.”

Response:

“Okay, let’s enter privileged EXEC mode!”

“enable”

Configuration Mode

Description:

Next, enter configuration mode.

Syntax:

(enter|start|do) confi(g|ure|guration) mode

Example:

“enter configuration mode”

“Please enter config mode.”

Response:

“configure terminal”

IP Access List

Description:

Next, specify the IP Access List name to be configured.

Syntax:

(ACL| ip access list) [list name]

Example:

“ACL 1”

“Let’s configure access-list 1”

Response:

“ip access-list 1”

Permit or Deny IPv4 Address

Description:

This command is used to permit or deny all traffic for a specified address. The command may specify or omit the subnet mask; if the subnet omitted the default subnet of the Class A, B, or C address will be populated in the system's response.

Syntax:

(permit|deny) the IP [IP Address] subnet [Subnet Mask]

Examples:

“I would like to permit the IP address 192.168.1.2 subnet 255.255.255.0”

“Please permit 192.168.1.2”

“deny the IP 128.1.0.2”

“Let's deny the IP address 128.1.0.2”

Response:

“permit ip 192.168.1.2 255.255.255.0”

“deny ip 128.1.0.2 255.255.0.0”

Permit or Deny Traffic Between Two Hosts by Protocol

Description:

This command is used to permit or deny traffic of a specified protocol between two hosts. These protocols include ICMP, TCP, UDP, HTTP, DNS, HTTPS, SSH, and RDP.

Syntax:

```
(permit|deny) [protocol] from IP [IP Address] to IP [IP Address]
```

Examples:

```
“permit udp from IP 192.168.1.2 to IP 192.168.1.3”
```

```
“Let’s go ahead and permit udp traffic from 192.168.1.2 to 192.168.1.3”
```

```
“deny https from IP 128.1.0.3 to IP 128.1.0.2”
```

```
“Please deny https from 128.1.0.3 to 128.1.0.2”
```

Response:

```
“permit udp host 192.168.1.2 host 192.168.1.3”
```

```
“deny tcp host 128.1.0.3 host 128.1.0.2 eq 443”
```

Permit or Deny All Traffic by Protocol

Description:

This command is used to permit or deny all traffic of a specific protocol.

Syntax:

```
(permit|deny) all [protocol]
```

Examples:

```
“permit all ssh”
```

```
“Hello system, please permit all ssh now!”
```

```
“deny all rdp”
```

```
“I need to deny all rdp traffic.”
```

Response:

```
“permit tcp any any eq 22”
```

```
“deny tcp any any eq 3389”
```

Permit or Deny All Traffic by IP Range

Description:

This command is used to permit or deny all traffic for a range of IP addresses. This function is extremely powerful as the end-user is not required to calculate the CIDR (Classless Inter-Domain Routing), subnet mask, or its associated inverse. This function is also very powerful because it provides the most precise and efficient number of ACL rules possible. This efficiency can become important for network performance and in large networks because ACL size is limited.

Syntax:

(permit/deny) IP in range [Network Number] [Subnet Mask]

Examples:

“permit IPs in range 192.168.1.1 – 192.168.1.10”

“I would like to permit all the traffic from range 192.168.1.1 to 192.168.1.10”

Response:

“

```
permit 192.168.1.1 0.0.0.0
```

```
permit 192.168.1.2 0.0.0.1
```

```
permit 192.168.1.4 0.0.0.3
```

```
permit 192.168.1.8 0.0.0.1
```

```
permit 192.168.1.10 0.0.0.0
```

“

BIBLIOGRAPHY

- Cisco. (2007, December 27). *Configuring IP Access Lists*. Retrieved March 8, 2015, from Cisco:
<http://www.cisco.com/c/en/us/support/docs/security/ios-firewall/23602-confaccesslists.html>
- Fuller, V., Li, T., Yu, J., & Varadham, K. (1993, September). *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*. Retrieved March 1, 2015, from RFC Editor: <http://www.rfc-editor.org/info/rfc1519>
- Fuller, V., Li, T., Yu, J., & Varadhan, K. (1992, June). *Supernetting: an Address Assignment and Aggregation Strategy*. Retrieved March 1, 2015, from RFC Editor: <http://www.rfc-editor.org/info/rfc1338>
- Liu, X., & Wu, D. (2014). PiE: Programming in Eliza. *Proceedings of the 29th IEEE.ACM International Conference on Automated Software Engineering (ASE 2014)* (pp. 695-700). Vasteras, Sweden: New Ideas Papers.
- Mogul, J., & Postel, J. (1985, August). *Internet Standard Subnetting Procedure*. Retrieved February 26, 2015, from RFC Editor: <http://www.rfc-editor.org/info/rfc950>
- Moss, D. P. (2014, December 12). *netaddr 0.7.14*. Retrieved February 20, 2015, from Python: <https://pypi.python.org/pypi/netaddr>
- Postel, J. (1981, September). *Internet Protocol*. Retrieved February 25, 2015, from RFC Editor: <http://www.rfc-editor.org/info/rfc791>
- Pummill, T., & Manning, B. (1995, October). *Variable Length Subnet Table For IPv4*. Retrieved March 1, 2015, from RFC Editor: <http://www.rfc-editor.org/info/rfc1860>
- Tsuchiya, P. (1991, April). *On the assignment of subnet numbers*. Retrieved February 22, 2015, from RFC Editor: <http://www.rfc-editor.org/info/rfc1219>

Weizenbaum, J. (1966, January). Computational Linguistics. (A. Oettinger, Ed.)

Communications of the ACM, 9(1), 36-45.

ACADEMIC VITA

Brett A. Holden

1768 Cambridge Drive, State College, PA 16803

(814) - 574 -7432 | bah5423@psu.edu

Education

The Pennsylvania State University – The Schreyer Honors College
University Park, PA

B.S. Security and Risk Analysis (with honors)

May 2015

Concentration: Information & Cyber Security

B.A. Russian

Awards: Phi Beta Kappa Honor Society, PNC Technologies Scholarship 2014-15, Garner Rothrock Memorial Fund 2014-15, PNC Technologies Scholarship 2013-14, Northrop Grumman Corporation Scholarship 2012 -13, Dean's List, The National Society of Collegiate Scholars

Activities: Web Coordinator: Penn State Association of Students in Russian, IM Flag Football, Student Hockey League, Penn State Student Russian Society, Information Assurance Club, PSU Car Club

CIEE Study Abroad Summer 2013: Saint Petersburg State University

- Russian Grammar, Advanced II
- Russian Phonetics, Advanced II
- Russian Conversation, Advanced II
- Russian Culture, Media, and Music

Relevant Courses

- Introduction to Application Programming (IST 297D)
- Overview of Information Security (SRA 221)
- Networking and Telecommunications (IST 220)
- Organization of Data (IST 210)
- Statistical Analysis for Information Sciences (SRA 497A)
- Computer and Cyber Forensics (IST 454H)
- Network Security (IST 451)
- Information Security Management (IST 456)
- Research Methods for the Information Sciences and Technology (IST 489H)

- Risk Management: Assessment and Mitigation (SRA 311)
- Language, Logic, and Discrete Mathematics (IST 230)
- Calculus with Analytic Geometry I (MATH 140)
- Calculus with Analytic Geometry II (MATH 141) (Pending Summer 2015)

Research Experience

Honors Thesis: Natural Language Translation and Program Synthesis for Access Control Lists College of Information Sciences and Technology, Adviser: Dr. Dinghao Wu

- Based on parent project, Programming in Eliza (PiE), which proposes a new method for natural language processing to include program synthesis, and applies it for computer science education use
- Research applies PiE to automatically synthesize access control lists (ACL) to Cisco IOS syntax from natural language descriptions
- Development of program interface and rules using the Python Programming Language
- Syntax documentation and user testing
- Formal thesis drafting

Employment Experience

KPMG – Information Protection and Business Resilience Intern

June 2014 – August 2014

Philadelphia, PA

- Performed vulnerability assessments and penetration tests
- Assisted in the vulnerability assessment/penetration testing lifecycle, including vulnerability research and detailed drafting of security assessment reports
- Managed data loss prevention tool for inventory, query, and analysis of sensitive data
- Assisted in interviews of clients' IT leaders to gather information for a HIPAA Risk Assessment
- Assisted in drafting of HIPAA IT Security Toolkit

Penn State ITS Service Desk Supervisor

May 2012 – February 2015

University Park, PA

- Supervision
 - Open/close desk, inventory users' and desk's property
 - Monitor and assist 2-5 consultants for smooth operation of desk
 - Perform advanced troubleshooting and hardware procedures
 - Provide feedback to Manager and Software Engineer to improve procedures and policy.
- Hiring Committee
 - Review applications and resumes of potential employees
 - Collaborate with committee members to identify most qualified applicants
 - Conduct interviews with qualified applicants
 - Provide feedback to management for hiring decisions
- Security Operations and Services Referrals
 - Rebuild user's operating system, install/configure antivirus, educate user about safe computing
 - Notarize and inform Security Operations and Services of rebuild
- ITS Accounts Support

- Initiate account setup, guide user in creating strong password within requirements
 - Verify account status, verify account password, communicate with ITS Accounts Services
- General Support
 - Hardware installation/diagnostics, software installation/troubleshooting
 - Virus detection/removal, data transfer/backup

Skills and Certifications

- External and internal penetration testing
- Vulnerability and exploit research and development
- Man-in-the-Middle and wireless exploitation
- Java, SQL, BASH, Python, Ruby, HTML, PHP
- Virtualization and server management
- Linux, Windows, and Macintosh operating systems
- Inter-autonomous and autonomous computer networking
- CompTIA A+
- Advanced Russian speech, reading, and writing.