Using DFA to enforce an Access Control List

Joseph Schmidt

May 8, 2017

1 Abstract

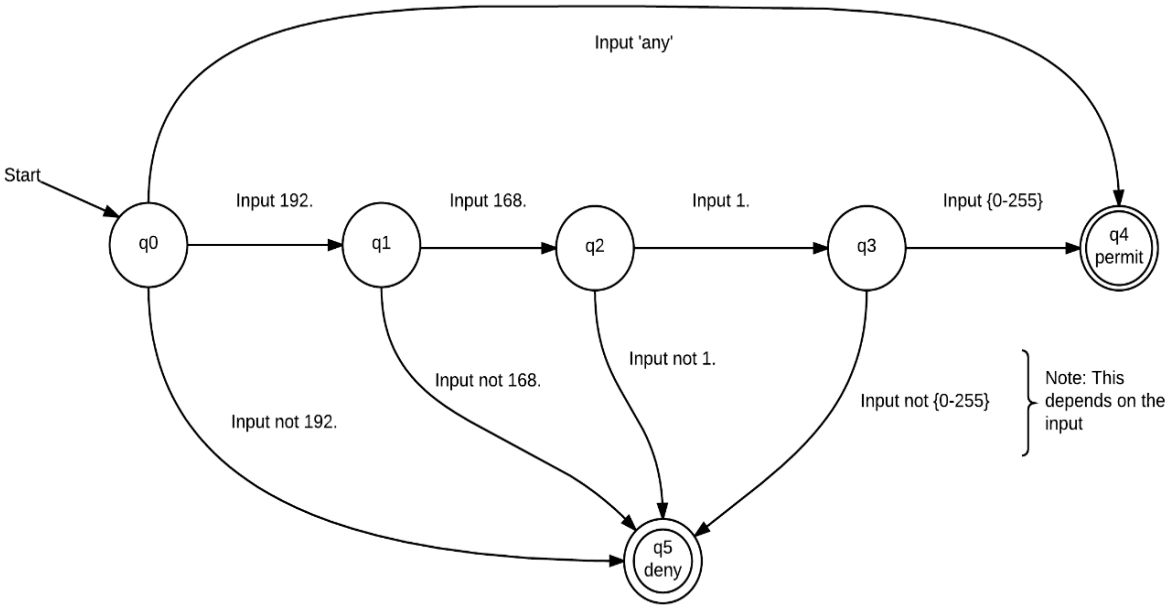
Deterministic Finite Automata can be used to validate data from a source. In cyber security, this can be used to validate and accept certain data sources on a network based on an access control list. A state machine can be created from a specified access control list to enforce the validation of data as it comes to a device. This project shows how this can be done, applying the theories of formal languages to the growing field of cyber security.

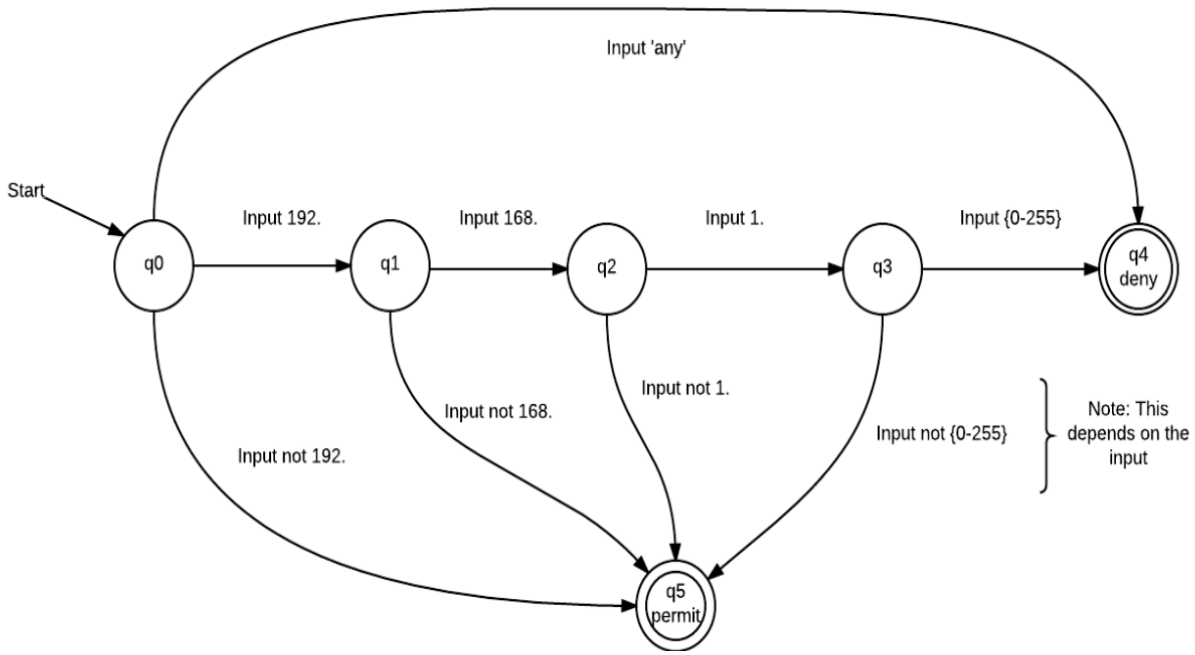
2 Introduction

In computer science, Distinct Finite Automata or DFA are state machines used to validate whether a string of symbols is accepted by a language. This concept can be applied to many facets of computer science, including cyber security, in which it is possible to use a DFA to enforce an Access Control List (ACL) on a networked device which restricts network traffic from potential threats. This project creates DFA dynamically based on a given ACL in order to restrict traffic on a device based on the source IP addresses of IP packets from a sample network stream containing several IP addresses. It works with addresses adhering to the the standard four octet format of X.X.X.X for IPv4, where X is any number between 0-255. While the project will work with all IP addresses in this format, the scope of this project is focused primarily on the 192.168.1.X local addressing scheme. The following will include the specific dectails of the project, how it is used, it's limits, and what can be done in the future with the project.

3 Detailed System Description

This project adheres to the DFA diagrams below. These diagrams show the processing of input after either a `deny` or `permit` statement, taken from an ACL. This reflects how the DFA will be created as statements are processed from an ACL; the string following the `deny` or `permit` statement, which will either be an IP address or the word `any`, is what the DFA will be matching on. For each of the `deny` or `permit` statements, there will simply be two DFA created based on a given ACL: one for the `deny` statements and





one for the `permit` statements. This project denys by default so there is no delay in the network communication which means the `deny` DFA will process commands after the `permit` DFA. This allows the permitted packets to be accepted by the device when

implemented on a device's network card.

The second diagram shown is for the `deny` DFA, where the only real difference is that state q4 is `deny` and q5 is `permit`. In processing ACLs, which is the first step of the application when the `main.py` file (the driver for the application) is ran, these DFA are the two that are created. Again, this project works with all IPv4 addresses but since the scope is limited to the 192.168.1.X scheme, that is the addressing reflected in these diagrams. Lastly, as shown above, if the `permit any` statement is declared, the DFA will simply accept any address that is checked against it. The actual enforcement of this happens with the transition method in the `transition.py` file and this behavior should be expected when the `deny any` statement is declared.

4 Requirements

Any system wanting to use this project should have the following:

- Have at least 2GB of RAM and 10GB of memory on your system.
- Have Python 3.5.0 installed on the system.
- Ensure all required libraries found in the `requirements.txt` file of project are downloaded and installed (though this should be done automatically with the `Makefile` for the project).

5 Literature Survey

The author was inspired by the Cisco IOS software and its implementation of access control lists [1]. This work is similar in that it can take an ACL and enforce it, however this project does so with DFA using a graph data structure, as well as provide the user with information such as what IP packets are being permitted and denied based on their source IP address.

6 User Manual

Please see `rules.txt` in the project for the rules this program follows regarding ACL processing, as well as other details.

To start, the user provides two arguments in order to properly run the `main.py` in a command terminal. The first is the name of the ACL wishing to be enforced which must be in the `/acl` folder and be a `.txt` file. The second is the stream of sample network source IP addresses that the user wishes to enforce their ACL on. Similar to the ACL specified, the user provides the name of a stream file that is a `.txt` and in the `/streams` folder. ACLs and sample streams can be added to either of these folders for continued development and testing. An error will be thrown, and the program will not complete, if either of these arguments are null or are not present in either

the `/acl` or `/streams` folders. After the ACL is validated, the DFA for permitted and denied addresses are created with the Node class in `node.py`, and the `dfafromacl.py` file. Actual enforcement of the created DFA happens with the transition method in the `transition.py` file. When ran, the program has a verbose output to show the processes of validating the ACL, constructing the DFA, and enforcing the DFA on the specified stream, providing two lists of permitted and denied addresses. Please see below for an example output showing the verbose process of validating the ACL and creating the DFA based on the arguments provided.

```
joseph@joseph-VirtualBox:~/GitHub/cmpt440_Schmidt/prj$ python main.py acl0 stream0
['permit 192.168.1.5', 'deny any']
Any repeat statements were removed.
['192.168.1.5']
1
['any']
('Looking at octets---', ['192', '168', '1', '5'])
('Looking at octet', '192', 0)
Node None (False)
Length is 0
('stateptr children', [])
('newNode children', [])
('newNode children', [])
('stateptr children', [Node 192 (False)])
('stateptr children', [])
('newNode children', [])
('Looking at octet', '168', 1)
Node 192 (False)
Length is 0
('stateptr children', [])
('newNode children', [])
('newNode children', [])
('stateptr children', [Node 168 (False)])
('stateptr children', [])
('newNode children', [])
('Looking at octet', '1', 2)
Node 168 (False)
Length is 0
('stateptr children', [])
('newNode children', [])
('newNode children', [])
('stateptr children', [Node 1 (False)])
('stateptr children', [])
('newNode children', [])
('Looking at octet', '5', 3)
Node 1 (False)
('Accepted state created', Node 5 (True))
DFA created for permit statements.
192
-168
--1
---5
```

The second figure below shows the lists created by the program to show which addresses were pemitted and denied. Note how the deny DFA does not have a tree print out like there was for the permit DFA. This is because in this example, the ACL provided has the `deny any` statement, meaning there is only one state in the deny DFA

which accepts any address. Furthermore, this is why the address 192.168.1.5 shows up in both output lists. However since the permit addresses are outputted/processed first, the address would still be accepted by the device running this program.

```
('Looking at octets---', ['192', '168', '100', '100'])
('Looking at octet', '192', 0)
Node None (False)
('Looking at node ', '192', ' for processing')
False
('Looking at octet', '168', 1)
Node 192 (False)
('Looking at node ', '168', ' for processing')
False
('Looking at octet', '100', 2)
Node 168 (False)
('Looking at node ', '1', ' for processing')
('Looking at octet', '100', 3)
Node 168 (False)
('Looking at node ', '1', ' for processing')
('Permitted addresses:', ['192.168.1.5'])
DFA created for deny statements.
('Denied addresses:', ['192.168.1.5', '192.168.1.6', '192.168.1.7', '192.168.1.8', '192.16
.90', '192.168.1.34', '192.168.1.123', '192.168.1.43', '192.168.1.255', '192.168.1.254', '
joseph@joseph-VirtualBox:~/GitHub/cmpt440_Schmidt/prj$
```

7 Future Improvements

For future versions of this program, the following will be considered for a more comprehensive design:

- Add another argument for enabling or disabling verbose mode.
- Use the DFA on live traffic from the network.
- Allow for wildcards.
- Work with specific network protocols (i.e. tcp, udp, icmp).

8 Conclusion

Overall this project shows how DFA can be used in cyber security, and how it can be used to protect devices on a network from possible threats. This idea can be further extended to have other features similar to Cisco IOS ACL functionality. Furthermore, this shows how effective state machines can be for validating data so it does not harm the device receiving a transmission. Cyber security is a rapidly growing field, and the author gained significant knowledge on how a network or a device can be protected using current methods of protection like an ACL. If there are any futher inqueries, please contact the author via email at Joseph.Schmidt2@marist.edu.

References

[1] Cisco IOS. Computer Software. Latest Version. Cisco. Accessed April 3, 2017. http://www.cisco.com/c/en/us/support/ios-nx-os-software/ios-software-release-15-5-3-m/model.html.