Illumio Coding Assignment 2017-2018, PCE team (Avenger)

Introduction

Thanks for your interest in Illumio! One component of our interview process is a "take-home" coding assignment. We believe that coding questions are best done in an environment you're comfortable with and where the problems are less contrived and give you greater room to decide how the solution should be implemented.

This assignment should take about one hour. Please set aside a single block of time, and don't spend more than 90 minutes on it. If you do not finish within that time, send in what you have, and let us know what you would have worked on if you had more time.

Feel free to use all the resources available to you – books, the Internet, etc. All we ask is that you do not consult with anyone else, and any code or inspiration that you may receive from an outside source (e.g. StackOverflow) is properly attributed in your code. You may use any language of your choice and the standard libraries that come with the language.

If you are selected for an onsite interview, one of the interviews will be focused on this coding assignment. We'll ask you to explain any design decisions and tradeoffs that you might have made. We'll also ask you about how you might approach and change your code to handle a few variants of the problem.

Lastly, if you have any questions, please do not hesitate to reach out to us!

Background

A core component of Illumio's product is a host-based firewall. As a simplified model, consider a firewall to be a system which is programmed with a set of predetermined security rules. As network traffic enters and leaves the machine, the firewall rules determine whether the traffic should be allowed or blocked.

Real-world firewalls support both "allow" and "block" rules, and their ordering is important in determining the fate of a packet. In this coding exercise, we will greatly simplify this model by only supporting "allow" rules. If a packet does not match any "allow" rule, then we assume it will be blocked.

Input

The provided input will be a CSV file in which each line contains exactly four columns: direction, protocol, ports, and IP address:

direction	Either “inbound” or “outbound”, corresponding to whether the traffic is entering or leaving the machine.
protocol	Either “tcp” or “udp”, all lowercase – we will just implement two of the most common protocols.
port	Either (a) an integer in the range [1, 65535] or (b) a port range, containing two integers in the range [1, 65535] separated by a dash (no spaces). Port ranges are inclusive, i.e. the port range “80-85” contains ports 80 and 85. Given a port range, you may assume that the range is well-formed i.e. the start of the range is strictly less than the end.
IP address	Either (a) an IPv4 address in dotted notation, consisting of 4 octets, each an integer in the range [0, 255], separated by periods or (b) an IP range containing two IPv4 addresses, separated by a dash (no spaces). Like port ranges, IP ranges are inclusive. Given an IP range, you may assume that the range is well-formed i.e. when viewed as a number, the starting address is strictly less than the ending address.

For example, the following are all valid inputs:

```
inbound,tcp,80,192.168.1.2
outbound,tcp,10000-20000,192.168.10.11
inbound,udp,53,192.168.1.1-192.168.2.5
outbound,udp,1000-2000,52.12.48.92
```

You may assume that the input file contains only valid, well-formed entries.

What to implement

Given a set of firewall rules, a network packet will be accepted by the firewall if and only if the direction, protocol, port, and IP address match at least one of the input rules. If a rule contains a port range, it will match all packets whose port falls within the range. If a rule contains an IP address range, it will match all packets whose IP address falls within the range.

Your job is to implement a Firewall class, whose interface contains two items:

- A constructor, taking a single string argument, which is a file path to a CSV file whose contents are as described above, e.g, in Ruby, this would be `Firewall.new("/path/to/fw.csv")`.
    - Note that you do not need to define a static 'new' method – simply use the constructor syntax in the language that you chose.
    - Remember that you may assume that all content in the input file is valid.

- A function, `accept_packet`, that takes exactly four arguments and returns a boolean: true, if there exists a rule in the file that this object was initialized with that allows traffic with these particular properties, and false otherwise.
    - direction (string): "inbound" or "outbound"
    - protocol (string): exactly one of "tcp" or "udp", all lowercase
    - port (integer) – an integer in the range [1, 65535]
    - ip_address (string): a single well-formed IPv4 address.

You may assume that all arguments passed into the `accept_packet` function will be well-formed according to the spec above.

Feel free to define any additional functions or classes you might want; the only thing we ask is that you don't change the interface described above.

In your implementation, you should consider efficiency and tradeoffs in both space and time complexity. There may be a massive number of rules (use 500K entries as a baseline), and real-world firewalls must be able to store this in a reasonably compact form while introducing only negligible latency to incoming and outgoing network traffic.

Some examples

These examples are in Ruby, but should translate with minor changes to most languages. The lines that are not prefaced by ">" represent the output in a typical REPL; your `accept_packet` function should return the value, not print it.

```
> fw = Firewall.new("/path/to/fw.csv")
> fw.accept_packet("inbound", "tcp", 80, "192.168.1.2") # matches first rule
true
> fw.accept_packet("inbound", "udp", 53, "192.168.2.1") # matches third rule
true
> fw.accept_packet("outbound", "tcp", 10234, "192.168.10.11") # matches second rule
true
> fw.accept_packet("inbound", "tcp", 81, "192.168.1.2")
false
> fw.accept_packet("inbound", "udp", 24, "52.12.48.92")
false
```

Evaluation guidelines

We will be evaluating your code in three main areas:

1. Functionality
    a. Does the code work correctly?
    b. Are there any valid inputs for which the code returns incorrect results or breaks?
    c. Did you test your code, exercising common cases and edge cases? If you have test files or scripts, please include them in your repository. Otherwise, please include a description of how you tested your code in your submission.

2. Code clarity and cleanliness
    a. Is the code well-structured and does it make use of object-oriented principles where appropriate?
    b. Is the logic well encapsulated? Is common logic shared in functions with reasonable interfaces?
    c. Is it easy to understand what the code is doing? Are the names of variables and functions descriptive? Does the code avoid overly complex or esoteric syntax?
    d. Are particularly tricky areas of the code well-commented to guide the reader?

3. Performance
    a. There are no "right or wrong" answers when it comes to this section. Making tradeoffs between space and time complexity is a core component of this coding assignment.
    b. We are interested in seeing that you thought about performance instead of simply settling for the naïve solution. Even if you do not get around to implementing an optimal solution, we are interested in ideas that you thought about or areas of the code that you've identified as candidates for optimization.
    c. If you are selected for a subsequent round of interviews, you and your interviewer may discuss performance-related tradeoffs at length, so please be ready to talk about the decisions that you made.
    d. In general, we expect the code to work "reasonably quickly" (i.e. not appear unresponsive) for large datasets of 500K – 1M items, after the dataset has been loaded (i.e. after the constructor has successfully returned).

Submission guidelines

After time is up, please do the following:

1. Put all of your code in a Git repository hosted on a site like Github or Bitbucket. Make sure to include any test code or scripts that you wrote.

2. Include a README file, which includes anything you'd like to communicate to the person that is reviewing your code. This may include items such as:
    a. how you tested your solution
    b. any interesting coding, design, or algorithmic choices you'd like to point out
    c. any refinements or optimizations that you would've implemented if you had more time
    d. anything else you'd like the reviewer to know

3. Include in your README file the particular area of the team that you're interested in. Descriptions of the three areas can be found on the next page. If you're interested in more than one area and/or have a ranking, please let us know!

The **platform team** works on the core components of the Policy Compute Engine, including high availability, caching and persistence layers, API support, background jobs, reporting infrastructure, cluster management, authentication and authorization, and more. The team ensures the system can scale to meet the demands of our customers, including active connections to tens of thousands of managed servers. You'll work with sophisticated open-source packages on a service-oriented architecture. Interns and new grads have worked on interesting problems like:

- Building and automating the capability to run tens of thousands of real Linux clients on a small number of host machines
- Building an application metrics subsystem to monitor performance of our servers
- Diving into the depths of the caching layer at the core of the PCE that serves security policies, collecting performance data, and identifying bottlenecks

The **policy team** works on the portion of the Policy Compute Engine that computes the security policy that is sent to Illumio-managed servers. We write code to handle the constant changes that happen in today's datacenters while ensuring that all servers receive necessary updates to their security policy quickly and at scale. We use Postgres as our database and Redis as our caching layer. We are constantly thinking about correctness, fault-tolerance, reliability, and performance. Interns and new grads have worked on numerous features, enabling our customers to:

- Analyze their rules and take action to write smaller and more efficient policies
- Write user-based rules to add an additional layer of security to critical infrastructure
- Write policies to control VPN connections between servers in public clouds and private datacenters

The **data team** works on three main areas: Illumination (our flagship data visualization feature), analytics, and cyber strategy. We take massive amounts of network flow data captured by Illumio-managed servers and combine them with metadata on the Policy Compute Engine in order to drive insights in customer environments, visualize complex application dependencies, recommend optimized security policy, and answer questions about current and historical network data. We recently launched "Illumination 2.0" with the Policy Generator and Explorer features. New team members have worked on new features and prototyped infrastructure improvements, like:

- Building out a fault-tolerant data pipeline to serve interfaces which enable queries on network flow data
- Investigating and prototyping a shared Redis cache to annotate network flow data with policy metadata for more efficient querying