University of Pretoria
Software Engineering - COS 301

Defendr Specification

Dark nITes
3 May 2019

Team members:

MI(Muhammed) Carrim	R(Ruslynn) Appana	SNL(Sisa) Khoza	CP(Christiaan) Opperman	J(Jeandre) Botha

Contents

1 Introduction

1.1 Purpose

The intent of this software requirements specification document is to provide the requirements and implementation plan for the project named ***Defendr***. This document serves to clarify and communicate all stakeholder's understandings and expectations of ***Defendr***.

This document is written for the perusal of the aforementioned stakeholders: our client and customer Advance, including the COS 301 module and Computer Science department of the University or Pretoria as clients too. The final audience member is the development team itself, Dark nITes.

1.2 Scope

Defendr is to be a blackbox implementation of a DoS protection service, as well as a network load-balancer for various back-end applications (henceforth called service collectively). The service is to be situated between the client and server; request from the client are to pass through the service, dropping/blacklisting offending packets/IPs. The service should employ direct server return; responses from the server are to be sent directly back to the client, and not routed back via the service.

Packets that are permitted to pass through DoS protection will then be subject to the load-balancer. Various load-balancing pools with multiple instances of a back-end situated in them. According to the algorithm that is managing the particular pool's load-balancing, packets will get passed to the intended back-end instance.

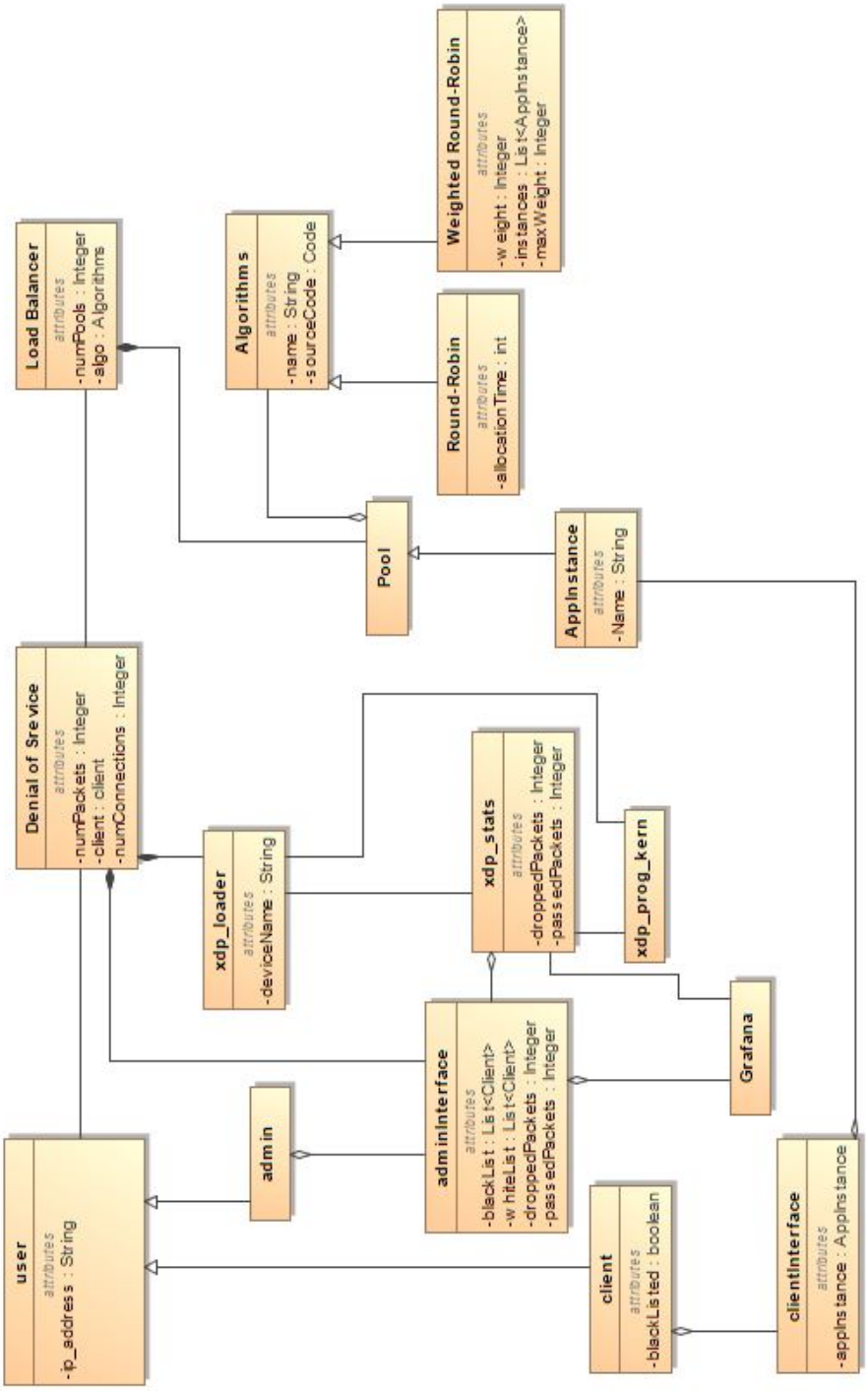
The service will measure the validity of request packets sent to a protected service by two criteria: # of packets per second, and # of connections. The limitations will be specified by the owner of the back-end being protected.

The service will also comprise of two user interfaces; an administration interface, and client interface. The interface will provide access to metrics comprising of:

- Current servers being protected
- Current status of the server, i.e. total # of packets, # of packets being let through (success rate), # of packets being dropped (failure rate)
- A heat-map that displays the geo-location of the origin of client request to protected back-ends
- List of blacklisted IPs
- Internal overhead

in a raw and graphical data format

1.3 Domain Modal



1.4 Definitions, Acronyms, and Abbreviations

Term	Definition
Blackbox	A method of software testing that examines the functionality of an application without peering into its internal structures or workings.
DOS	Disk Operating System.
DoS	Denial of Service.
DDoS	Distributed denial of service.
Load balancer	A subset of the service that will distribute network traffic to various instance of an application, as determined by the current governing algorithm.
Client	The originating device from which a request is received.
IPs	Internet Protocol.
Packets	The units of data that are being transmitted from a client to a protected application.
XDP	eXpress Data Path.
eBPF	extended Berkley Packet Filter.
Prometheus	A monitoring system that has a time-series metrics database and ways to query said metrics.
Grafana	A toolkit that presents data in a graphical form. Can be used in conjunction with Prometheus to graph the service's metrics.

1.5 References

References

2 User Characteristics

2.1 Client

The client is the primary user of the system. Their actions will only include the sending of packets and receiving of responses. By using this system, they will be able to protect the integrity of their application, and experience an increase to throughput by virtue of DoS protection and network load-balancing. The client will use the front-facing segment of the system. As such a user, who can not cause much detriment to the system, they are expected to have no more knowledge other than how to use the (protected and balanced) application.

2.2 Administrator

Administrators will have the duties of installing components of the system and maintaining its health and performance. As these decisions will determine the success of the service, as well as well as require a level of skill, these users would be expected to have experience with networking and some degree software development and maintenance. These users will be able to make changes to the system that are integral to its running, e.g. manually blacklisting IPs/IP ranges, adjusting packet filtering rules or removing services from protected pools.

3 Functional Requirements

3.1 DoS Protection

- R1.1. The subsystem must be able to detect and mitigate a DoS attack by dropping the offending packets from the source IP or IP ranges, and allow access after an exponential timeout.
- R1.2. The subsystem must provide functionality to gather metrics. These metrics packet success/failure rates, total # of packets per pool/back-end.
- R1.3. The subsystem rules that determine an incipient DoS should be able to be manually altered.

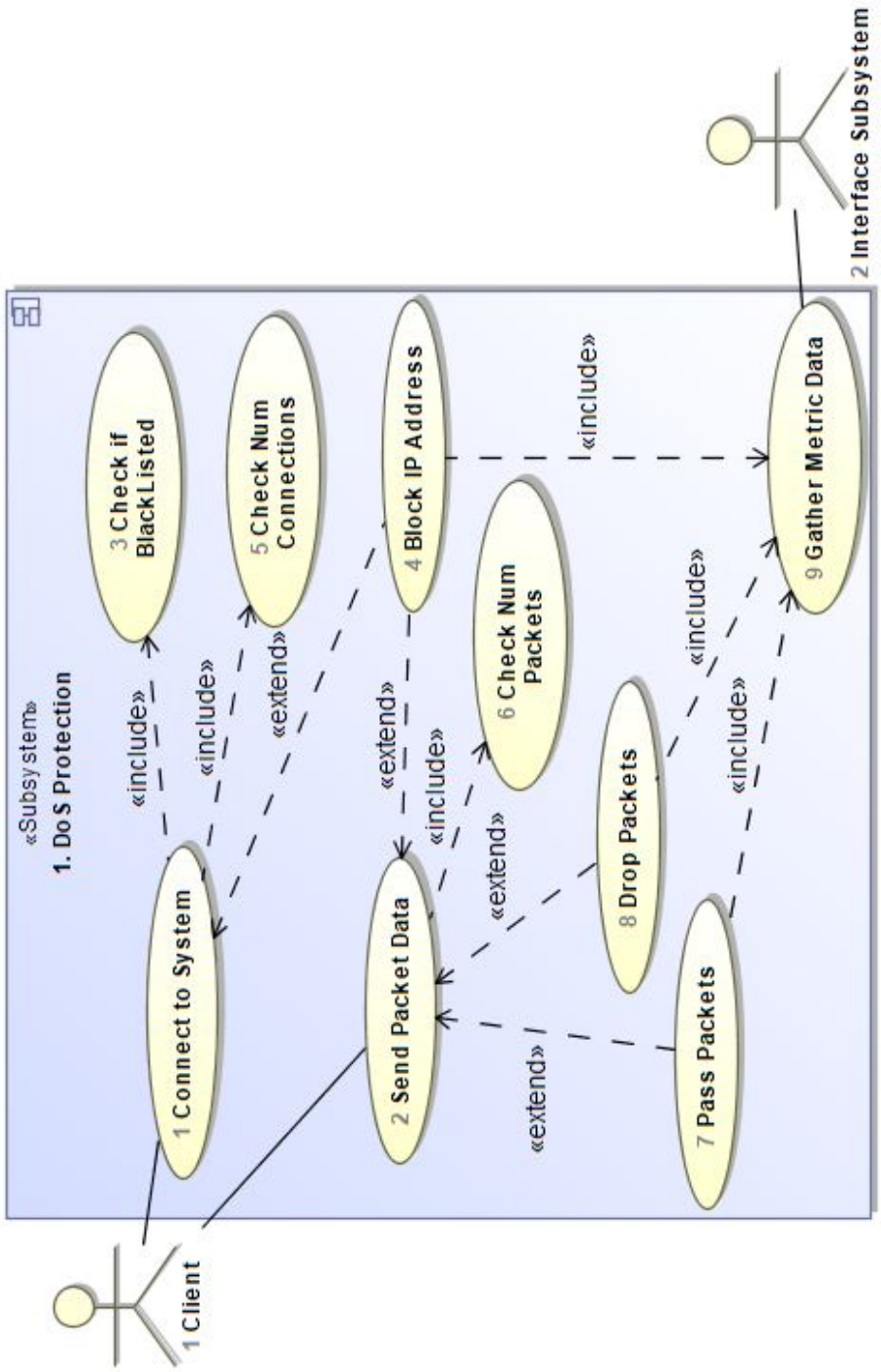
3.2 Load-Balancer

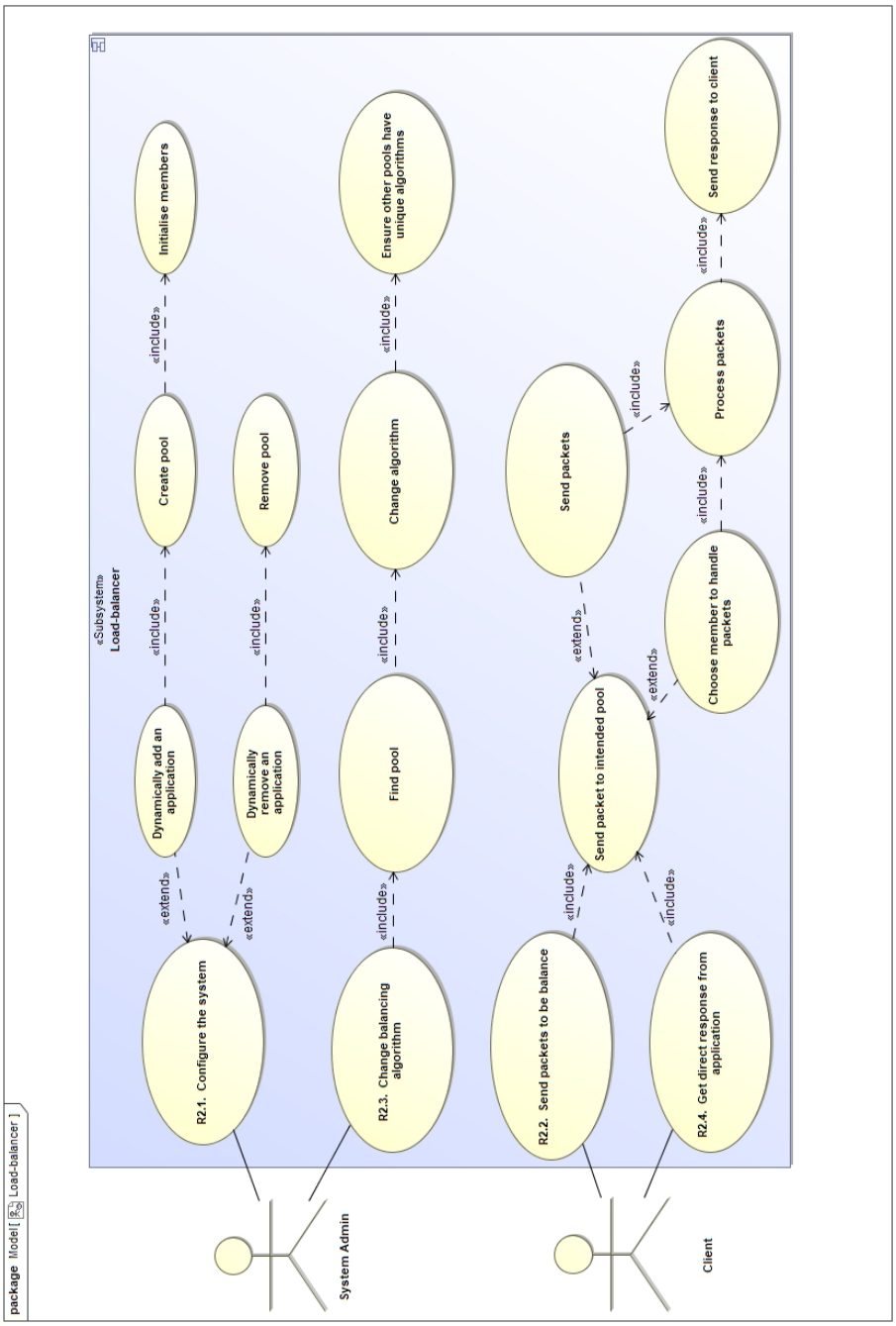
- R2.1. The subsystem needs to be configurable; applications/back-ends should be able to be dynamically added/removed
- R2.2. The subsystem must have multiple load-balancing pools, where pools are defined by the back-ends. Members of each pool are instances of the back-end that are to be load-balanced.
- R2.3. The subsystem must support multiple load-balancing algorithms, of which Round Robin and Weighted Round Robin must be included, different per pool. These algorithms should be changeable on-the-fly. Network load anomaly detection, with an option of prediction, should also be included.
- R2.4. The method of request response should be via a direct server response to the requesting client, that is responses should not return via the service.

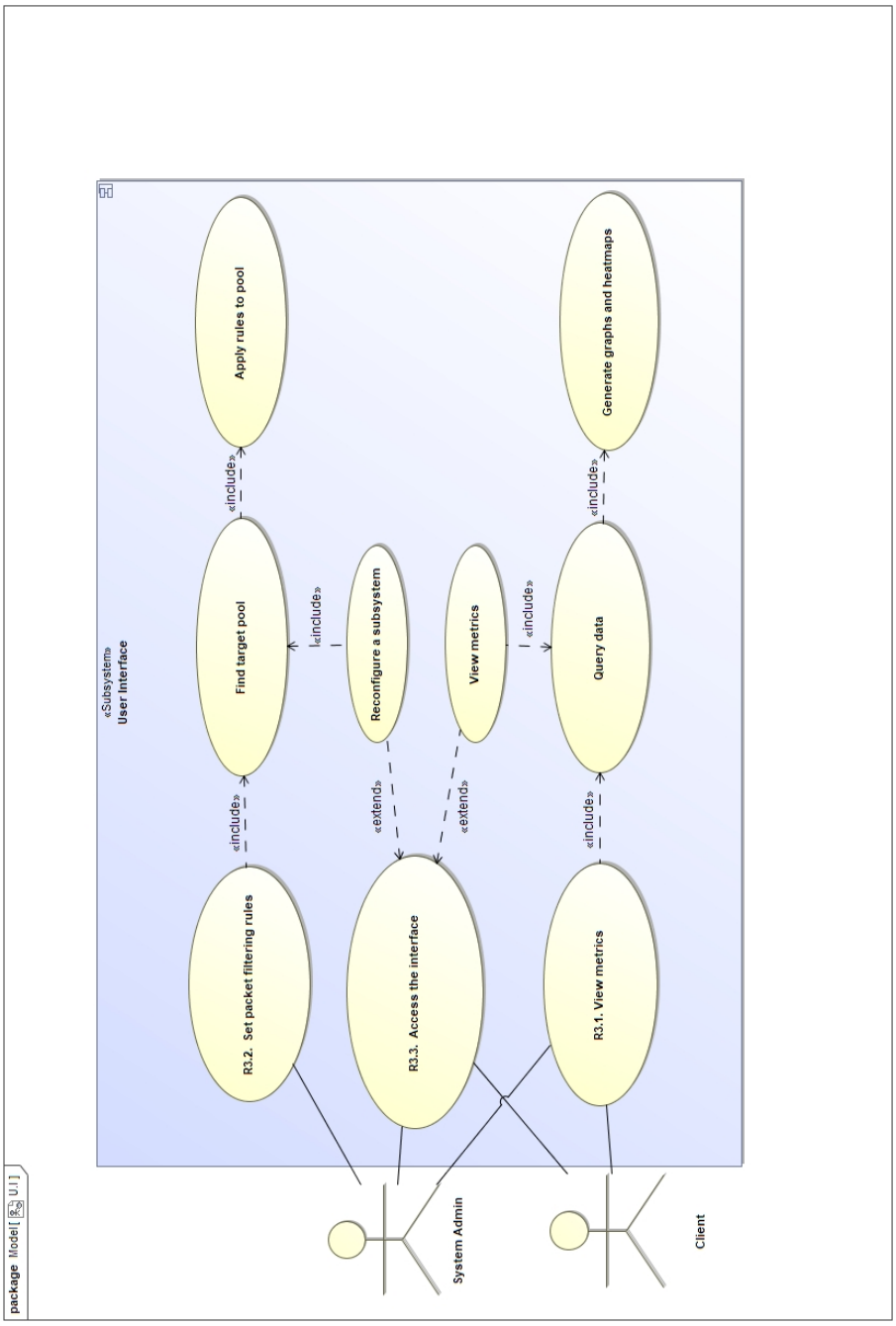
3.3 User Interface

- R3.1. The interface should show the metrics in a graphical medium, i.e. graphs and heat-maps.
- R3.2. The interface should also provide a means of manually configuring the service, e.g. blacklisting/white-listing IPs/IP ranges, defining rules that govern network traffic
- R3.3. The interface must be accessible from anywhere, and not from just a specific machine, i.e. the interface must be hosted somewhere accessible, but for demo purposes localhost will do

3.4 Use Case Diagrams




package Model [Load-balancer]
«Subsystem»
Load-balancer
Initialise members
Create pool
Remove pool
Change algorithm
Ensure other pools have unique algorithms
Send packets
Process packets
Send response to client
Dynamically add an application
Dynamically remove an application
Find pool
Send packet to intended pool
Choose member to handle packets
R2.1. Configure the system
R2.3. Change balancing algorithm
R2.2. Send packets to be balance
R2.4. Get direct response from application
System Admin
Client
«include»
«extend»


package Model [U.I]
«Subsystem»
User Interface
Apply rules to pool
Find target pool
Reconfigure a subsystem
View metrics
Query data
Generate graphs and heatmaps
«include»
«include»
«include»
«include»
«include»
«include»
«extend»
«extend»
R3.2. Set packet filtering rules
R3.3. Access the interface
R3.1. View metrics
System Admin
Client

4 Quality Requirements

4.1 Performance

The software system must be able to handle the same number of packages that the application which it is protecting, usually handles. In other words, the system should have the same performance capabilities as the services it is protecting so that it will not cause a bottleneck.This can be measured by looking at the drop rate and number of packages.

4.2 Security

The security of the system has to be high, since one of the main purposes of the software is to protect the services against DOS attacks. The software runs on the kernel which will expose the client to other threats if the security isn't high enough.

4.3 Availability

The system should have the same availability as the applications that it protects. Seeing as the system will also be a program, it will in all probability encounter challenges seeing that it is fallible. Thus we came to an agreement with the client, that a 99.5% availability will be acceptable.

4.4 Maintainability

Maintainability is important in two ways. Namely the upkeep of the code and the addition and removal of services which have to be protected.
- Upkeep of code: The software must be able to be changed so that the software as a whole works on the latest version of Linux.
- Addition and removal of services: The user should be able to add and remove services as they see fit through the use of a user interface.

4.5 Scalability

The system should be able to scale to the size of the server which it is protecting, without requiring additional changes to be made to the basic structure of the system. It should also be able to accommodate a different number of load balancing pools.

4.6 Cost

Cost isn't very important, since all the technologies that are being used to design the system is free and open sourced. It will not cost anything to create the software, except for the time needed to learn how to use the technologies. Installing the software will only cost time, since all that needs to be done is to compile the code and link it with the kernel.

4.7 Usability

The system will be easy to use by the client, because of the intuitive user interface.The only skills the client will need will be knowledge on how to install the software on a server.

4.8 Flexibility

Th system will need to be highly flexible so that it can accommodate different servers with different amounts and types of services which it protects. This will be obtained through the second interface which allows users to allocate and remove services which have to be protected.

4.9 Monitorability

The system will be monitored through a GUI which displays packet rates (total and per pool), drop rates, heat-map, packet size, internal overhead and white and black listed IPs. This GUI should be accessible anywhere, i.e. a website online.

5 Trace-ability Matrix

Requirement	**Priorily**	**DoS**	**Load-balancer**	**U.I.**
R1.1	1	X		
R1.2	1	X		
R2.1	1		X	
R2.2	1		X	
R2.3	1		X	
R2.4	1		X	
R3.1	1			X
R3.2	1			X
R3.3	1			X