# Summary

## Meeting time and location

24 April 2019 @ 11h30 in the Paris conference room

46 Ingersol Rd, Lynnwood Glen

## Participants

* Advance
  + Roelof
  + Loutjie
* Dark nITes
  + Ruslynn
  + Jeandre
  + Muhammed
  + Sisa
  + Christiaan

## Decisions/Amendments made/adopted

* Meeting schedule will be an as-needed basis. As soon as we have progress to show meetings can be scheduled. Frequency is not a limiter.
* Communication is best kept to email; everyone can be in the loop and a trace remains

# Overview

Defendr is to be a backbox implementation of a DOS prevention service, so well as a load-balancer (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.

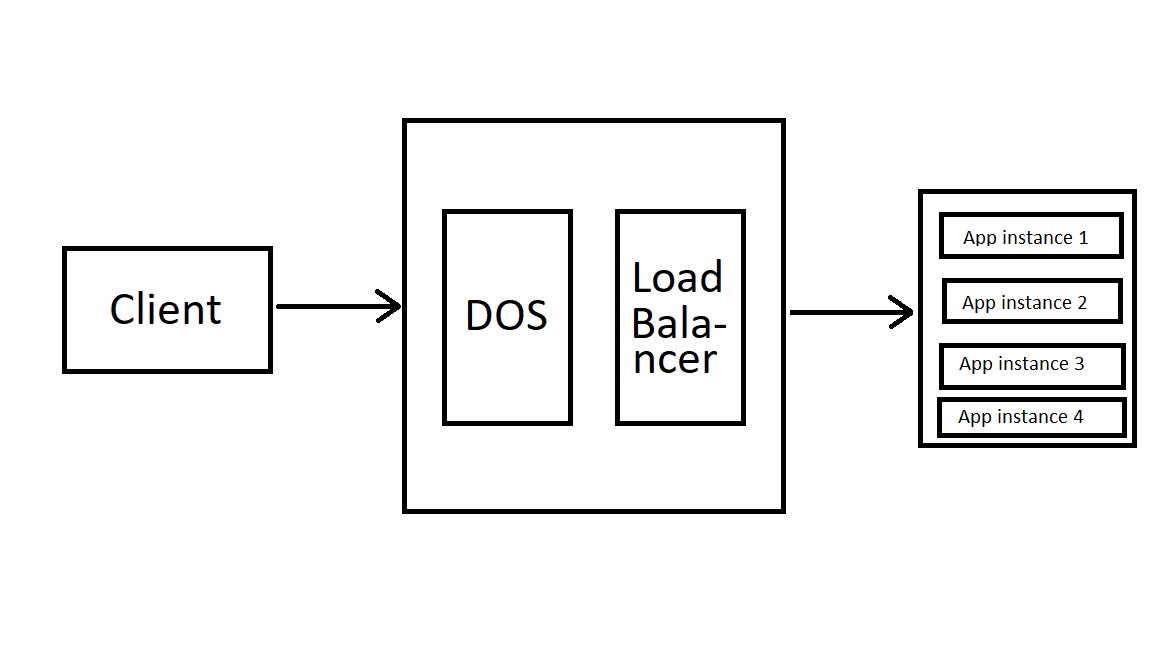
Further, responses from the server are to be sent directly to the requesting client and not via the service (direct server return). Facebook’s Katron project is a good place to start to get a better understanding.

Essentially what is required is to provide:

* The service that will handle DOS mitigation and load balancing
  + DOS: subsystem that will act sentinel to prevent/mitigate attacks
  + LB: balance network traffic to ensure a good spread of work across the servers
* A user interface that will
  + Allow for manual configuration of the service (black/whitelist IPs)
  + Setting limits that will determine what constitutes an attack
  + Metrics that will display
    - Current servers being protected
    - Current status of the server, i.e. total # of packets, # of packets being let through, # of packets being dropped
    - A heatmap that displayed the geolocation of clients sending request to the a server
    - List of blacklisted IPs
  + Be accessible from everywhere, i.e. an interface that can be hosted on an online service that will then allow modification of the service. For demo purposes, this may be implemented on the localhost
  + A design pattern may be used to guide the UI design

A Unix kernel of version 4.16+ is suggested for use of this project.

# The approach



As various methods exist for detecting a DOS attack, the current strategy will be to evaluate on two criteria:

* Packets per second
* Number of connections

Requests will be evaluated for their threat-level by using the above criteria, once packets are approved the load-balancer will determine which instance of the application the packets will be sent to.

# Technologies

* XDP (eXpress Data Path)
  + Allows the skipping of the kernel’s networking layer, allowing user-space programs to do packet processing
* eBPF (extended Berkley Packet Filter)
  + Gives functionality to allow the execution of mini-programs, that execute on events e.g. disk I/O, to run in a safe VM of the kernel.
* Prometheus
  + A monitoring system that has a time-series database, a time-series metrics database and ways to query said metrics
    - Prometheus has mini-programs called “Exporters” that interpret the data in a usable format. Readily available implementations exist, however custom variations are available. It is suggested that we use a custom implementation as we will be reading kernel data structures.
* Grafana
  + Basically, a toolkit that presents data in a graphical form. Can be used in conjunction with Prometheus to graph the service’s metrics
* Design patterns
  + As XDP and eBPF are not conventional, traditional design patterns may not be applicable. We will use best coding practices, and possible architectural patterns specifically for this domain
* C
  + We’re working at a low level. So C.
* LLVM framework

# Reading materials

Some material was suggested to better grasp load-balancing methodologies. The specific suggested technologies to read up were

* Google Maglev (<https://ai.google/research/pubs/pub44824>)
* Facebook Katran (<https://code.fb.com/open-source/open-sourcing-katran-a-scalable-network-load-balancer/>)
* GitHub Director (<https://github.com/github/glb-director>)

# First deliverable

* SRS document
  + System overview with the system architecture visible (a deployment model?)
  + The need for use-case diagram is not necessary, written language will suffice
  + Wireframes and mock-ups of the UI are also acceptable

# Testing specification

## Unit testing

* Golang (Go is it’s own language, however implemented in C++ thus can be assumed to be a subset of C)
  + Unit testing framework for Go
* Any other unit testing frameworks that are native to our language will need to be used. As such we need to properly state what will be used so that we can proceed to find testing frameworks

# Extra added features/functions

* Some form of AI for DOS mitigation/prevention

# To-do

* Provide view access right to Roelof (Naude-R) and Loutjie (TBA) on GitHub for oversight
* Solidly state which languages will be used
* Everything that is listed under the “[First deliverable](#_First_deliverable)” heading