Санкт-Петербургский государственный университет

Кафедра системного программирования

Программная инженерия

Кузиванов Сергей Юрьевич

Распространение существующих автоматизированных тестов для сетей хранения данных на изолированные сегменты сети

Курсовая работа

Научный руководитель:
ст. преп. Я.А. Кириленко

Консультант:
старший ведущий инженер-разработчик ПО
Dell Technologies
Д. Л. Довженко

Санкт-Петербург
2020

Оглавление

Введение

В компании Dell EMC существует задача распространения автоматизированных тестов для сетей хранения данных на подсистемы, находящиеся в изолированных сегментах сети. Тестовый контроллер общается с агентами, находящимися на тестируемых системах, через TCP-сокеты. Для простоты можно считать, что передаваемые данные незашифрованы, а все приложения работают в ОС Linux. Внести изменения в код агентов не представляется возможным, а переписывание автоматизированных тестов слишком трудоёмко. Фактически, моделью поведения агента является чат-сервер, принимающий от тестового контроллера команды в виде текстовых сообщений. Поэтому в первом приближении можно считать, что необходимо перехватывать незашифрованные текстовые сообщения и иметь возможность модифицировать их со сменой конечного получателя в чат-сети из одного чат-сервера и ограниченного количества клиентов, каждый из которых подключается к общему чат-серверу.

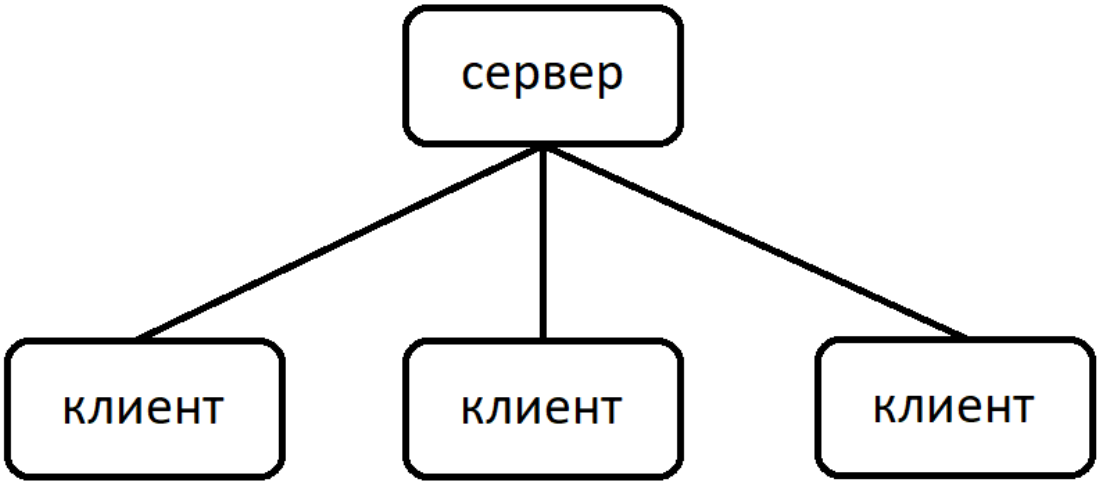
Постановка задачи

Цель данной работы - разработать прототип приложения под Linux, позволяющего создавать приложения, перехватывающие и модифицирующие поток сетевых данных, передающихся на базе протокола TCP/IP в незашифрованном виде. Так как тестирование реализуемого инструмента планировалось проводить через тестирование работы чат-приложения с модифицированной архитектурой сетевого взаимодействия, а не через тестирование работы тестирования СХД, то архитектура создаваемых приложений должна быть по возможности независимой от конфигурации сети, в которой создаваемое приложение будет работать. Для достижения цели были поставлены следующие задачи:

1) Установка и запуск чат-приложения
2) Разработка модифицированной архитектуры, которая включает создаваемое приложение
3) Разработка логики работы создаваемого приложения
4) Написание прототипа создаваемого приложения
5) Разработка основной логики инструмента
6) Реализация инструмента

1. Обзор

Архитектура чат-приложения, на котором происходило тестирование, следующая:



Все клиенты подключаются к единому чат-серверу, вся логика реализована в чат-сервере, клиенты только отображают информацию, приходящую от сервера.

2. Реализация

2.1. Архитектура взаимодействия

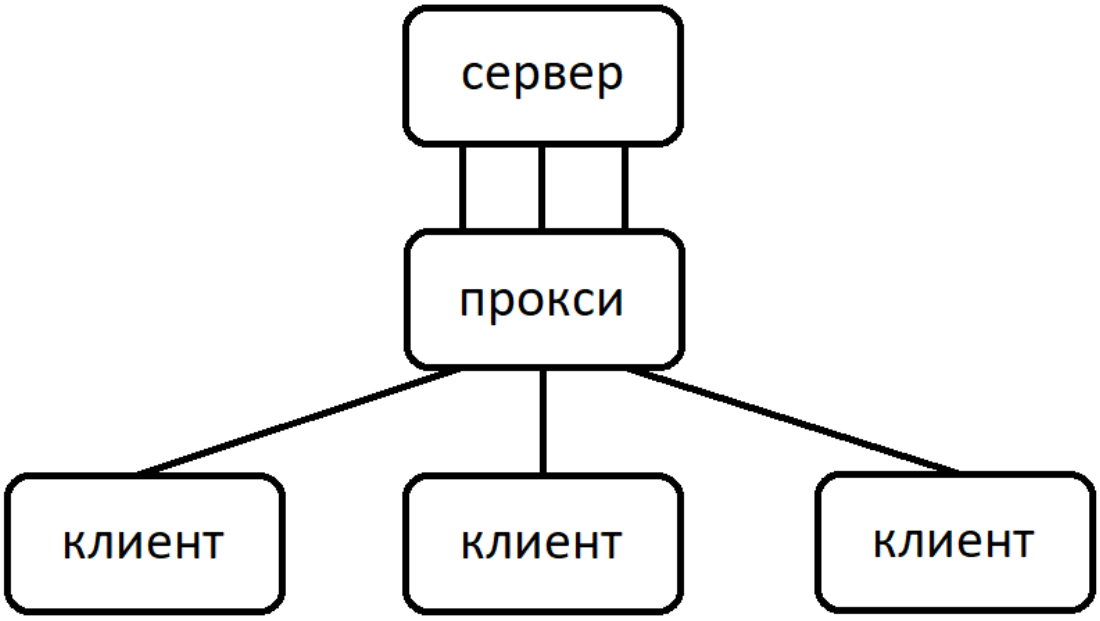
Были рассмотрены различные способы решения данной задачи, а именно:

- Настройка существующего прокси-сервера (например, simpleproxy)
- Создание программы, использующей уже существующие утилиты (например, wireshark)
- Написание своего прокси-сервера без использования реализованных утилит

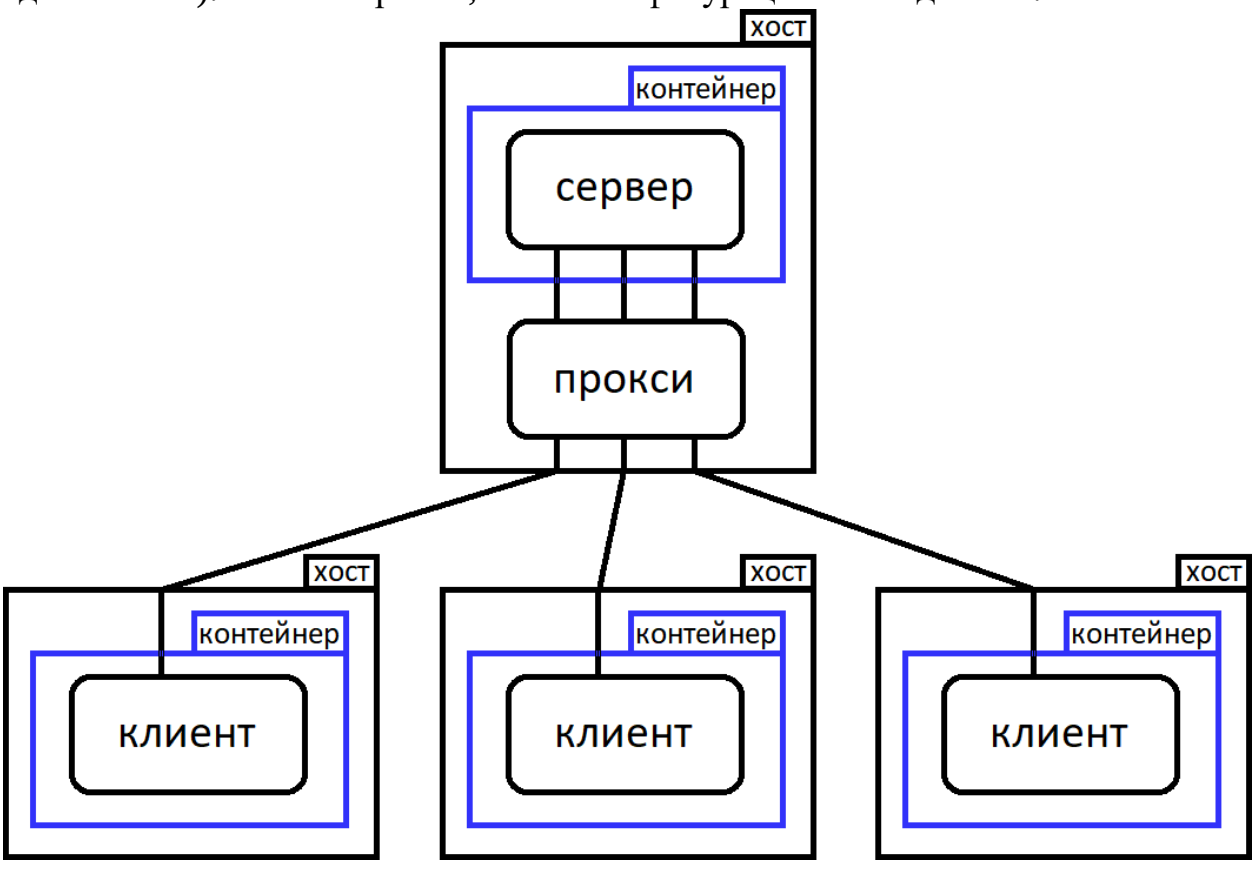
В итоге было принято решение написать прототип инструмента, создающий программу прокси-сервера, перехватывающего и модифицирующего поток незашифрованных текстовых данных на базе протокола TCP/IP, так как:

1. Использование таких утилит, как wireshark и tcpdump не представляется возможным, так как данные утилиты не позволяют модифицировать сетевые пакеты.
2. Использование таких инструментов, как eBPF, libpcap и nfqueue также не представляется возможным, так как для работы с данными утилитами требуется гораздо больший объём знаний об устройстве и работе сети, чем у меня были изначально.
3. Написание своего прокси-сервера на базе какого-либо уже написанного прокси-сервера (например, simpleproxy) мне не показался лучшим вариантом, так как было проще связаться при необходимости с разработчиком выданного мне чат-сервера и узнать о его работе вместо чтения документации по работе других уже написанных прокси-серверов.

Таким образом новая конфигурация клиентов и чат-сервера в случае выше должна быть такой:

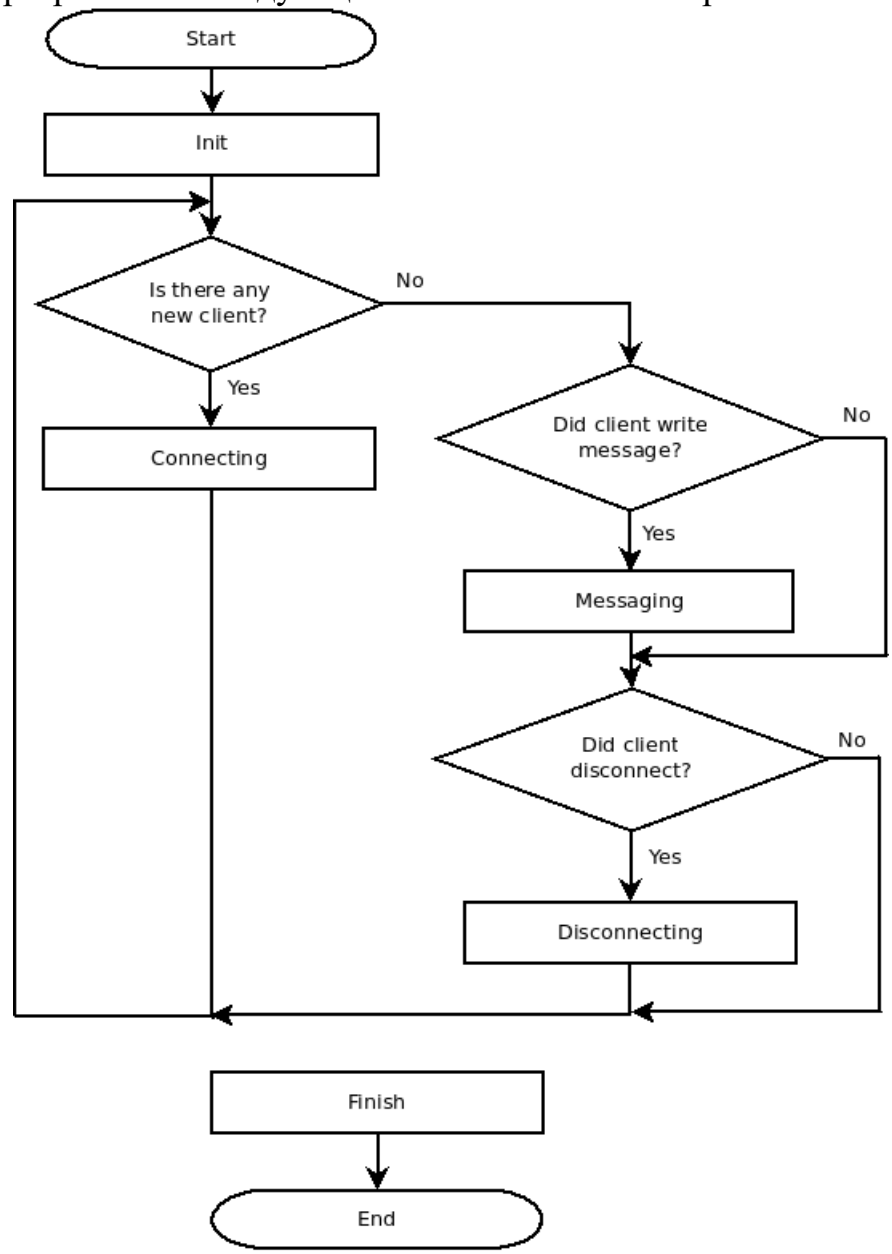


Также планировалось путём использования реализуемого инструмента автоматически перестраивать сетевое взаимодействие программ в данной сети с использованием контейнеров docker (в данной чат-сети запуск клиентов в контейнере обеспечивал бы переадресацию подключения с реального чат-сервера на создаваемый прокси-сервер, а запуск сервера в контейнере обеспечивал бы изоляцию реального чат-сервера при тестировании работы в условиях запуска чат-сервера и прокси-сервера на одном хосте). Таким образом, новая конфигурация выглядит так:



2.2. Логика создаваемого приложения

Так как разрабатываемый инструмент не должен создавать прокси-сервера с именно такой архитектурой (при которой на каждое соединение с клиентом создаётся одно соединение с сервером), то было принято решение разработать максимально независимую от архитектуры проекта, в который встраивается создаваемый инструментом прокси-сервер, логику работы. В результате были выделены 5 основных блоков в работе прокси-сервера, в которых содержится вся основная логика работы прокси-сервера, а также была разработана следующая блок-схема логики работы:



1) Init – действия, происходящие при запуска прокси-сервера
2) Connecting – действия, происходящие при получении запроса на подключение

3) Messaging – действия, происходящие при отправке данных на прокси-сервер
4) Disconnecting – действия, происходящие при потере соединения с прокси-сервером
5) Finish – действия, происходящие при появлении непредвиденной ошибки во время работы прокси-сервера

2.3. Описание работы инструмента

Реализация вышеописанных основных блоков программистом, использующим данный инструмент для создания прокси-серверов, осуществляется через написание одноимённых функций, объявленных в файле user_funcs.h, на C или C++:

1) int init(int argc, char* argv[])
2) int connecting(int proxy_socket_fd)
3) int disconnecting(int socket_fd)
4) int messaging(int socket_fd)
5) void finish()

Для упрощения реализации программистом вышеуказанных функций были реализованы вспомогательные функции, которые объявлены в файлах proxy_funcs.h и proxy_trace.h:

1) int close_socket(int socket_fd) – закрывает соединение (рекомендуется вместо стандартного close)
2) int connect_to_server(const char* node, const char* service) – устанавливает соединение с сервером (node и service аналогичны использующимся в getaddrinfo)
3) int accept_connection(int accepting_socket_fd) – принимает входящее соединение (рекомендуется вместо стандартного accept)
4) char* get_ip_addr(int socket_fd) – возвращает IP-адрес
5) int trace(TraceLevel tl, const char* format, ...) – рекомендуется для debug-вывода

Для создания прокси-сервера с заданной логикой роботы, прописанной через 5 основных функций, была разработана утилита newproxy, принимающая файл с реализованными функциями и создающая исполняемый файл прокси-сервера.

Использование:

newproxy -s <source_name> -p <port> [-h <host>] [-n <file_name>] [-c] [-?]

-s <source_name> - название файла с основными функциями, реализованными программистом

-p <port> - порт, прослушиваемый прокси-сервером на предмет входящих подключений

-h <host> - хост, на котором работает прокси-сервер (по умолчанию "localhost")

-n <file_name> - название создаваемого прокси-сервера (по умолчанию "proxy")

-c – скомпилировать прокси-сервер как проект на C (по умолчанию как проект на C++)

-? – написать справку и выйти

3. Апробация прототипа

Тестирование созданного прототипа осуществлялось путём анализа debug-вывода создаваемого прокси-сервера, а также путём анализа работы клиентов и чат-сервера в описанной выше чат-сети и создаваемых прокси-сервером файлов. В результате проведения тестов поведения прокси-сервера, которое выходило за рамки предсказуемого и допустимого, обнаружено не было.

Заключение

В ходе работы получены следующие результаты.

1) Ознакомился с предметной областью, а именно изучил основы работы с контейнерами Docker и с сокетами TCP в Linux
2) Разработал прототип инструмента
3) Разработал несколько демонстрационных приложений, создаваемых прототипом инструментом

Список литературы

[1] Man pages – https://www.opennet.ru/man.shtml
[2] Docker docs – https://docs.docker.com/get-docker/
[3] Chat project – https://github.com/kseniadumpling/ChatProject
[4] Scapy overview – https://habr.com/ru/post/208786/
[5] Wireshark overview – https://losst.ru/kak-polzovatsya-wireshark-dlya-analiza-trafika