Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования "Национальный исследовательский университет

"Высшая школа экономики"

МИЭМ им. А.Н. Тихонова

Департамент прикладной математики

10.05.01 «Компьютерная безопасность»

**ОТЧЕТ**

**по лабораторной работе №3 по дисциплине “Программирование алгоритмов защиты информации”**

Выполнили:

Студент группы СКБ202 Гришаев Р.

Студент группы СКБ201 Сергеев Н.

Проверил:

Преподаватель

Нестеренко А.Ю.

Москва 2024

# Проект, ориентированный на изучение заданной СФ или задачи разработки СКЗИ

# 4.4. Простейший VPN типа точка-точка (с использованием виртуальных сетевых адаптеров)

# Обзор

Задача - Изменение алгоритма шифрования в vtun версии 2.5 с BlowFish (OpenSSL) на отечественный алгоритм шифрования Кузнечик.

Почему выбрана версия 2.5? Потому что начиная с 3.0, разработчики внедряли в основном поддержку новых алгоритмов шифрования, что отягощало проект и код нагрузкой, которая в этой работе не нужна и отключается. В 2.5 можно отключить SSL и сжатие через через ключи `--disable-ssl --disable-lzo --disable-zlib --disable-shaper`

3.0.0-pre0:

Added multiple cipher support. (dpf)

Added multiple cipher modes support. (dpf)

Added support for different sized keys. (dpf)

Re-sync ciphers when using non-ECB modes over a UDP

connection. (dpf)

Fixed Bug#908824 (persist=keep not re-applying routes) (dpf)

# Проделанная работа

## Изучение Vtun.

The easiest way to create Virtual Tunnels over TCP/IP networks

with traffic shaping, compression and encryption.

Документации на код как таковой у проекта нет, из источников - оф. сайт <https://vtun.info/> и разные usage-cases, из которых мы и почерпнули информацию о работе программы.

## Первый запуск.

Первое, что мы решили сделать, это поднять vtun на стандартном конфиге и с OpenSSL. Подбором конфига занимался Никита, а тем, как это реализовать на “инфраструктуре” - Ростислав.

Конфиги:

client.conf

| #client  options {  port 5344; # Connect to this port.  timeout 60; # General timeout  syslog daemon;  # Path to various programs  ppp /usr/sbin/pppd;  ifconfig /sbin/ifconfig;  route /sbin/route;  firewall /sbin/ipchains;  ip /sbin/ip;  }  # TUN example. Session 'cobra'.  cobra {  password test;  type tun; # Device tun1  persist yes;  encrypt yes;  up {  # Connection is Up  # Assign IP addresses.  ifconfig "%% 10.3.0.2 pointopoint 10.3.0.1 mtu 1450";  };  } |
| --- |

server.conf

| options {  port 5344; # Listen on this port.  # Syslog facility  syslog daemon;  # Path to various programs  ppp /usr/sbin/pppd;  ifconfig /sbin/ifconfig;  route /sbin/route;  firewall /sbin/ipchains;  ip /sbin/ip;  }  # Default session options  default {  compress no; # Compression is off by default  speed 0; # By default maximum speed, NO shaping  }  # TUN example. Session 'cobra'.  cobra {  pass test;  type tun; # IP tunnel  proto tcp; # UDP protocol  comp no; # LZO compression level 9  encrypt yes; # Encryption  keepalive yes; # Keep connection alive  up {  # Connection is Up  # 10.3.0.1 - local, 10.3.0.2 - remote  ifconfig "%% 10.3.0.1 pointopoint 10.3.0.2 mtu 1450";  };  } |
| --- |

В качестве того, как это всё реализовать на железе, была выбрана среда виртуализации VirtualBox. Было решено использовать 2 машины Ubuntu-server 24.01 для клиента и сервера, но далее версию ОС мы сменили на 22 (об этом позже).

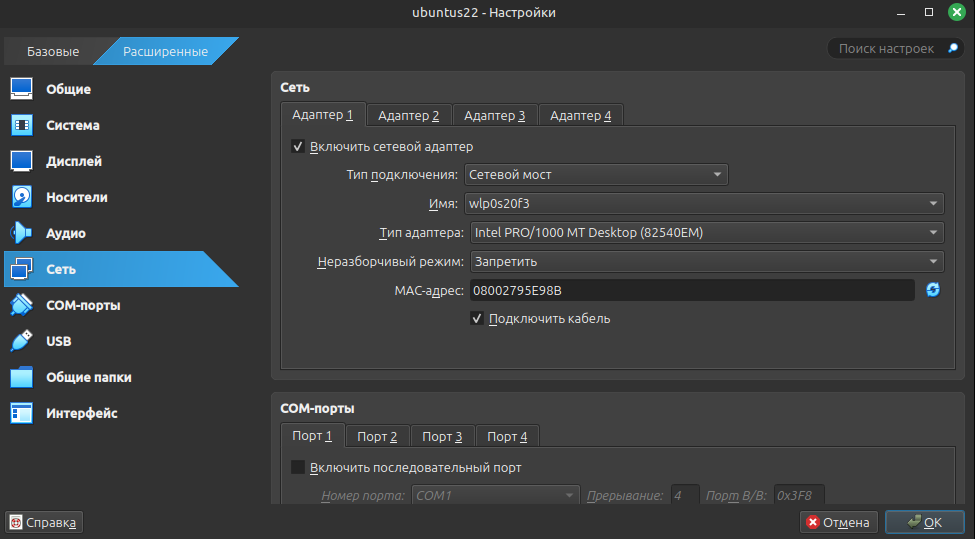
Сетевые интерфейсы настроены в режиме Сетевой мост на интерфейс, в нашем случае, подключаемый к Wi-Fi - это давало нам интернет-соединение для удобного обновления и пакет-менеджмента, а также DHCP Wi-Fi сети сразу выдавал адреса на машины. Также можно использовать внутреннюю сеть, но обязательным условием будет нахождение хотя бы одного интерфейса ВМ в одной сети. Вторые интерфейсы ВМ настраивали на разные внутренние сети, далее от их использования отказались. Не смотря на то, что был выбран вариант `pointopoint` изначально мы тестировали с имитацией частных сетей

Поднималось 2 ВМ с двумя сетевыми интерфейсами

- 1 сконфигурирован как bridge

- 2 сконфигурирован как внутренняя "частная" сеть

Однако потом от такого варианта отказались, так как впн точка точка и оставили только "внешние интерфейсы"



запуск, установка зависимостей, проверка работоспособности

| sudo apt update  sudo apt full-upgrade  sudo apt instal gcc make zlib1g zlib1g-dev flex yacc libssl-dev ifconfig -y  wget http://www.oberhumer.com/opensource/lzo/download/lzo-1.08.tar.gz  tar zxvf lzo-1.08.tar.gz  cd lzo-1.08/  ./configure  sudo make install  cd ..  wget https://sourceforge.net/projects/vtun/files/vtun/2.5/vtun-2.5.tar.gz  tar -xzvf vtun-2.5.tar.gz  cd vtun/  ./configure --with-ssl=/usr/include/openssl --with-ssl-lib=/usr/lib/x86\_64-linux-gnu --disable-lzo --disable-zlib --disable-shaper  sudo make install |
| --- |

Далее от библиотеки lzo (сжатие) отказались, чтобы фреймы были чище.

Server:

sudo vtund -s -f /home/server.conf

Client:

sudo vtund -p cobra 192.168.31.197 -f /home/client.conf

Одним из свидетельств того, что всё завелось и заработало, является появление в ip a настроенного интерфейса туннеля tun0.

Для более подробной информации можно посмотреть syslog.

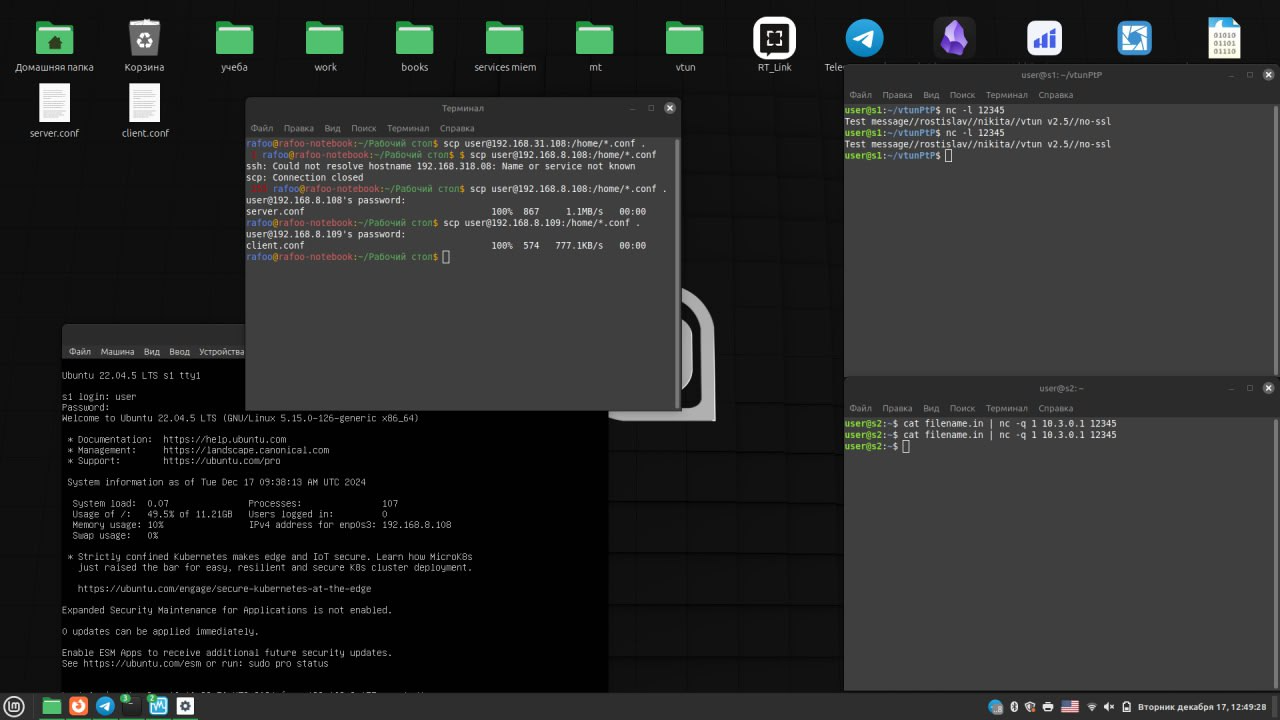
Проверка того, что пакеты доходят и корректно расшифровываются:

Client

cat filename.in | nc -q 1 10.3.0.1 12345

Server

nc -l 12345



Проверка того, что пакеты шифруются:

Client

cat filename.in | nc -q 1 10.3.0.1 12345

Server:

sudo tcpdump -i enp0s3 -nn -lex port 5344 -w dump

Достаем дамп из виртуальной машины и читаем с помощью Wireshark

Полезные команды:

tail -f /var/log/syslog - информация о работе vtund

killall vtund - убить процессы vtund

## 

## 

## Ошибки, фиксы, дополнительные настройки:

- pseudo tty

В более новых версиях Ubuntu (24.01) имеются изменения в конфигурации ядра и в модулях, которые влияют на работу VTun. Поскольку на 24 версии он не запускался, выдавая ошибку “pseudo tty”, было принято решение перейти на старую версию Ubuntu Server 22

- ifconfig не был установлен

По умолчанию ifconfig не предустановлен в US22, но он используется vtun. Об этом не написано на оф. сайте, также об ошибке не писало в логах, и только чудом в конфиге мы увидели путь к ifconfig и установили его.

- sysctl -w net.ipv4.ip\_forward=1

Команда sysctl -w net.ipv4.ip\_forward=1 используется для включения функции маршрутизации (IP forwarding) в ядре Linux. Эта функция позволяет вашему серверу пересылать IP-пакеты между сетевыми интерфейсами, что необходимо для настройки маршрутизаторов или VPN-серверов.

Также на сайте об этом ни слова.

## Внедрение алгоритма шифрования

В процессе выяснения какой конфиг всё таки подойдет, начал смотреть какие файлы отвечают за шифрование трафика.

Путём несложной команды `grep -rn vtun/ -e 'encrypt'` выснилось что за шифрование отвечает файл lfd\_encrypt

В нём есть 2 основных блока:

#ifdef HAVE\_SSL

тут шифрование с BLOWFISH + MD5, если при конфигурации vtun не был указан флаг –disable-ssl

#else /\* HAVE\_SSL \*/

тут просто ничего нету и вывод в syslog

#endif /\* HAVE\_SSL \*/

Проанализировав код, примерно было понятно как технически производится шифрование. Файл практически оказался независим от остальной части проекта, что помогло нам по итогу не разбираться во всём коде проекта. Никита решил для начала написать простой xor, чтобы удостовериться в правильности гипотез и посмотреть на поведение программы.

#define ENC\_BUF\_SIZE (VTUN\_FRAME\_SIZE + 16)

static char \*enc\_buf = NULL;

static unsigned char xor\_key = 0xAA;

int alloc\_encrypt(struct vtun\_host \*host)

{

// Выделяем буфер, как в оригинале

if(!(enc\_buf = lfd\_alloc(ENC\_BUF\_SIZE))){

syslog(LOG\_ERR,"Can't allocate buffer for encryptor");

return -1;

}

syslog(LOG\_INFO, "Simple XOR encryption initialized");

return 0;

}

int free\_encrypt()

{

lfd\_free(enc\_buf);

enc\_buf = NULL;

return 0;

}

int encrypt\_buf(int len, char \*in, char \*\*out)

{

int i;

for(i=0; i<len; i++){

enc\_buf[i] = in[i] ^ xor\_key;

}

\*out = enc\_buf;

return len;

}

int decrypt\_buf(int len, char \*in, char \*\*out)

{

int i;

// Дешифруем на месте

for(i=0; i<len; i++){

in[i] = in[i] ^ xor\_key;

}

\*out = in;

return len;

}

Ключ пока просто захардкодили и вывели в syslog.

Для тестирования использовались стандартные утилиты tcpdump и nc

Client

cat filename.in | nc -q 1 10.3.0.1 12345

Server:

sudo tcpdump -i enp0s3 -nn -lex port 5344 -w dump.pcap

Информация о формировании пакета:

Вы увидите, что vtun собирает пакеты (без сжатия) как:

ushort packet\_len (2 байта)

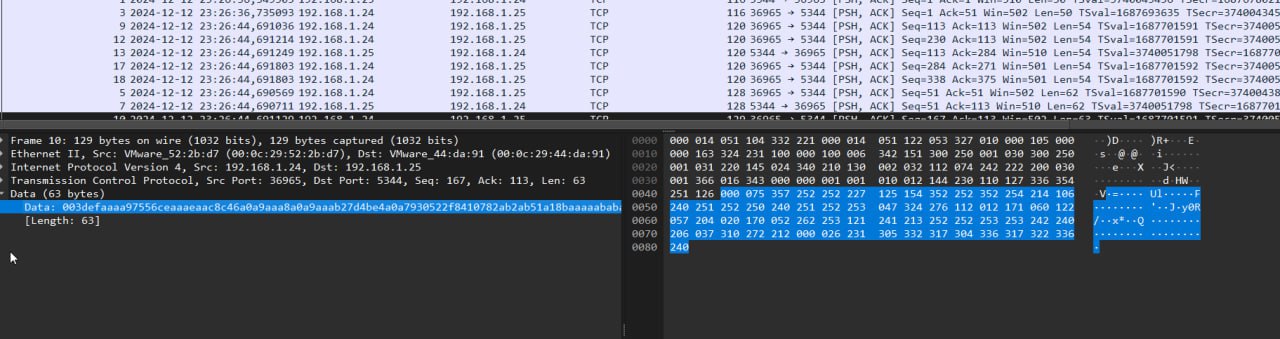
int padding (1-8 байт), причем первые байты определяют длину padding.

n1-байт полезной нагрузки

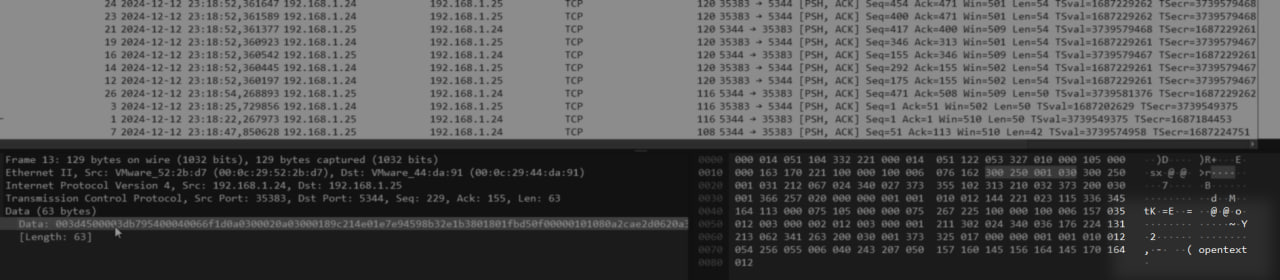
Таким образом, vtun добавляет минимум (1 (padding) + 2) = 3 байта и максимум (8(padding) + 2) = 10 байт накладных расходов. Если используется сжатие, то эти накладные расходы уменьшатся. Если шифрование не применяется, то слой vtun не будет добавлять байты подшивки, в результате чего накладные расходы составят всего два байта.

Файлы затем доставали из ВМ, анализировали через wireshark и xor проверялся руками. Методику тестирования разработал Ростислав.

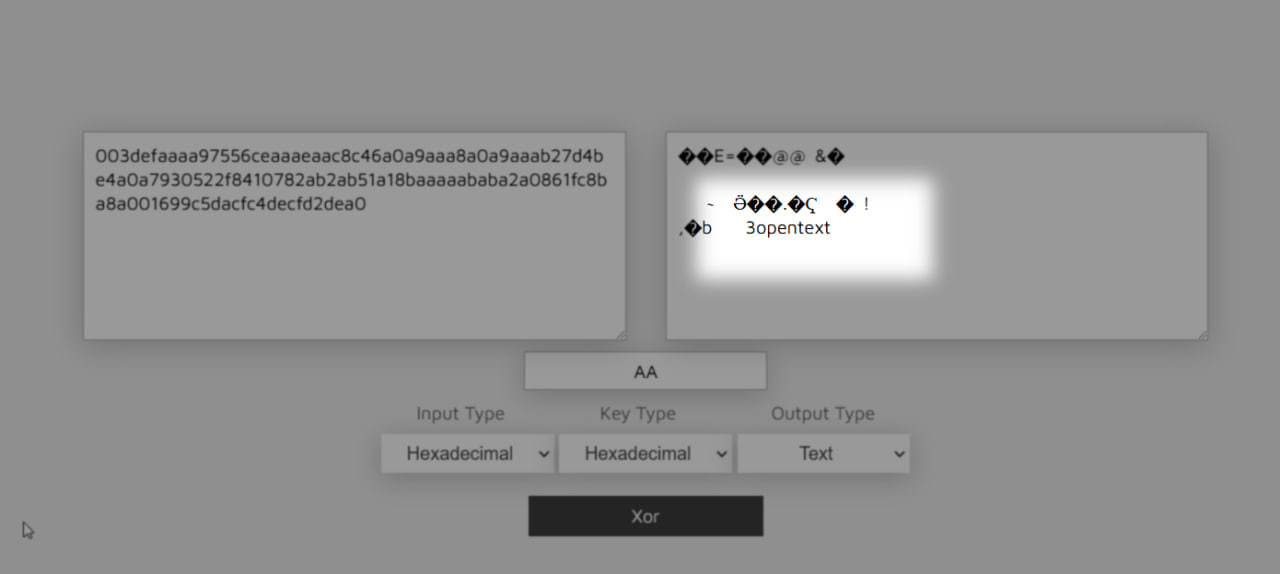
Пакет на интерфейсе



Пакет на tun0



Проверка операции XOR



Всё нормально. Бегает. Если интересует шифруется ли - то tcpdump -i \_имя\_интерфейса\_ -lex вам поможет. Вначале на интерфейсе tun0, а потом на внешнем. На tun будет видно, что это icmp (при пингах), а на внешнем это уже будет Unknown.

Далее выполняем замену XOR на Кузнечик в реализации библиотеки libakrypt.

Скачиваем и устанавливаем библиотеку libakrypt.

| git clone https://git.miem.hse.ru/axelkenzo/libakrypt-0.x  mkdir build  cd build/  sudo apt install libbz2-dev cmake  cmake ../libakrypt-0.x  sudo make  sudo make install |
| --- |

Заметили особенность, что после выполнения в папке vtun

./configure --disable-ssl --disable-lzo --disable-zlib --disable-shaper

формируется MakeFile, но чтобы в коде можно было использовать libacrypt, необходимо после конфигурации добавить в него следующие строки:

| LDFLAGS += -L/usr/local/lib -lakrypt  CFLAGS += -I/usr/local/include  LDFLAGS+= -Xlinker -rpath=/usr/local/include/  LIBS += -libakrypt |
| --- |

Далее было много работы с документацией, разбор примеров использования криптофункций, выбор режима шифрования. Работа делалась в режиме 50/50. Изначальный код был написан Ростиславом, Никита долго и упорно его дебажил и доводил до рабочего состояния.

Сначала написали с использованием ak\_bckey\_encrypt\_cbc

int alloc\_encrypt(struct vtun\_host \*host)

{

// Выделяем буфер для шифрования

if (!(enc\_buf = lfd\_alloc(ENC\_BUF\_SIZE))) {

syslog(LOG\_ERR, "Can't allocate buffer for encryptor");

return -1;

}

// Инициализация ключа "Кузнечик"

if (ak\_bckey\_create\_kuznechik(&kuznechik\_key) != ak\_error\_ok) {

syslog(LOG\_ERR, "Failed to create Kuznechik key");

lfd\_free(enc\_buf);

return -1;

}

// Устанавливаем ключ (например, все байты равны 0x01 для теста)

uint8\_t key[32];

memset(key, 0x01, sizeof(key));

if (ak\_bckey\_set\_key(&kuznechik\_key, key, sizeof(key)) != ak\_error\_ok) {

syslog(LOG\_ERR, "Failed to set Kuznechik key");

ak\_bckey\_destroy(&kuznechik\_key);

lfd\_free(enc\_buf);

return -1;

}

syslog(LOG\_INFO, "Kuznechik encryption initialized");

return 0;

}

int free\_encrypt()

{

ak\_bckey\_destroy(&kuznechik\_key);

lfd\_free(enc\_buf);

enc\_buf = NULL;

return 0;

}

int encrypt\_buf(int len, char \*in, char \*\*out)

{

// Шифрование с использованием режима CBC

if (ak\_bckey\_encrypt\_cbc(&kuznechik\_key, in, enc\_buf, len, iv, KUZNECHIK\_BLOCK\_SIZE) != ak\_error\_ok) {

syslog(LOG\_ERR, "Encryption failed");

return -1;

}

\*out = enc\_buf;

return len;

}

int decrypt\_buf(int len, char \*in, char \*\*out)

{

// Расшифровка с использованием режима CBC

if (ak\_bckey\_decrypt\_cbc(&kuznechik\_key, in, enc\_buf, len, iv, KUZNECHIK\_BLOCK\_SIZE) != ak\_error\_ok) {

syslog(LOG\_ERR, "Decryption failed");

return -1;

}

\*out = enc\_buf;

return len;

}

Но еще раз посмотрев на эталонный пример с BlowFish, решили взять режим электронной книги (ECB). Также были попытки (и успешные) внести изменения в файл auth.c, main.c, но в итоге от изменений отказались. Оставили аутентификацию прежней, а в main.c пытались добавить подключение libakrypt, что в итоге можно сделать и в файле шифрования.

Финальный код файла lfd\_encrypt.c:

#include "config.h"

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <syslog.h>

#include <strings.h>

#include <string.h>

#include "vtun.h"

#include "linkfd.h"

#include "lib.h"

#include <libakrypt.h>

#ifdef HAVE\_SSL

#include <md5.h>

#include <blowfish.h>

#define ENC\_BUF\_SIZE VTUN\_FRAME\_SIZE + 16

#define ENC\_KEY\_SIZE 16

BF\_KEY key;

char \* enc\_buf;

int alloc\_encrypt(struct vtun\_host \*host)

{

if( !(enc\_buf = lfd\_alloc(ENC\_BUF\_SIZE)) ){

syslog(LOG\_ERR,"Can't allocate buffer for encryptor");

return -1;

}

BF\_set\_key(&key, ENC\_KEY\_SIZE, MD5(host->passwd,strlen(host->passwd),NULL));

syslog(LOG\_INFO, "BlowFish encryption initialized");

return 0;

}

int free\_encrypt()

{

lfd\_free(enc\_buf); enc\_buf = NULL;

return 0;

}

int encrypt\_buf(int len, char \*in, char \*\*out)

{

register int pad, p;

register char \*in\_ptr = in, \*out\_ptr = enc\_buf;

/\* 8 - ( len % 8 ) \*/

pad = (~len & 0x07) + 1; p = 8 - pad;

memset(out\_ptr, 0, pad);

\*out\_ptr = (char) pad;

memcpy(out\_ptr + pad, in\_ptr, p);

BF\_ecb\_encrypt(out\_ptr, out\_ptr, &key, BF\_ENCRYPT);

out\_ptr += 8; in\_ptr += p;

len = len - p;

for (p=0; p < len; p += 8)

BF\_ecb\_encrypt(in\_ptr + p, out\_ptr + p, &key, BF\_ENCRYPT);

\*out = enc\_buf;

return len + 8;

}

int decrypt\_buf(int len, char \*in, char \*\*out)

{

register int p;

for (p = 0; p < len; p += 8)

BF\_ecb\_encrypt(in + p, in + p, &key, BF\_DECRYPT);

p = \*in;

if (p < 1 || p > 8) {

syslog(LOG\_INFO, "decrypt\_buf: bad pad length");

return 0;

}

\*out = in + p;

return len - p;

}

/\*

\* Module structure.

\*/

struct lfd\_mod lfd\_encrypt = {

"Encryptor",

alloc\_encrypt,

encrypt\_buf,

NULL,

decrypt\_buf,

NULL,

free\_encrypt,

NULL,

NULL

};

#else /\* HAVE\_SSL \*/

#define ENC\_BUF\_SIZE (VTUN\_FRAME\_SIZE + 16)

#define SALT\_SIZE 4 // "rand", 4 bytes

typedef struct {

struct bckey bkey; // bckey structure, not a pointer

char \*enc\_buf; // Encryption buffer

} encrypt\_context;

static encrypt\_context ctx = {0};

int alloc\_encrypt(struct vtun\_host \*host)

{

int error = ak\_error\_ok;

const char salt[SALT\_SIZE] = "rand"; // fixed salt

ctx.enc\_buf = lfd\_alloc(ENC\_BUF\_SIZE);

if (ctx.enc\_buf == NULL) {

syslog(LOG\_ERR, "Can't allocate buffer for encryptor");

return -1;

}

if (ak\_libakrypt\_create(NULL) != ak\_true) {

syslog(LOG\_ERR, "Failed to initialize libakrypt");

lfd\_free(ctx.enc\_buf);

ctx.enc\_buf = NULL;

return -1;

}

error = ak\_bckey\_create\_oid(&ctx.bkey, ak\_oid\_find\_by\_name("kuznechik"));

if (error != ak\_error\_ok) {

ak\_error\_message(error, \_\_func\_\_, "Failed to create block cipher key context");

ak\_libakrypt\_destroy();

lfd\_free(ctx.enc\_buf);

ctx.enc\_buf = NULL;

return -1;

}

error = ak\_bckey\_set\_key\_from\_password(&ctx.bkey,

host->passwd, strlen(host->passwd),

salt, SALT\_SIZE);

if (error != ak\_error\_ok) {

ak\_error\_message(error, \_\_func\_\_, "Failed to set key from password");

ak\_bckey\_destroy(&ctx.bkey);

ak\_libakrypt\_destroy();

lfd\_free(ctx.enc\_buf);

ctx.enc\_buf = NULL;

return -1;

}

syslog(LOG\_INFO, "libakrypt encryption initialized with Kuznechik cipher");

return 0;

}

int free\_encrypt()

{

if (ctx.enc\_buf) {

lfd\_free(ctx.enc\_buf);

ctx.enc\_buf = NULL;

}

ak\_bckey\_destroy(&ctx.bkey);

ak\_libakrypt\_destroy();

return 0;

}

int encrypt\_buf(int len, char \*in, char \*\*out)

{

int pad; // добавка до размера блока

size\_t total\_len;

char \*out\_ptr = ctx.enc\_buf;

int error;

pad = ctx.bkey.bsize - (len % ctx.bkey.bsize);

if (pad == 0) pad = ctx.bkey.bsize;

total\_len = len + pad;

if (total\_len > ENC\_BUF\_SIZE) {

syslog(LOG\_ERR, "Encrypted buffer size is insufficient");

return -1;

}

memset(out\_ptr, pad, pad);

memcpy(out\_ptr + pad, in, len);

error = ak\_bckey\_encrypt\_ecb(&ctx.bkey,

(const ak\_pointer)out\_ptr,

(ak\_pointer)out\_ptr,

total\_len);

if (error != ak\_error\_ok) {

ak\_error\_message(error, \_\_func\_\_, "Encryption failed");

return -1;

}

\*out = ctx.enc\_buf;

return total\_len;

}

int decrypt\_buf(int len, char \*in, char \*\*out)

{

int pad;

int error;

error = ak\_bckey\_decrypt\_ecb(&ctx.bkey,

(const ak\_pointer)in,

(ak\_pointer)in,

len);

if (error != ak\_error\_ok) {

ak\_error\_message(error, \_\_func\_\_, "Decryption failed");

return -1;

}

pad = (unsigned char)\*in;

if (pad < 1 || pad > ctx.bkey.bsize) {

syslog(LOG\_INFO, "decrypt\_buf: bad pad length");

return 0;

}

\*out = in + pad;

return len - pad;

}

struct lfd\_mod lfd\_encrypt = {

"libakrypt Encryptor",

alloc\_encrypt,

encrypt\_buf,

NULL, // No additional initialization

decrypt\_buf,

NULL, // No additional processing

free\_encrypt,

NULL, // No additional cleanup

NULL // No additional functions

};

#endif /\* HAVE\_SSL \*/

Финальные конфиги:

server.conf

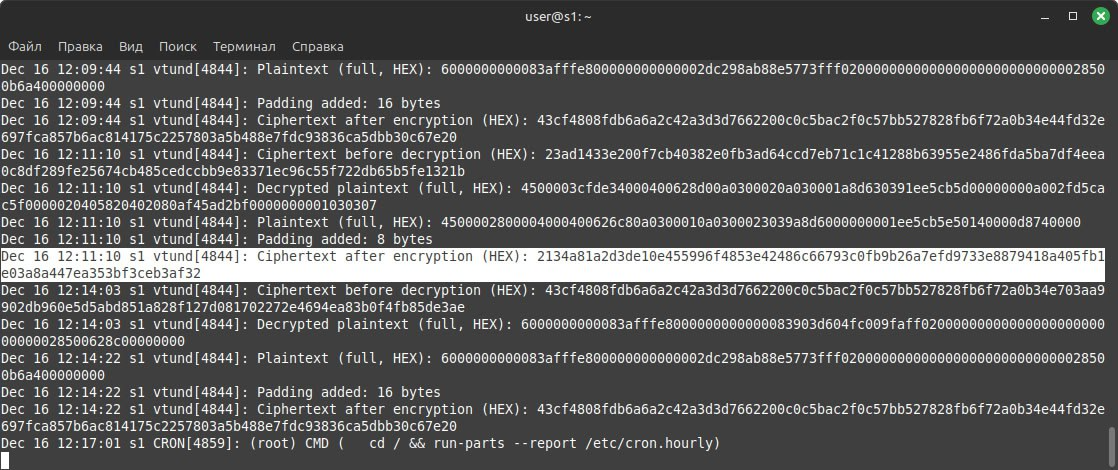
| #server  options {  port 5344; # Listen on this port.  # Syslog facility  syslog daemon;  # Path to various programs  ppp /usr/sbin/pppd;  ifconfig /sbin/ifconfig;  route /sbin/route;  firewall /sbin/ipchains;  ip /sbin/ip;  }  # Default session options  default {  compress no; # Compression is off by default  speed 0; # By default maximum speed, NO shaping  }  # TUN example. Session 'cobra'.  cobra {  pass test;  type tun; # IP tunnel  proto udp; # UDP protocol  comp no; # LZO compression level 9  encrypt yes; # Encryption  keepalive yes; # Keep connection alive  up {  # Connection is Up  # 10.3.0.1 - local, 10.3.0.2 - remote  ifconfig "%% 10.3.0.1 pointopoint 10.3.0.2 mtu 1450";  };  } |
| --- |

client.conf

| #client  options {  port 5344; # Connect to this port.  timeout 60; # General timeout  syslog daemon;  # Path to various programs  ppp /usr/sbin/pppd;  ifconfig /sbin/ifconfig;  route /sbin/route;  firewall /sbin/ipchains;  ip /sbin/ip;  }  # TUN example. Session 'cobra'.  cobra {  password test;  type tun; # Device tun1  persist yes;  encrypt yes;  up {  # Connection is Up  # Assign IP addresses.  ifconfig "%% 10.3.0.2 pointopoint 10.3.0.1 mtu 1450";  };  } |
| --- |

Пример работы:

tail -f /var/log/syslog

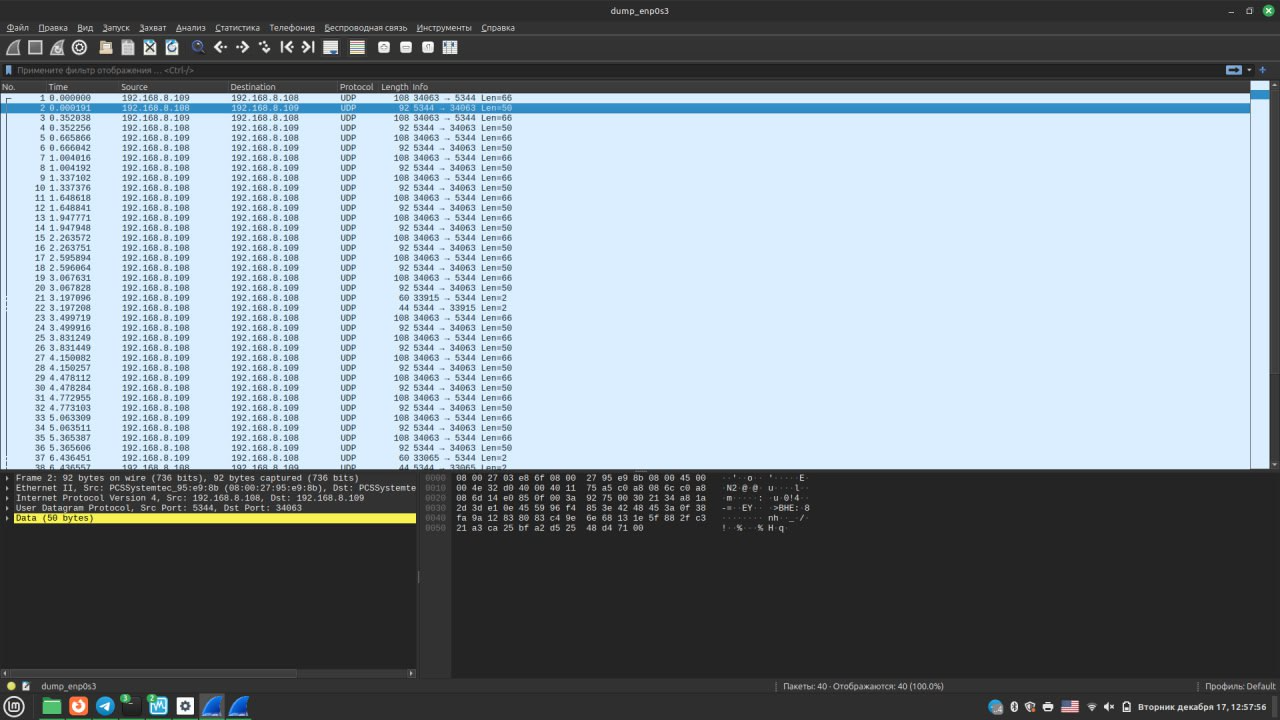


Client:

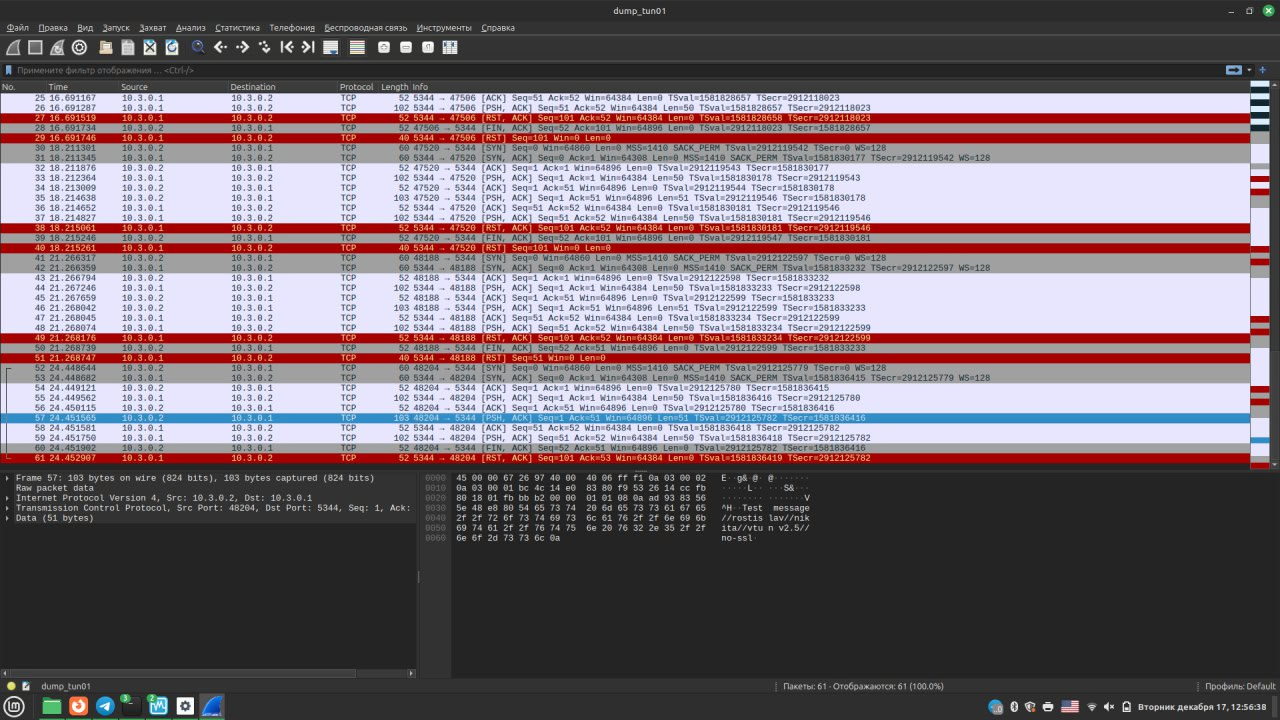
cat filename.in | nc -q 1 10.3.0.1 5344

Server:

sudo tcpdump -i enp0s3 -nn -lex port 5344 -w dump\_enp0s3



sudo tcpdump -i tun0 -nn -lex port 5344 -w dump\_tun0



Логи корректной работы:

| Dec 12 09:47:39 s1 vtund[2706]: Session cobra[192.168.8.109:33065] opened  Dec 12 09:47:39 s1 vtund[2706]: UDP connection initialized  Dec 12 09:47:39 s1 vtund[2706]: libakrypt encryption initialized with Kuznechik cipher  Dec 12 09:47:39 s1 systemd-networkd[627]: tun1: Link UP  Dec 12 09:47:39 s1 systemd-networkd[627]: tun1: Gained carrier  Dec 12 09:47:39 s1 systemd-networkd[627]: tun1: Gained IPv6LL |
| --- |

## Ошибки, фиксы, дополнительные настройки:

Основные проблемы решались путем добавления сообщений в syslog, чтением и пониманием кода, чтением малочисленных примеров использования vtun, изучением использования библиотеки libakrypt и письмам разработчикам (ответа, на которые, мы, к сожалению, не получили)

- make clean

Иногда make cache забивался, и перед make install при пересборке vtun (что происходило очень часто, для этого написали небольшой скрипт) необходимо было его почистить, тк make install не отрабатывал корректно.

- sudo ldconfig

При сборке libakrypt и запуске vtun были ошибки типа error while loading shared libraries, а именно ругался на /usr/local/lib/libakrypt.so.0.9.17. Видимо какие то файлы не прилинковались к друг другу.

## Дальнейшее возможное улучшение

* configure

Добавить в configure поддержку режима шифрования кузнечик, как это реализовано с openssl BlowFish, чтобы пользователь просто указывал ./configure –enable-kuznechik … и формировался корректный Makefile

* config.h

#define HAVE\_LIBAKRYPT 1 - добавить ключ, который при конфигурации вёл бы в отдельную секцию файла lfd\_encrypt.c с реализованным Кузнечиком. Так бы мы добились поддержки как BlowFish, так и кузнечика одновременно, в зависимости от конфигурации.