Скрытые Каналы. Лабораторная работа #3.

Соколов А.Д. Б20-505

Вариант 7

Модель 4. Пакеты случайной длины передаются в случайные моменты времени.

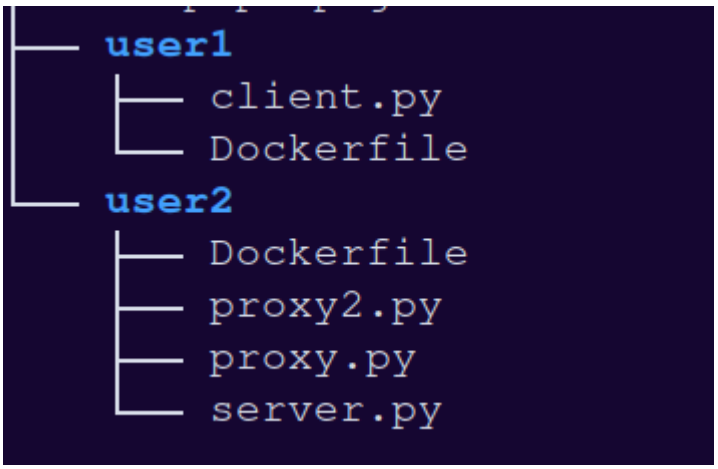
Пример 9. Time Relay Covert Channel

Исходя из измерения возможных значений задержек между пакетами в открытом
трафике, определяется медианное значение этого множества, и при передаче скрытого
сообщения для кодирования 0 выбирается значение межпакетного интервала из
подмножества значений, которые больше медианного, а для 1 — которые меньше

Возможности закладки:

- Буферизация трафика
- Генерация фиктивного трафика

Архитектура



Есть сокет сервера, который непрерывно слушает.
Есть сокет закладки, которая непрерывно слушает и делает свои дела с полученными пакетами.
Есть сокет защиты, который непрерывно слушает и защищает
Есть клиент - сокет который отправляет n пакетов на порт закладки.

Реализация

client.py

```python
import argparse
import socket
from os import urandom
from random import randint, random
from time import sleep


def model_packets(host, port, n_packets=200):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((host, port))

    sent = 0
    while sent < n_packets:
        data = urandom(randint(1, 100))
        data = len(data).to_bytes(4, "big") + data
        client.send(data)

        sent += 1
        sleep(random())
    client.close()


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-p", "--port", help="server running port", type=int, required=True
    )
    parser.add_argument("--host", help="server running host", type=str, required=True)
    args = parser.parse_args()

    model_packets(args.host, args.port)
```

proxy.py

```python
import socket

from logging import error
import argparse

from time import sleep
from random import random, randint
from os import urandom

from queue import Queue
from threading import Thread

from termcolor import colored

# from concurrent.futures import ThreadPoolExecutor

M = 0.48871803283691406

def start_proxy(port: int, server_host: str, server_port: int, message: str):
    q = Queue()
    msg = int.from_bytes(message.encode(), "big")
    finished = False

    def listen_and_buffer(port):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(("localhost", port))
        sock.listen(1)

        client, addr = sock.accept()
        error(colored(f"PROXY::Connection from {addr}", "red"))

        while True:
            data = client.recv(1024)
            error(colored(f"PROXY::Received data {data}", "red"))
            if not data:
                break
            q.put(data)
        client.close()

        global finished
        finished = True

```

```
    def be_proxy_and_send_message(host: str, port: port, msg):
        data = b""
        delay = 0.0
        started = False
        server = None
        while msg or not finished:
            print(finished)
            if msg and started:
                if msg & 1:
                    delay = 0.1 * random() + 0.2 + M
                else:
                    delay = random() * 0.9 * M
                msg >>= 1

                if q.empty():
                    data = urandom(randint(1, 100))
                    data = len(data).to_bytes(4, "big") + data
                else:
                    data = q.get()

                error(colored(f"PROXY::sending data {data}", "blue"))
                sleep(delay)
                server.send(data)
            elif not q.empty():
                if started:
                    data = q.get()
                    server.send(data)
                else:
                    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    error(colored(f"PROXY::trying {host}:{port}", "blue"))
                    server.connect((host, port))

                    started = True

        server.close()

    finished = False
    receive = Thread(target=listen_and_buffer, args=(port,))
    receive.start()
    send = Thread(
        target=be_proxy_and_send_message, args=(server_host, server_port, msg)
    )
    send.start()
```

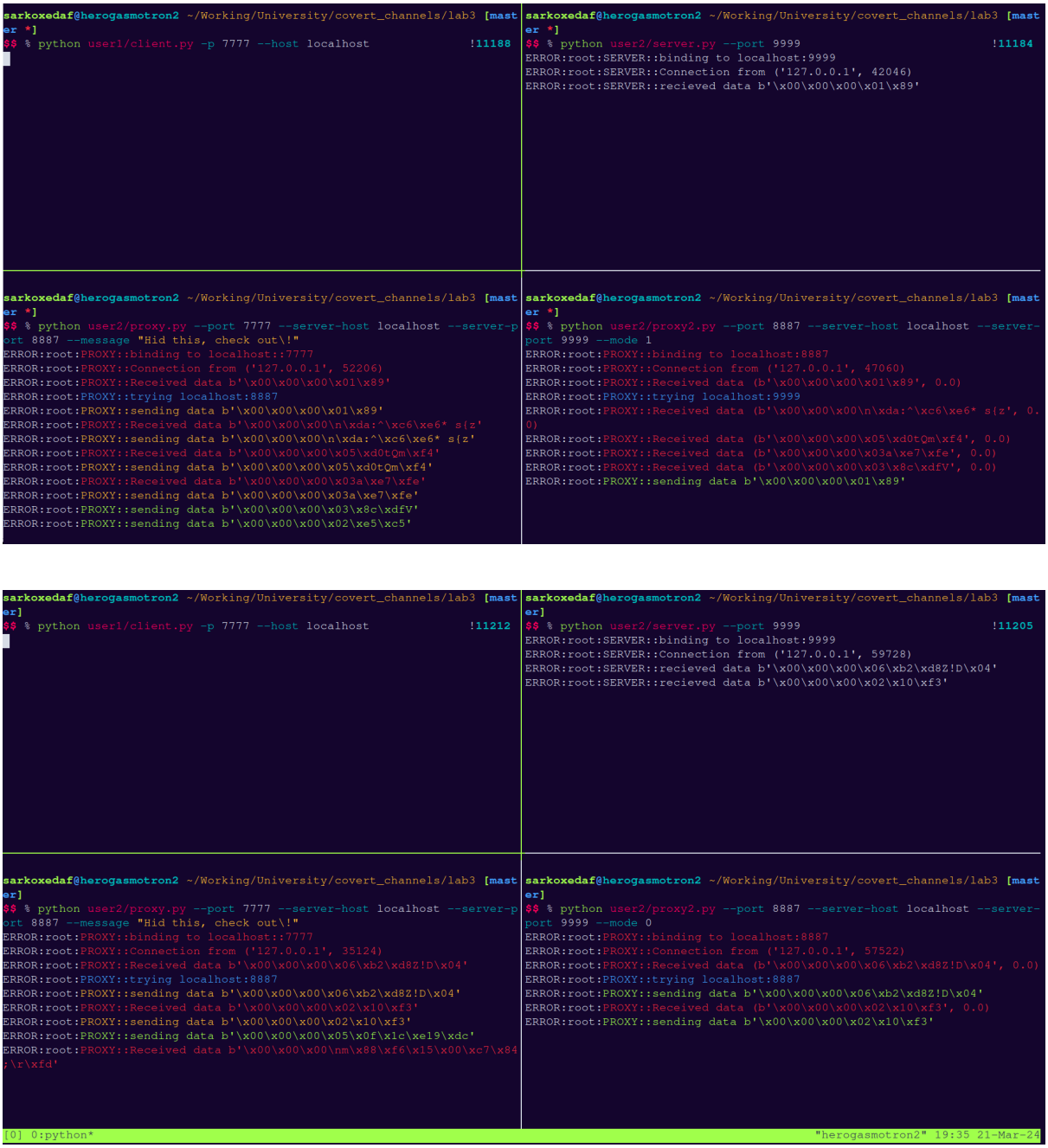
```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-p", "--port", help="proxy running port", type=int, required=True
    )
    parser.add_argument(
        "--server-port", help="server running port", type=int, required=True
    )
    parser.add_argument(
        "--server-host", help="server running host", type=str, required=True
    )
    parser.add_argument(
        "-m", "--message", help="secret message", type=str, required=True
    )
    args = parser.parse_args()

    start_proxy(args.port, args.server_host, args.server_port, args.message)
```

proxy2.py

```python
M = 1.0


def start_proxy(port: int, server_host: str, server_port: int, mode: bool):
    q = Queue()

    def listen_and_buffer(port):
        error(colored(f"PROXY::binding to localhost:{port}", "red"))
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(("localhost", port))
        sock.listen(1)

        client, addr = sock.accept()
        error(colored(f"PROXY::Connection from {addr}", "red"))

        start = None
        while True:
            data = client.recv(1024)
            end = time()
            if start is None:
                data = (data, 0.0)
            else:
                data = (data, end - start)
                end = start

            error(colored(f"PROXY::Received data {data}", "red"))
            q.put(data)
            if not data[0]:
                break
        client.close()

    receive = Thread(target=listen_and_buffer, args=(port,))
    receive.start()

```

```
    if mode:
        while True:
            if not q.empty():
                server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                error(colored(f"PROXY::trying {server_host}:{server_port}", "blue"))
                server.connect((server_host, server_port))

                while True:
                    if q.empty():
                        continue

                    data, delay = q.get()
                    if data == b"":
                        break

                    sleep(M - delay)
                    error(colored(f"PROXY::sending data {data}", "green"))
                    server.send(data)

                server.close()
    else:
        while True:
            if not q.empty():
                server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                error(colored(f"PROXY::trying {server_host}:{port}", "blue"))
                server.connect((server_host, server_port))

                while True:
                    if q.empty():
                        continue
                    data, _ = q.get()

                    if data == b"":
                        break

                    error(colored(f"PROXY::sending data {data}", "green"))
                    server.send(data)
                    if random() < 0.2:
                        data = urandom(randint(1, 10))
                        data = len(data).to_bytes(4, "big") + data

                        sleep(random())
                        error(colored(f"PROXY::sending data {data}", "white"))
                        server.send(data)

                server.close()

```

Демонстрация работы закладки и защиты



Демонстрация работы злоумышленника

Нормализация

```
, 1.0002222061157227, 1.00020432472229, 1.000230073928833, 1.000326395034
79, 1.001526117324829, 1.002342700958252, 1.0001840591430664, 1.000221729
2785645, 1.0002331733703613, 1.0002286434173584, 1.0002477169036865, 1.00
06024837493896, 1.0001778602600098, 1.0002410411834717, 1.000221967697143
6, 1.0002264976501465, 1.000230312347412, 1.00193190574646, 1.00085997581
48193, 1.000464677810669, 1.0000851154327393, 1.000291109085083, 1.000201
4636993408, 1.0002596378326416, 1.0002098083496094, 2.6226043701171875e-0
5]
ERROR:root:Recovered message: b'?\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff\xff\xff\xff\xff'
```

Фиктивные пакеты:

```
43800354004, 0.762732744216919, 0.36688804626464844, 0.7343463897705078,
0.555079460144043, 0.1961348056793213, 0.7109658718109131, 0.215986013412
47559, 0.029045820236206055, 0.10543298721313477, 0.37146997451782227, 0.
38469600677490234, 0.2131969928741455, 0.3140983581542969, 0.786209583282
4707, 0.25713038444519043, 0.11668133735656738, 0.06682085990905762, 0.16
183876991271973, 0.3476269245147705, 0.34636926651000977, 0.7791616916656
494, 0.009315013885498047, 0.03346824645996094, 0.7423806190490723, 0.761
2903118133545, 0.38947391510009766, 0.7275714874267578, 0.944987773895263
7, 2.47955322265625e-05, 1.7642974853515625e-05, 0.24085474014282227, 2.5
033950805664062e-05, 0.03184056282043457, 0.759533166885376, 0.7705786228
179932, 0.25400829315185547, 0.0447080135345459, 0.06457138061523438, 0.0
2575516700744629, 0.7168200016021729, 0.26305413246154785, 0.617027044296
2646, 2.47955322265625e-05, 0.47793102264404297, 0.001234292984008789]
ERROR:root:Recovered message: b'\x01C\x06\xc8\x10\x16\x8cF\x8b\x99P\x80\x
9aXFi3D\x81\xbd\xc6\xe8\x10'
```

Выводы

Нормализация

Пропускная способность скрытого канала падает, так как от среднего в пол секунды промежутка мы переходим к одной секунде всегда.

Аналогично с основным каналом, так как там накладывается еще и задержка, создаваемая закладкой и фиктивный трафик.

Однако схема полностью справилась с противодействием скрытому каналу, так как теперь медианные значения всегда будут давать 1

Фиктивный трафик

Пропускная способность скрытого канала изменяется, так как мы добавляем случайные задержки и дополнительные пакеты, которые сильно влияют на возможность декодирования.

Пропускная способность основного канала изменяется значительно, так как у нас появляется уже удвоенный фиктивный трафик и задержки.

Схема полностью справилась с противодействием скрытому каналу, так как теперь у нас не сохраняется как последовательность бит, так и временные отрезки, необходимые для декодирования.