

Graficzny interfejs manipulatora pakietów

[Graficzny interfejs manipulatora pakietów](#)

[Wprowadzenie](#)

[Cel](#)

[Motywacja](#)

[Droga rozwoju](#)

[Narzędzia i technologie](#)

[Harmonogram](#)

[Opis projektu](#)

[Protokoły](#)

[IPTables](#)

[Scapy](#)

[PyQt](#)

[Schemat działania](#)

[Qt](#)

[TCP](#)

[UDP](#)

[Instrukcja obsługi](#)

[Cheat sheet](#)

Wprowadzenie

Projekt realizowany w ramach zajęć “Podstawy Telekomunikacji” pod nadzorem mgr Przemysława Walkowiaka.

Cel

Celem projektu było utworzenie interfejsu, który umożliwi edycje przechwyconych pakietów oraz ich odesłanie w spreparowanej formie. Domyślnym systemem, pod którym interfejs graficzny był testowany to Kali Linux. Z racji wykorzystywanych technologii, używanie programu pod systemem Windows jest niemożliwe - IPTables nie występuje na systemach rodziny Windows. Dodatkowo obsługę IPTables oparliśmy na wywoływaniu funkcji z powłoki systemu.

Motywacja

Temat został wybrany ze względu na chęci rozszerzenia umiejętności programowania w języku Python oraz rozszerzenia znajomości systemu Linux. Dodatkowo dał nam możliwości analizy oraz

nauki wykorzystywanych protokołów takich jak TCP, UDP, ICMP, IP w sposób praktycznie i namacalny.

Droga rozwoju

Program, który zdajemy w pełni obsługuje protokoły UDP, TCP, IP oraz ICMP. W przyszłości można zaimplementować obsługę również innych protokołów takich jak Ethernet, ARP, oraz wielu innych, które są bardzo popularne. Dodatkowo w programie zostały zawarte pewne szablony odpowiedzi. Na przykład możemy odpowiedzieć automatycznie na zapytanie TCP - "Three way handshake", które może nas połączyć chociażby z gniazdkiem TCP klienta, który się łączy. Jest bardzo dużo różnych wzorców, które mogłyby być rozpoznane oraz zaimplementowane - dzięki temu można pomijać pakiety, które nie są dla nas istotne, a są standardem funkcjonowania danego protokołu. Dzięki takim funkcjom program nadawałby się doskonale do analizy programów oraz systemów. Ciekawą funkcjonalnością również byłoby wsparcie pełnego fuzzingu pakietów.

Narzędzia i technologie

Scapy - Jest to interaktywne narzędzie do manipulacji pakietami. Za jego pomocą można fałszować, dekodować pakiety z wielu protokołów, wysyłać je, przechwytywać, dopasowywać wymagania i wysyłać pakiet zwrotny. Scapy jest napisany w języku Python, najlepiej funkcjonuje w wersji 2.7. Strona internetowa narzędzia - <https://github.com/phaethon/scapy>.

Python - Jest to wysokopoziomowy, dynamiczny język programowania. Typy w pythonie są dynamiczne, zaś pamięć jest zarządzana automatycznie. Zyskał popularność dzięki rozbudowanym bibliotekom standardowym oraz prostocie. Strona internetowa języka - <https://www.python.org/>.

PyQt - Wieloplatformowe, bardzo rozbudowane narzędzie, które pozwoliło nam m.in. na budowę interfejsu graficznego programu. PyQt.Gui - przestrzeń nazw, w której znajdują się wszystkie interesujące nas komponenty. Zawiera szereg klas, które obsługują tworzenie okien, przycisków, suwaków, pól tekstowych, tabel i wielu innych. PyQt zawiera również szereg klas związanych z gniazdkami, plikami, etc. Jednakże nie było nam to potrzebne. Opis platformy programistycznej - <https://wiki.python.org/moin/PyQt>.

PyCharm - IDE pod którym tworzyliśmy projekt, w wersji darmowej "Community". Wybrane ze względu na przyjemny interfejs, uzupełnianie kodu, podkreślanie błędów i wiele innych funkcji wspomagających produktywność. Strona IDE - <https://www.jetbrains.com/pycharm/>.

IPTables - Jest to program sterujący filtrem pakietów. Wymaga jądra skompilowanego z ip_tables. Funkcją wykorzystywaną była NFQUEUE, która razem ze Scapy tworzy moduł dzięki któremu można edytować pakiety. Może być również stosowany jako zaporę ogniową systemu. Dla nas służył jako filtr - każdy pakiet spełniający zdefiniowaną regułę trafiał do NFQUEUE. Artykuł opisujący użycie nfqueue, iptables oraz scapy - <https://5d4a.wordpress.com/2011/08/25/having-fun-with-nfqueue-and-scapy/>.

C++ Qt - Jest to odpowiednik silnika Qt dla języka C++. W nim napisana została prosta aplikacja, która korzysta z gniazdek TCP oraz UDP. Dzięki niej mogliśmy obserwować zmieniające się pakiety edytowane przez aplikację manipulatora.

Qt Creator - IDE dla silnika Qt. Przejrzyste narzędzie, która usprawnia pracę, dodatkowo wsparte przez debugger oraz kompilator Visual Studio 2015.

Harmonogram

Poniżej w tabeli znajduje się opis prac wykonanych w trakcie powstawania projektu. Prace nie są rozdzielone na osobę - każda z funkcjonalności była opracowywana wspólnie (mniej lub bardziej).

Zawartość tabeli potwierdza historia projektu na GitHub,
<https://github.com/bojakowsky/OnTheFlyPacketManipulator>.

Data	Zadanie
30.03	Prezentacja, praca wejścia
13.04	Utworzenie projektu, repozytorium, prostego GUI, skupienie się na obsłudze IPTables z kodu
27.04	Rozbudowa GUI - dodanie okna pozwalającego na dodawanie reguł dla IPTables, prosta tabela, podstawowa obsługa IPTables z kodu (wywołania na podstawie pól)
11.05	Pełne funkcjonowanie formy dodawania reguł dla protokołów ICMP, TCP oraz UDP, jak również obsługa ich z poziomu menadżera pakietów, testowanie oraz nauka obsługi otrzymywanych pakietów
25.05	Dodanie funkcji odzyskiwania reguł po wyłączeniu programu, rozdzielenie aplikacji okienkowej oraz menadżera pakietów na dwa osobne wątki, wypełnianie wstępne tabeli danymi pobranymi przez menadżera pakietów na podstawie zasobów współdzielonych, refaktoryzacja modułów
08.06	Dodanie automatycznego odświeżania tabeli na podstawie zasobów współdzielonych, dodanie przycisków usuwania elementów tabeli, wszystkich elementów tabeli. Dodaliśmy przycisk automatycznej odpowiedzi oraz przycisk odpowiedzi typu "Fuzz" (losowe dane są generowane), na dwuklik dodaliśmy formę edycji pakietu wraz z opcjami automatyzacji podmiany danych
po 08.06	Dodanie aplikacji w Qt C++ z obsługą gniazdek UDP i TCP, testowanie podmiany pakietów, poprawki

Opis projektu

Aby w pełni zrozumieć działanie programu poniżej opisane są poszczególne części projektu, które składają się na całość. Począwszy od obsługiwanych protokołów, po strukturę projektu.

Protokoły

W projekcie dodana została obsługa trzech protokołów - ICMP, UDP oraz TCP. Nie oznacza to jednak, że nie możemy przechwytywać innych pakietów. Oznacza to, że dodane zostały funkcje pozwalające na usprawnienie pracy z tymi protokołami.

Dla protokołu TCP usprawnieniem jest opcja automatycznego odsyłania pakietu, z wyszczególnieniem dwóch przypadków:

1. na przychodzącym pakiecie jest ustawiona flaga SYN - Three way handshake
2. dla każdego innego pakietu

Kod zaprezentowany na rys 1. przedstawia w jaki sposób odsyłane są automatycznie pakiety z flagą SYN. Adres IP wysyłającego jest zamieniany z odbiorcą, następnie porty są zamieniane. Na samym końcu wartość pola ACK wysyłającego jest ustawiona jako wartość pola odbiorcy SEQ, zaś wartość pola SEQ + 1 wysyłającego jest ustawiona na wartość pola ACK odbiorcy. Flaga jest ustawiana na wartość 'SA', co oznacza SYN-ACK, co potwierdza przyjęcie połączenia.

```
ip = IP()
tcp = TCP()
ip.src = pkt[IP].dst
ip.dst = pkt[IP].src

tcp.sport = pkt[TCP].dport
tcp.dport = pkt[TCP].sport

tcp.ack = pkt[TCP].seq + 1
tcp.seq = pkt[TCP].ack
tcp.flags = flag
send(ip / tcp)
```

Rys 1.: Three way handshake, protokół TCP.

W przeciwnym przypadku, przypadek 2. z listy, nie musimy podnosić numeru sekwencji, oraz dodawany jest "payload" do pakietu, tzn. warstwa bajtów. Zaprezentowany na rys. 2.

Flaga domyślnie jest ustawiana na wartość 'PA' - PSH-ACK. Co zazwyczaj spowoduje odesłanie pakietu.

```

ip = IP()
tcp = TCP()
ip.src = pkt[IP].dst
ip.dst = pkt[IP].src
tcp.ack = pkt[TCP].seq
tcp.seq = pkt[TCP].ack
tcp.sport = pkt[TCP].dport
tcp.dport = pkt[TCP].sport
tcp.flags = flag
data = pkt[TCP].payload
send(ip / tcp / data)

```

Rys 2.: Pozostałe pakiety TCP, bez flagi SYN.

Istnieje też opcja wysyłania pakietów automatycznie generowanych - fuzz. Służy to do testów automatycznych, zaimplementowany dla protokołów TCP, UDP oraz ICMP, przykład dla protokołu TCP na rys. 3. Uwagę przykuwa funkcja fuzz()

```

ip = IP()
tcp = TCP()
ip.src = pkt[IP].dst
ip.dst = pkt[IP].src
tcp.sport = pkt[TCP].dport
tcp.dport = pkt[TCP].sport
send(ip/fuzz(tcp))

```

Rys. 3.: Przykład testu typu fuzz na pakiecie TCP.

Dla pakietów UDP sytuacja jest uproszczona - sytuacja automatycznego odsyłania pakietu przedstawiona na rys. 4. Dla wysyłanego pakietu adres dostawcy oraz odbiorcy są zamieniane. Port docelowy jest zwiększany o 1 (na potrzeby analizy z klientem napisanym w Qt C++), port źródła zostaje niezmieniony. Dodatkowo przepisywany jest payload.

```

ip = IP()
udp = UDP()
ip.src = pkt[IP].dst
ip.dst = pkt[IP].src
udp.sport = pkt[UDP].sport
udp.dport = pkt[UDP].dport+1
print(pkt[UDP].payload)
data = pkt[UDP].payload
send(ip / udp / data)

```

Rys. 4.: Automatyczna obsługa pakietu UDP.

W przypadku pakietu ICMP całość zaprezentowana jest na rys. 5. Interesującymi polami są code oraz type. Type oznacza typ pakietu ICMP, wiele z tych typów posiada własne kody, dokładna rozpiska (RFC 792) - <http://www.nthelp.com/icmp.html>.

```
ip = IP()
icmp = ICMP()
ip.src = pkt[IP].dst
ip.dst = pkt[IP].src
icmp.type = 0
icmp.code = 0
icmp.id = pkt[ICMP].id
icmp.seq = pkt[ICMP].seq
data = pkt[ICMP].payload
send(ip / icmp / data, verbose=0)
```

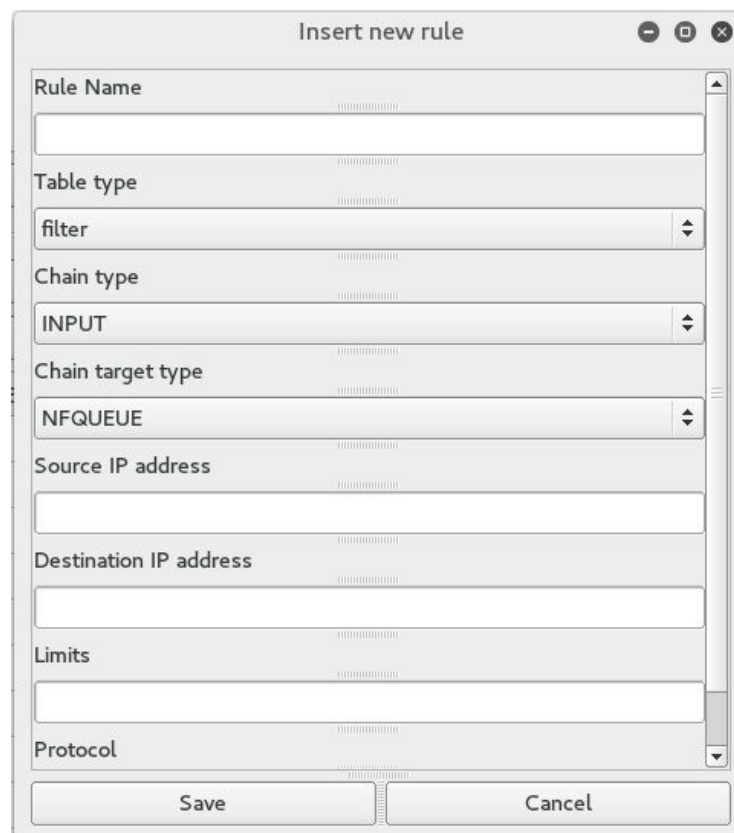
Rys. 5.: Automatyczna obsługa pakietu ICMP.

IPTables

Sterowanie filtrami pakietów umożliwia nam umieszczenie odpowiedniego pakietu w kolejce NFQUEUE, z której korzysta biblioteka Scapy.

Aby wiedzieć jak korzystać z programu należy dokładnie rozumieć użycie programu IPTables. Ponieważ dane wprowadzane do formy są wywoływane z poziomu powłoki za pomocą właśnie tego narzędzia. Dokładna dokumentacja - <http://linux.die.net/man/8/iptables>.

Jednakże krótko opiszę poszczególne komponenty, które zostały zaimplementowane w formie dodawania reguł. Na rys. 6. zaprezentowana jest forma dodawania reguł.



Rys. 6.: Forma dodawania reguł

Spoglądając na rys. 6. szybki opis pól i ich znaczenie:

1. Table type - typ tabeli
 - a. filter - domyślna tabela, gdzie trafiają reguły
 - b. nat - tabela dla pakietów ustanawiających połączenia
 - c. mangle - tabela dla wyspecjalizowanych zmian w pakietach
 - d. raw - tabela z najwyższym priorytetem, pakiety trafiają tutaj od razu po przesłaniu
2. Chain type - Typ łańcucha, w którym mogą znaleźć się pakiety, różne typy są dostępne dla różnych typów tabel
 - a. input - wywoływana dla nadchodzących pakietów przeznaczonych dla lokalnej maszyny - dla typów tabeli filter, mangle
 - b. forward - wywoływana dla pakietów tworzonych lokalnie, przeznaczonych dla pakietów opuszczających lokalną maszynę - dla typów tabeli filter, mangle

- c. output - wywoływana dla pakietów, które są trasowane przez lokalną maszynę, ale nie są dla niej przeznaczone - dla typów tabeli filter, nat, mangle, raw
 - d. prerouting - wywoływane dla pakietów z zewnątrz przed tym jak rozpoczną być trasowane - dla typów tabeli nat, mangle, raw
 - e. postrouting - dla pakietów opuszczających lokalną maszynę, po trasowaniu - dla typów tabeli nat, mangle
3. Chain target - można określić to jako politykę określonego łańcucha (co się z nim stanie)
 - a. accept - akceptujemy przyjęcie pakietu, nie jest przetwarzany przez kod programu
 - b. drop - porzucamy przyjęcie pakietu, nie jest przetwarzany przez kod programu - wysyłający nie jest informowany
 - c. reject - odrzucamy przyjęcie pakietu, nie jest przetwarzany przez kod programu - wysyłający jest informowany
 - d. nfqueue - pakiet trafia do kolejki nfqueue, utworzonej za pomocą Scapy - my decydujemy co się z nim stanie
 4. Source address - adres IPv4, z którego przyjdzie pakiet
 5. Destination address - adres IPv4, do którego trafi pakiet
 6. Limits - określa liczbę pakietów jaka zostanie przyjęta w pewnym czasie np. 1/s (1 na sekundę), 5/h (5 na godzinę), 3/m (3 na minutę)
 7. Protocol - filtr protokołu dla reguły
 - a. None - niesprecyzowany, czyli wszystkie protokoły
 - b. ICMP - tylko protokół ICMP
 - c. UDP - tylko protokół UDP
 - d. TCP - tylko protokół TCP

Następnie w zależności od wybranego protokołu, pojawiają się pola nowe pola do wypełnienia.

Prezentuje się to następująco:

1. None - brak nowych pól
2. UDP
 - a. Source port - port źródłowy datagramu
 - b. Destination port - port docelowy datagramu
3. ICMP
 - a. ICMP type - wspomniany wcześniej w opisie protokołu (RFC 792)
4. TCP
 - a. Source port - port źródłowy datagramu
 - b. Destination port - port docelowy datagramu
 - c. Considered flags, Matched flags - wyjaśnię na przykładzie
 - i. Założmy, że wartości "considered flags" to: SYN,ACK,FIN,RST
 - ii. Założmy, że "matched flags" to: SYN

Oznacza to, że zostaną dopasowane pakiety tylko takie z ustawioną flagą SYN i nieustawioną flagą ACK,FIN,RST (wprowadzając dane nie stosujemy spacji)

Scapy

Klasa PacketManager w pełni wykorzystuje funkcje z biblioteki Scapy, na niej opiera się cała logika działania manipulatora pakietów. Spójrzmy na rys. 7., w funkcji run_manager odbywa się tworzenie kolejki NFQUEUE, na kolejce ustawiany jest callback, który wywołuje metodę process. Warto zauważyć, że kolejka jest tworzona na pozycji 0.

```
class PacketManager(object):

    def __init__(self, queue, queueRaw):
        print("PacketManager initialized.")
        self.queue = queue
        self.queueRaw = queueRaw

    def run_manager(self):
        q = nfqueue.queue()
        q.open()
        q.bind(socket.AF_INET)
        q.set_callback(self.process)
        q.create_queue(0)
        try:
            print("NFQUEUE ran, socket binded.")
            q.try_run()
        except:
            print(sys.exc_info()[0])
            print("NFQUEUE closed, socket unbinded.")
            q.unbind(socket.AF_INET)
            q.close()
```

Rys. 7.: Część kodu klasy PacketManager.

Metoda process przedstawiona na Rys. 8. wybiera dane z pakietu dodanego do kolejki NFQUEUE, następnie rozpakowuje je za pomocą metody IP(), pakiet jest porzucany (NF_DROP), po to by obsłużyć go z poziomu kodu. Istotne są dwie listy - queue oraz queueRaw. Lista queue zawiera dane, które będą wyświetlane w tabeli, zaś w queueRaw jest utrzymywany nierozpakowany pakiet, tak aby móc go wykorzystać w dowolnym momencie. Warto dodać, że obie te listy są zasobem dzielonym przez dwa procesy.

```

def process(self, i, payload):
    data = payload.get_data()
    pkt = IP(data)
    proto = pkt.proto

    #print(str(pkt).encode("HEX"))
    # Dropping the packet
    payload.set_verdict(nfqueue.NF_DROP)

    # Some console logging
    print(pkt.summary())

    #Add to multiprocessing queue the data (displayed in table)
    layers = build_packet_layer(pkt)
    dictToDisplay = {}
    for layer in layers:
        dictToDisplay[layer.name] = layer.fields
    self.queue.append(dictToDisplay)

    #Origin not formatted data holded in queueRaw, also multiprocess resource
    self.queueRaw.append(data)

```

Rys. 8.: Metoda “process”

Najbardziej istotną metodą, która odsyła zmodyfikowany przez nas pakiet, w sposób półautomatyczny lub manualny jest “send_packet_based_on_layers”, widoczny na rysunku 9. Jej definicja znajduje się poza klasą PacketManager, jednakże w tym samym pliku. Klasa ta na podstawie każdej warstwy pakietu (każdego wiersza w tabeli edycji pakietu) buduje pakiet, który zostanie odesłany. Argumenty, które są przekazywane to odpowiednio:

- layersNew - nowe warstwy (pobrana bezpośrednio z tabeli)
- raw - oryginał pakietu, którego warstwy zostały nadpisane
- handleSrcAndDst - zamienia miejscami wartości pól src i dst
- handlePorts - w zależności od protokołu modyfikuje bądź zamienia miejscami wartości pól sport i dport (source port, destination port)
- handleChksumAndLen - oblicza za nas długość oraz sumę kontrolną pakietu

```
def send_packet_based_on_layers(layersNew, raw, handleSrcAndDst, handlePorts, handleChksumAndLen):
    pkt = get_packet_from_raw(raw)
    counter = 0
    while True:
        lay = pkt.getlayer(counter)
        if (lay == None):
            send_raw_packet_back(pkt, handleSrcAndDst, handlePorts, handleChksumAndLen)
            break
        else:
            for key, value in lay.fields.iteritems():
                newLay = eval(str(layersNew[counter]))
                lay.fields[key] = newLay[key]
                if "load" in key:
                    pkt[counter - 1].payload = newLay[key]
            counter = counter + 1
```

Rys. 9.: Metoda obsługująca odesłanie pakietu zmodyfikowanego widoku edycji pakietu

Aby zobrazować omawiane wyżej funkcje spójrzmy na rys. 10. W górnym lewym rogu znajduje się hexdump oryginalnej wiadomości. Na prawo znajdują się trzy pola, które można zaznaczyć. Są to wspomniane trzy ostatnie argumenty z metody “send_packet_based_on_layers”. Argument pierwszy layersNew - to każdy z wierszy złożony w jeden pakiet. Edytujemy pakiet przez bezpośrednie zmiany wartości. Jeżeli wykonamy coś źle - pakiet nie zostanie wysłany, a w konsoli pojawią się błędy.

The screenshot shows a 'Packet edit' window. On the left, there is a hex dump of a packet. The top right contains three checkboxes: 'Handle src and dst', 'Handle sport and dport', and 'Handle chksum and len'. The main area displays a list of packet layers with their corresponding JSON-like structures. At the bottom, there are 'Send' and 'Cancel' buttons.

Offset	Hex Dump	ASCII
0000	45 00 00 3C 4F 40 00 00 80 01 69 67 C0 A8 00 64	E...<O@....ig...d
0010	C0 A8 00 65 08 00 4D 0F 00 01 00 4C 61 62 63 64	...e..M....Labcd
0020	65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74	efghijklmnopqrst
0030	75 76 77 61 62 63 64 65 66 67 68 69	uvwabcdefghi

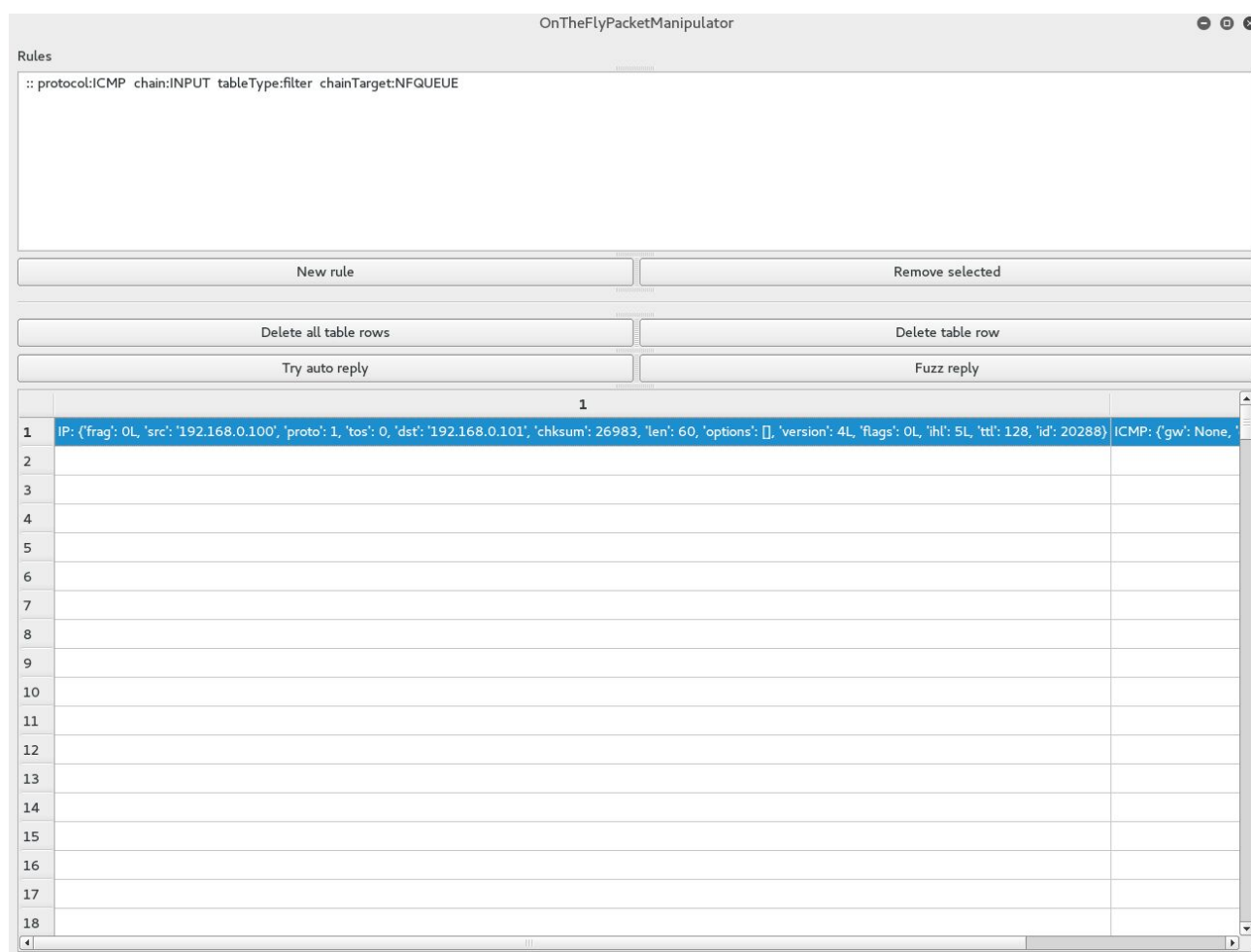
Layer	Structure
1	{'frag': 0L, 'src': '192.168.0.100', 'proto': 1, 'tos': 0, 'dst': '192.168.0.101', 'chksum': 26983, 'len': 60, 'options': [], 'version': 4L, 'flags': 0L, 'ihl': 5L, 'ttl': 128, 'id': 20288}
2	{'gw': None, 'code': 0, 'ts_ori': None, 'addr_mask': None, 'seq': 76, 'nextthopmtu': None, 'ptr': None, 'unused': None, 'ts_rx': None, 'length': None, 'chksum': 19727, 'reserved': None, 'ts_tx': None, 'type': 0}
3	{'load': 'abcdefghijklmnopqrstuvwabcdefghi'}

Rys. 10.: Okno edycji pakietu

PyQt

Na całość aplikacji składa się kilka okien oraz komponentów. Główne okno przedstawione jest na rys. 11. Na samej górze w sekcji “rules” znajdują się reguły, które zostały dodane - dodawanie odbywa się przez naciśnięcie przycisku “New rule”, co otwiera okno, które zostało już wcześniej omówione. “Remove selected” usuwa jeden wpis z sekcji “rules”.

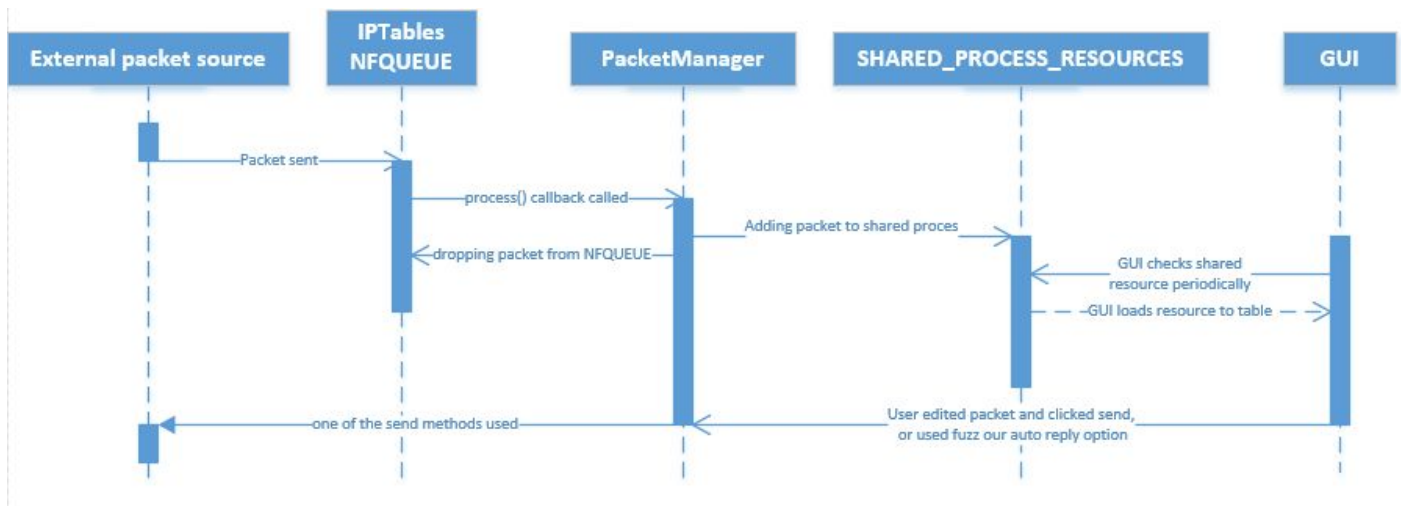
Następne przyciski dotyczą wierszy w tabeli. Każda z kolumn tabeli przedstawia pewną warstwę pakietu, wiersze w tym widoku są nieedytowalne, aby to zrobić należy nacisnąć dwukrotnie na wierszu (otwierany jest wtedy okno edycji pakietu). “Delete all table rows” usuwa wszystkie wiersze tabeli, “delete table row”, tylko jeden - aktualnie zaznaczony. Widoczne są również opcje “Try auto reply” oraz “Fuzz reply”. Zostały już wcześniej przedstawione ich funkcje - naciśnięcie ich nie powoduje otwarcia okna edycji pakietów. Opis kodu GUI jest nieistotny z punktu założeń funkcjonalnych - wyjaśnienie znajdziemy w dokumentacji PyQt.



Rys. 11.: Główne okno programu

Schemat działania

Rys. 12. prezentuje poglądowy diagram sekwencji, który przedstawia przepływ danych przez aplikację. Na początku otrzymujemy pakiet z zewnętrznego źródła - jeżeli reguła została dodana w IPTables to zostaje przekazany menadżerowi pakietów. Ten dodaje go do wspólnych zasobów dla procesów, oraz ustawia na nim werdykt, by go porzucić. GUI w okresie 500ms odświeża tabelę, sprawdzając czy zaszły zmiany w zasobach. Użytkownik otwiera jeden z pakietów edytuje oraz odsyła lub korzysta z opcji automatycznej odpowiedzi, czy też testowania pakietu (fuzz), pakiet trafia w zmienionej formie do pewnego źródła - bazując na zawartości pakietu.



Rys. 12.: Poglądowy diagram sekwencji przepływu pakietu

Najbardziej istotna część programu znajduje się w pliku "ApplicationMain". Kod pliku przedstawiony jest na rys. 13. Metoda "main()" jest główną metodą całej aplikacji. W niej na początku wykonywana jest kopia zapasowa zawartości iptables. Następnie cała zawartość iptables jest czyszczona. Kolejno tworzone są dwie listy, które są zasobem współdzielonym dla procesu packetManagerProcess, który jest demonem oraz dla procesu appProcess. Na samym końcu zawartość iptables jest przywracana do stanu sprzed uruchomienia programu oraz upewniamy się, że program został wyłączony całkowicie.

```

def runApp(queue, queueRaw):
    mv = MainView(queue, queueRaw)
    sys.exit(app.exec_())

def runPacketManager(queue, queueRaw):
    pm = PacketManager(queue, queueRaw)
    pm.run_manager()

def main():
    try:
        call("iptables-save > iptables-backup", shell=True)
        call("iptables -t filter --flush", shell=True)
        call("iptables -t nat --flush", shell=True)
        call("iptables -t raw --flush", shell=True)
        call("iptables -t mangle --flush", shell=True)

        queue = multiprocessing.Manager().list()
        queueRaw = multiprocessing.Manager().list()
        packetManagerProcess = multiprocessing.Process(target=runPacketManager, args=(queue, queueRaw, ))
        packetManagerProcess.daemon = True
        packetManagerProcess.start()

        appProcess = multiprocessing.Process(target=runApp, args=(queue, queueRaw))
        appProcess.daemon = False
        appProcess.start()

        appProcess.join()
        packetManagerProcess.terminate()
    finally:
        print("Restoring ip tables...")
        call("iptables-restore < iptables-backup", shell=True)
        print("Restored.")

        print("Cleaning up.")
        call("pkill -f ApplicationMain.py", shell=True) #making sure scapy nfqueue has been closed
        print("Bye!")

```

Rys. 13.: Główna funkcja programu, powoływanie procesów

Qt

Aby lepiej zrozumieć działanie przesyłanych pakietów postanowiliśmy utworzyć prostą aplikację, które wysyła pakiety (TCP) oraz datagramy (UDP). W tym celu skorzystaliśmy z C++ oraz silnika Qt.

TCP

Na rys. 14. zaprezentowany jest kod wykorzystywany do testowania pakietów protokołu TCP. Slot `printData()` informuje nas jaką wiadomość odbierzemy od manipulatora, zaś sloty `connected()` oraz `disconnected()` poinformują nas, gdy się połączymy bądź rozłączymy.

```
socket = new QTcpSocket();
connect(socket, SIGNAL(connected()), this, SLOT(connected()));
connect(socket, SIGNAL(disconnected()), this, SLOT(disconnected()));
connect(socket, SIGNAL(readyRead()), this, SLOT(printData()));

socket->connectToHost("192.168.0.101", 1000);
socket->waitForConnected(45000);
if (socket->state() == QTcpSocket::ConnectedState){
    socket->write("ALOHA MAN");
}
```

Rys. 14.: Kod gniazda TCP

UDP

Na rys. 15. zaprezentowany jest kod wykorzystywany do testowania pakietów protokołu UDP. Slot `printUdpData()` odbiera oraz odsyła dane do manipulatora. Nasłuchiwanie odbywa się na porcie 1000, zaś wysyłanie na porcie 999.

```
udpSocket = new QUdpSocket();
connect(udpSocket, SIGNAL(readyRead()), this, SLOT(printUDPData()));

QHostAddress address("192.168.0.100");
udpSocket->bind(address, 1000);

udpSocketSender = new QUdpSocket();
udpSocketSender->writeDatagram("Jeden dwa trzy!", QHostAddress("192.168.0.101"), 999);
```

Rys. 15.: Kod gniazda UDP

Instrukcja obsługi

Wszystkie wskazówki dotyczące programu można wyczytać na poprzednich stronach dokumentacji.

Aby uruchomić program należy mieć zainstalowane:

- System linux z jądrem skompilowanym wraz z ip_tables
- Python 2.7
- Scapy
- NFQUEUE

Następnie uruchamiamy terminal, przechodzimy do folderu z projektem i wpisujemy:

```
python ApplicationManager.py
```

Wszystkie błędy aplikacji logowane są w konsoli.

Cheat sheet

Zamieszczone zostaje tutaj kilka tabel, spisów, które usprawnią manipulowanie pakietami, bez konieczności posiadania dokumentacji.

Wartości HEX flag protokołu TCP	
0x00 NULL	0x80 CWR
0x01 FIN	0x81 FIN-CWR
0x02 SYN	0x82 SYN-CWR
0x03 FIN-SYN	0x83 FIN-SYN-CWR
0x08 PSH	0x88 PSH-CWR
0x09 FIN-PSH	0x89 FIN-PSH-CWR
0x0A SYN-PSH	0x8A SYN-PSH-CWR
0x0B FIN-SYN-PSH	0x8B FIN-SYN-PSH-CWR
0x10 ACK	0x90 ACK-CWR
0x11 FIN-ACK	0x91 FIN-ACK-CWR
0x12 SYN-ACK	0x92 SYN-ACK-CWR
0x13 FIN-SYN-ACK	0x93 FIN-SYN-ACK-CWR
0x18 PSH-ACK	0x98 PSH-ACK-CWR
0x19 FIN-PSH-ACK	0x99 FIN-PSH-ACK-CWR
0x1A SYN-PSH-ACK	0x9A SYN-PSH-ACK-CWR
0x1B FIN-SYN-PSH-ACK	0x9B FIN-SYN-PSH-ACK-CWR
0x40 ECE	0xC0 ECE-CWR
0x41 FIN-ECE	0xC1 FIN-ECE-CWR
0x42 SYN-ECE	0xC2 SYN-ECE-CWR
0x43 FIN-SYN-ECE	0xC3 FIN-SYN-ECE-CWR
0x48 PSH-ECE	0xC8 PSH-ECE-CWR
0x49 FIN-PSH-ECE	0xC9 FIN-PSH-ECE-CWR
0x4A SYN-PSH-ECE	0xCA SYN-PSH-ECE-CWR
0x4B FIN-SYN-PSH-ECE	0xCB FIN-SYN-PSH-ECE-CWR
0x50 ACK-ECE	0xD0 ACK-ECE-CWR
0x51 FIN-ACK-ECE	0xD1 FIN-ACK-ECE-CWR

0x52 SYN-ACK-ECE	0xD2 SYN-ACK-ECE-CWR
0x53 FIN-SYN-ACK-ECE	0xD3 FIN-SYN-ACK-ECE-CWR
0x58 PSH-ACK-ECE	0xD8 PSH-ACK-ECE-CWR
0x59 FIN-PSH-ACK-ECE	0xD9 FIN-PSH-ACK-ECE-CWR
0x5A SYN-PSH-ACK-ECE	0xDA SYN-PSH-ACK-ECE-CWR
0x5B FIN-SYN-PSH-ACK-ECE	0xDB FIN-SYN-PSH-ACK-ECE-CWR

Typy ICMP	
0	Echo Reply
1	Unassigned
2	Unassigned
3	Destination Unreachable
4	Source Quench
5	Redirect
6	Alternate Host Address
7	Unassigned
8	Echo
9	Router Advertisement
10	Router Selection
11	Time Exceeded
12	Parameter Problem
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
19	Reserved (for Security)
20-29	Reserved (for Robustness Experiment)
30	Traceroute
31	Datagram Conversion Error
32	Mobile Host Redirect
33	IPv6 Where-Are-You
34	IPv6 I-Am-Here
35	Mobile Registration Request
36	Mobile Registration Reply
37	Domain Name Request
38	Domain Name Reply
39	SKIP
40	Photuris
41-255	Reserved