



Podstawy Teleinformatyki
Dokumentacja

 EvilPostman

Graficzny modyfikator pakietów On-The-Fly
v.1.0.2-a1

prowadzący:
mgr. inż. Przemysław Walkowiak

1.Cel projektu:

Celem projektu jest implementacja narzędzia/aplikacji mającej za zadanie umożliwić użytkownikowi przechwytywanie przesyłanych pakietów w sieci, a następnie przy pomocy graficznego interfejsu wykonanie modyfikacji ich parametrów i ponowną transmisję.

2.Charakterystyka ogólna projektu:

Zadanie to powinno być zrealizowane przy użyciu dostępnej w komputerze karty sieciowej, a wykonywanie wyżej wymienionego celu powinno przebiegać przy jak najmniejszym możliwym opóźnieniu. Wspomniane opóźnienie wynikać będzie z potrzeby zatrzymywania pakietów na czas ich przetworzenia, a także ponownej transmisji w sieci. Ponadto proces przechwytywania, parsowania i modyfikowania pakietów powinien przebiegać w sposób zautomatyzowany bez potrzeby nieustannej ingerencji użytkownika w “locie” (ang. *on-the-fly*).

Do tego celu należy prawidłowo skonfigurować reguły programu iptables w systemie Linux, przy jednoczesnym umiejętnym wykorzystaniu mechanizmu kolejkowania pakietów w karcie sieciowej (np. *NFQUEUE*). W celu ułatwienia edycji i przekształcania przechwytywanych danych, zaleca się skorzystanie z biblioteki *Scapy*. Ze względu na wykorzystanie reguł programu iptables i funkcji *NFQUEUE* projekt ten wymaga użycia dowolnego systemu z rodziny Linux, gdyż oprogramowanie to jest ściśle związane z linuxowym kernelem.

Polecane dystrybucje Linux'owe dla aplikacji Evilpostman:

- a. Arch - <https://www.archlinux.org/>
- b. Ubuntu - <https://www.ubuntu.com/>
- c. Debian - <https://www.debian.org/>
- d. Gentoo - <https://www.gentoo.org/>
- e. DSL - <http://www.damnsmalllinux.org/>
- f. Knoppix - <http://www.knopper.net/knoppix/>
- g. Mint - <https://linuxmint.com/>
- h. KaliLinux - <https://www.kali.org/>

3. Wymagania funkcjonalne

- a. Możliwość przechwytywania pakietów.
- b. Możliwość modyfikacji pakietów.
- c. Możliwość odfiltrowania pakietów wybranych do modyfikacji.
- d. Retransmisja zmodyfikowanych pakietów.
- e. Wykonania czynności „on-the-fly”.

4. Wymagania niefunkcjonalne:

- a. interfejs powinien być przejrzysty i intuicyjny.
- b. możliwość wyświetlenie instrukcji dotyczącej użytkowania.
- c. implementacja aplikacji w języku python.
- d. skalowalny interfejs.
- e. wyświetlanie przechwyconych pakietów w kolejności ich nadania.

5. Funkcjonalność oferowana przez aplikacje:

- a. Przechwytywania wybranych rodzajów pakietów wraz ze specyfikacją konkretnych warstw i wartości pól.
- b. Możliwość tworzenia filtrów i ich zestawów połączonych za pomocą relacji *and* oraz *or*. Modyfikacja dokładnych wartości w konkretnych warstwach pakietów(protokołach).
- c. Możliwość tworzenia i łączenia modyfikatorów z filtrami na podstawie nazwy.
- d. Modyfikacja pola danych przy użyciu modyfikatora wpływającego na pola typu Raw.
- e. Modyfikacja dowolnych pól danych we wszystkich pakietach wspieranych przez Scapy

6. Uzasadnienie wyboru projektu

Wybór modyfikatora pakietów “on-the-fly”, jako projektu podyktowany został chęcią zgłębienia trudnego jak i specyficznego zagadnienia dotyczącego przechwytywania, analizowania i parsowania danych komunikacji sieciowej. Na rynku istnieje wiele gotowych narzędzi umożliwiających sniffowanie sieci (np. *Wireshark*), ekstrakcję plików i danych (np. *Network Miner*) przekierowywanie ruchu (np. *Burp Suite*) czy manipulowanie zawartością pakietów (*Polymorph*).

Dlatego też implementacja podobnej aplikacji może przynieść sporą wiedzę dotyczącą działa tego typu programów, a co za tym idzie zrozumienie funkcjonowania urządzeń internetowych jak np. karta sieciowa czy router, a także informacji na temat zabezpieczeń sieci i działania towarzyszącym im protokołów.

Poza wymienionymi już aspektami realizacja tego projektu jest również doskonałą okazją do pogłębienia doświadczenia na temat biblioteki *Scapy* i umiejętności jej wykorzystywania w programowaniu aplikacji internetowych.

Również ważnym powodem wybrania powyższego tematu dla projektu jest wysokopoziomowy i dynamiczny język programowania Python 3. Kilkuletnie doświadczenie podczas studiów i pracy w tworzeniu skryptów za pomocą języka Python zdecydowanie ułatwi wykonanie powyższych celów, a umiejętności z nim związane pozwalają ocenić jego wybór jako idealny do tego zadania.

Za tym wyborem stoją nie tylko wspomniane cechy czy doświadczenie, ale także wsparcie społeczności i tysiące aktualnie rozwijanych bibliotek takich jak np.

biblioteka *Scapy* - pozwalająca na analizę i modyfikację pakietów sieciowych (wymieniona wcześniej),

PyQt5 - pythonowe bindingi (zaimplementowane w Pythonie/C++) do złożonego zestawu cross-platformowych bibliotek i narzędzi

Qt5 - służących do szybkiego i efektywnego budowania graficznych interfejsów,

bindingi *python-nfqueue* dla programu *iptables* umożliwiające dostęp i obsługę Nfqueue z poziomu skryptów.

Należy jednocześnie zwrócić uwagę, że osoby odpowiedzialne za bezpieczeństwo systemów informatycznych jak administratorzy, czy pentesterzy nierzadko wybierają ten język jako narzędzie do budowy skryptów. Przeważająca prostota i dynamiczne przydzielanie typów, upraszcza i przyspiesza budowę narzędzi pozwalających na zarządzanie czy testowanie odmiennych systemów informatycznych, a tym samym kontrolę ich bezpieczeństwa.

Podsumowując wybór modyfikatora pakietów jako projektu podyktowany był następującymi aspektami:

- Interesująca tematyka dotycząca podsłuchiwania i przetwarzania pakietów sieciowych.
- Dekodowanie pakietów sieciowych i manipulacja ich zawartością (DPI - Deep Packet Inspection).
- Zainteresowanie tematyką bezpieczeństwa systemów komputerowych i działania protokołów.
- Możliwość poszerzania umiejętności w programowaniu aplikacji sieciowych.
- Wieloletnie doświadczenie w programowania w języku Python 3.x.
- Biblioteki Scapy - chęć pogłębienia wiedzy na jej temat i możliwości jej wykorzystania.
- Zainteresowanie technologiami sieciowymi.
- Potrzeba stworzenia własnych narzędzi do testów penetracyjnych, z naciskiem na możliwość szybkiej rozbudowy i dostosowania do aktualnych potrzeb.

7. Podział prac:

Imię i nazwisko	Zadania
Jarosław Wieczorek	<ul style="list-style-type: none">• Automatyzacja procesu projektowania interfejsów.• Utworzenie klas pośredniczących - dążenie do rozdzielenia funkcjonalności od interfejsu.• Obsługa pakietów pobieranych z NFQUEUE• Generowanie niezbędnych danych do tworzenia filtrów• Generowanie niezbędnych danych do tworzenia modyfikatorów• Budowanie dynamicznych interfejsów na podstawie obsługiwanych protokołów w Scapy
Krzysztof Orczyk	<ul style="list-style-type: none">• Konfiguracja NFQUEUE.• Tworzenie reguł iptables.• Asynchronicznie przechwytywanie pakietów.• Obsługa pakietów dodawanych do NFQUEUE.• Generowanie filtrów na podstawie danych zwróconych z interfejsu.• Generowanie modyfikatorów na podstawie danych zwróconych z interfejsu.• Przekazywanie pakietów z powrotem do systemu.• Modyfikacja i filtrowanie pakietów.
Dominika Pawlaczyk	<ul style="list-style-type: none">• DPI• GUI• Filtry automatyzujące• Konfiguracja iptables, NFQUEUE

8. Opis poszczególnych zadań:

Automatyzacja procesu projektowania interfejsów przed rozpoczęciem realizacji projektu:

Zadanie to polega na opracowaniu środowiska i skryptów usprawniających proces budowania interfejsów użytkownika. Dzięki takiemu rozwiązaniu czas poświęcony na projektowanie interfejsów graficznych i późniejszym ich odwzorowaniu za pomocą kodu (w wybranej bibliotece graficznej) będzie ograniczony do wymaganego minimum.

Wykonanie opisanej konfiguracji środowiska niesie za sobą dodatkową zaletę. W razie potrzeby modyfikacji któregokolwiek z gotowych widoków lub dołożenia nowego okienka, przycisku, bądź tabeli, powyższe zmiany będą wprowadzane do aplikacji za pomocą jednej komendy w terminalu.

Zaprojektowany interfejs zostanie transformowany w kod w języku Python 3 z wykorzystaniem wybranej biblioteki graficznej - PyQt5.

Tworzenie klas pośredniczących:

W celu zapewnienia możliwie jak najbardziej hermetycznego kodu i utrzymania podziału między logiką aplikacji Evilpostman, a jej interfejsem postanowiono zastosować dodatkowe klasy zarządzające.

Klasy zarządzające dziedziczą klasy wyprodukowane z projektu interfejsu użytkownika. Stanowią one warstwę komunikacyjną, pomiędzy logiką, a samym interfejsem. To dzięki ich metodom wywoływane są okna, obsługiwane event'y czy dodawane elementy do listy. Do ich głównych zadań należy przede wszystkim udostępnianie wysokopoziomowych method takich jak:

setButtonFunct(button_name, function_name).

Metoda ta ułatwia "podpinanie" dowolnej funkcji z dowolnie wybranym przyciskiem w interfejsie. Dzięki takiemu rozwiązaniu połączenie funkcjonalności odbywa się poprzez wywoływanie metod ***setButtonFunct*** z argumentami będącymi odpowiednio nazwą przycisku i nazwą podpinanej funkcji (z ewentualnymi argumentami).

Przykład użycia:

```
self.setButtonFunct(self.button_foo, partial(self.foo, arg1, arg2))
```

W przeciwnym przypadku, aby podłączyć metodę do wybranego *button'a* musielibyśmy skorzystać z poniższej instrukcji:

```
self.button_foo.clicked.connect(partial(self.funct_foo, arg1, arg2)).
```

Choć wydawać, by się mogło, że różnica w kodzie jest niewielka, a nawet pogorszyła się czytelność i przybyły dodatkowych znaków, należy nadmienić fakt, że różne komponenty biblioteki PyQt5 mogą korzystać z całkiem odmiennych sposobów podlinkowywania funkcjonalności. Za konkretny przykład może posłużyć obsługa eventów dla widżetu QCombobox, której instrukcja wywołania przedstawiona jest poniżej:

```
self.combo_box.activated.connect(self.handleActivated).
```

Gdzie słowo “activated” oznacza event aktywowania widżetu tzn. użytkownik kliknął na niego w celu zmiany aktualnej opcji. Zastosowane powyższego podejście pozwala na eliminację tych różnic. Poza tym klasy zarządzające są dziedziczone przez klasy pythonowe z wyższej warstwy. Udostępnione tak metody pozwalają wielokrotnie i w prosty sposób zmieniać funkcjonalność przycisków nawet w trakcie działania aplikacji.

Powstaje tu dodatkowy problem dotyczący zmieniających się nazw widgetów podczas projektowania czy modyfikowania interfejsu. Aby pozbyć się tego problemu planowane jest rozbicie powyższej metody na wiele mniejszych.

A każda z nich odpowiedzialna będzie za konkretny widżet. Przykład podłączenia funkcji pod przycisk odpowiedzialnego za włączenie sniffowania pakietów:

```
self.setFuncOnSniffButton(partial(self.fuct, arg1, arg2))
```

Jeśli nazwa widgetu z jakiegoś powodu się zmieni np. z *push_button_1* na *push_button_2* dla programistów odpowiedzialnych za logikę nie będzie to miało znaczenia.

Konfiguracja NFQUEUE:

Zadaniem jest wybrać odpowiednią bibliotekę, która pozwoli python'owi współpracować z NFQUEUE. Istnieje wiele rozwiązań, jednym z wymagań jest możliwość akceptacji i modyfikacji pakietów. Program musi przed rozpoczęciem przechwytywania połączyć się z NFQUEUE i rozpocząć nasłuchiwanie oczekując na pakiety.

Tworzenie reguł iptables:

Przekazywanie pakietów do odpowiedniego NFQUEUE dzieje się poprzez dodanie odpowiednich reguł zarządzających ruchem sieciowym do iptables. Zadanie polega na stworzenia odpowiednich zasad, które przekazywały by właściwe pakiety do NFQUEUE. W tym wypadku potrzebujemy przekazywać, pakiety przychodzące do tej maszyny i przekazywać je do kolejki numer 0.

Asynchroniczne przechwytywanie pakietów:

Podczas przechwytywania pakietów przez NFQUEUE, program oczekuje na nowe pakiety przychodzące i uruchamia odpowiednią funkcję mającą za zadanie obsługę nowego pakietu. Problemem, który należy tutaj rozwiązać jest oddelegowanie oczekiwania na pakiety przychodzące do nowego wątku i zamykanie tego wątku po zakończeniu działania programu.

Wykorzystana jest do tego biblioteka python Threading. Pozwala to na stworzenie klasy NFQController, dziedziczącej po threading.Thread. Uzyskany w ten sposób interfejs pozwala uruchomić przechwytywanie pakietów w tle tak aby nie przeszkadzało w wykonaniu reszty programu i nie powodowało dodatkowych spowolnień.

Obsługa pakietów dodawanych do NFQUEUE:

Pakiety, które zostaną przechwycone i dodane do odpowiedniego NFQUEUE muszą być obsłużone przez program. Aby w szybki i wygodny sposób obsłużyć taki pakiet wykorzystana zostanie biblioteka Scapy. Każdy pakiet zostaje rzutowany na taką postać i przekazywany dalej.

Ważną rzeczą w takiej sytuacji jest szybkie zwrócenie pakietu wraz z decyzją czy powinien zostać zaakceptowany czy też odrzucony, jako że NFQUEUE ma ograniczoną długość i jeżeli zostanie zapełniona, następne pakiety będą odrzucane. Należy więc stworzyć funkcje uruchamianą z każdym nowym pakietem, która będzie podejmować decyzję na ich temat.

Pierwszą rzeczą powinno być sprawdzenie czy dany pakiet nie pasuje do żadnego z filtrów. Jeżeli tak należy go zmodyfikować i zwrócić do systemu. Wszystko to musi zająć jak najmniej czasu tak więc należy sprawdzać tylko tyle ile jest absolutnie konieczne, żeby nie wprowadzać zbyt dużych opóźnień.

Generowanie filtrów na podstawie danych zwróconych z interfejsu:

Użytkownik będzie miał do dyspozycji interfejs użytkownika przy pomocy, którego będzie mógł stworzyć filtry, które będą używane do sortowania pakietów. Interfejs powinien zwracać uniwersalny i szybki w iteracji format, który będzie mógł być wykorzystany do sprawnej weryfikacji pakietów przychodzących. Wykorzystany jest do tego słownik, pozwalający na szybką referencję konkretnych wartości.

Generowanie modyfikatorów na podstawie danych zwróconych z interfejsu:

Podobnie jak przy generowaniu filtrów część programu obsługująca modyfikacje pakietów będzie otrzymywać uporządkowaną strukturę danych zawierającą odpowiednie wartości, które należy zmodyfikować podczas obsługi pakietu. Ważne jest tutaj przekazanie nazwy filtra, do którego odnosi wybrany modyfikator. Dzięki temu wiadomo jakie pola powinny być zmienione w jakich rodzajach pakietów.

Przekazywanie pakietów z powrotem do systemu:

Po dokonaniu modyfikacji otrzymany pakiet w formacie Scapy musimy odpowiednio obrobić przed zwróceniem go do systemu. Pierwszą rzeczą jest wykorzystanie Scapy aby przeliczyć ponownie i uzupełnić wszystkie dynamicznie generowane pola takie jak na przykład sumy kontrolne. Następnie musimy zamienić pakiet w tablicę bajtów, zwrócić do kolejki i zaakceptować, tak aby przesłać go dalej i zrobić miejsce na nowe pakiety przychodzące.

9.Użyte Technologie:

- a. Python w wersji 3.x
- b. NFQUEUE
- c. Iptables

10. Opis użytych technologii:

Opis użytych technologii ma na celu przybliżyć szczegółowo wybrane technologie zapomocą, których realizowano zadania projektowe.

Zawiera informacje dotyczące każdej z wykorzystywanych technologii jak:

Co to jest?

Do czego służy?

Do czego jest to wykorzystywane w projekcie?

Opis Python w wersji 3.x:

Python – język programowania wysokiego poziomu ogólnego przeznaczenia, o rozbudowanym zestawie bibliotek standardowych, którego ideą przewodnią jest czytelność i klarowność kodu źródłowego.

Jego składnia cechuje się prostotą, przejrzystością i zwięzłością. Python wspiera różne paradigmaty programowania: obiektowy, imperatywny czy funkcyjny. I podobnie jak w przypadku **C++** pozwala na stosowanie mieszanych stylów tzn. nie narzuca użycia jednego stylu. Język Python dla ułatwienia instalacji pakietów posiada własny *packet-manager* o nazwie *Pip*.

Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią, będąc w tym podobnym do języków takich jak Perl czy Ruby. Podobnie jak inne języki dynamiczne jest często używany jako język skryptowy. Interpretery Pythona są dostępne na wiele systemów operacyjnych, w tym także na mikrokontrolery (np. micro-python).

iptables:

Co to jest:

Iptables jest programem obsługiwany z linii komend do konfigurowania zapory ogniowej na poziomie kernela w systemie linux.

Do czego służy:

Wykorzystując iptables możemy skonfigurować jak chcemy aby nasz system obsługiwał pakiety i ruch sieciowy w wybrany przez nas sposób. Możemy zablokować lub zezwolić na pewne rodzaje połączeń i tworzyć tablice routingu i forwardingu. Pozwala to na bardzo dokładne zarządzanie interakcją naszej maszyny z otoczeniem sieciowym.

Wykorzystanie w naszej aplikacji:

Iptables pozwala nam na przechwytywanie przychodzących pakietów zanim trafią do reszty systemu i zostaną obsłużone. Daje nam to dużą kontrolę nad rodzajem ruchu sieciowego, który przekazujemy do naszej aplikacji. Wykorzystujemy do tego zwykłą zasadę, iptables, tylko celem, do którego przekazujemy przychodzące pakiety jest funkcja NFQUEUE opisana niżej.

NFQUEUE:

Co to jest:

NFQUEUE jest to część linuxowego programu iptables i ip6tables przekazująca decyzje na temat losu pakietów, do warstwy aplikacji w systemie. Jest to standardowa funkcja iptables dostępna w każdym kernel'u linux'owym.

Jak działa:

Aby przekazywać pakiety do NFQUEUE należy najpierw utworzyć odpowiednią regułę zgodnie z zasadami dotyczącymi iptables na podstawie, której będziemy przekazywali pakiety do odpowiedniego numeru NFQUEUE. Stamłąd programy działające w środowisku użytkownika mogą je pobrać. Należy oczywiście pamiętać, że wielkość NFQUEUE jest ograniczona, i jeżeli zostanie zapełniona, następne pakiety przychodzące będą ignorowane.

Do czego służy:

Dzięki **NFQUEUE** aplikacje mają dostęp do pakietów, które przechodzą przez system. Mogą dzięki temu podglądać je, modyfikować oraz decydować czy powinny być zaakceptowane czy też odrzucone zanim zostaną przekazane dalej.

Wykorzystanie w naszej aplikacji:

Dzięki wykorzystaniu reguł iptables możemy dodatkowo filtrować pakiety dodawane do **NFQUEUE**, pozwoliłyby to na przykład na rozdzielanie pakietów na podstawie różnych kryteriów na różne kolejki. Nasza aplikacja wykorzystuje iptables oraz NFQUEUE dodając ogólną zasadę, która przekazuje wszystkie pakiety przechodzące do NFQUEUE o numerze pierwszym. Następnie wykorzystujemy event, wykonywany za każdym razem gdy nowy pakiet zostanie dodany do kolejki i tworzymy na jego podstawie obiekt pakietu Scapy i przekazujemy go do reszty programu, która wykona jego analizę i ewentualną modyfikację. Po zakończeniu tego procesu pakiet jest zwracany przekształcany w postać binarną po czym zwracany do NFQUEUE i akceptowany.

11. Użyte Biblioteki:

a. biblioteki standardowe:

- i. sys
- ii. os
- iii. functools
- iv. thread
- v. setuptools

b. biblioteki dodatkowe:

- i. Scapy
- ii. pyqt5
- iii. netfilterqueue

Scapy - bardzo rozbudowana biblioteka python'owa wykorzystywana do manipulacji pakietami sieciowymi.

Pozwala między innymi na tworzenie, dekodowanie, wysyłanie, łączenie pakietów, zapytań, odpowiedzi i wiele więcej. Może być wykorzystywany do debugowania programów i urządzeń sieciowych, skanowania, unit testów i wciąż więcej.

Wysokopoziomowe podejście umożliwia na szybkie prototypowanie i tworzenie własnych narzędzi w postaci relatywnie prostych skryptów w python'ie.

Jednym z ciekawszych systemów udostępnionych przez Scapy jest możliwość tworzenia pakietów od podstaw korzystając z rozbudowanych konstruktorów pakietów.

W naszym oprogramowaniu korzystamy głównie z:

- łatwego dekodowania, w postaci tworzenia instancji pakietu Scapy na podstawie tablicy bajtów,
- odnoszenia się do warstw pakietu poprzez ich nazwę co pozwala na szybką iteracje po polach i protokołach zawartych w pakiecie,
- funkcjonalności pozwalającej na automatyczne przeliczenie i uzupełnienie pól takich jak na przykład suma kontrolna.

Gdyby nie te wszystkie funkcje byłoby dużo trudniej stworzyć nasz program.

PyQt5 - jest nakładką na Qt napisaną w języku python pozwalającą na tworzenie interfejsów w pythonie. Samo Qt jest między platformową biblioteką udostępniającą możliwość tworzenia interfejsów napisaną oryginalnie w **C++**.

Netfilterqueue - jest biblioteką udostępniającą binding'i do libnetfilter_queue pozwalające programowi napisanemu w pythonie wygodnie komunikować się z NFQUEUE. Jest to bardzo ważna część naszego programu, pozwala ona nam przechwytywać, zatrzymywać i modyfikować pakiety w czasie rzeczywistym korzystając z narzędzi wbudowanych w każdy system linux.

12. Wykorzystane narzędzia:

Qt Designer - jest to narzędzie przygotowane przez programistów Qt do szybkiego, prostego projektowania interfejsów. Przy pomocy tego programu zaprojektowano wszystkie elementy interfejsu użytkownika Evilpostman. Interfejsy zapisywane są w formacie .ui, a wewnętrzną budowę tworzy kod języka QML przypominający składnię XML.

Spyder3 IDE - zintegrowane środowisko programistyczne (IDE) dla języka programowania Python 3. Istnieje również wersja Spyder IDE dla Pythona 2.7. Narzędzie wyposażone w wiele usprawnień jak podpowiadanie składni i zmiennych, interpreter IPython, wbudowaną powłokę terminala systemowego, wtyczkę do notebook'a jupytera i wiele innych dodatków.

PyCharm – zintegrowane środowisko programistyczne (IDE) dla języka programowania Python firmy JetBrains. Zapewnia m.in.: edycję i analizę kodu źródłowego, graficzny debugger, uruchamianie testów jednostkowych, integrację z systemem kontroli wersji. Wspiera także programowanie i tworzenie aplikacji internetowych w Django. Jest oprogramowaniem wieloplatformowym pracującym na platformach systemowych: Microsoft Windows, GNU/Linux oraz OS X. Wydawany jest w wersji Professional Edition, która jest oprogramowaniem własnościowym oraz w wersji wolnej Community Edition, która pozbawiona jest jednak niektórych zaawansowanych funkcjonalności.

PyCharm jest inteligentnym edytorem kodu z:

- Podświetlaniem składni dla języka Python oraz szablonów Django, formatowaniem kodu, autouzupełnianiem kodu źródłowego, tworzeniem snippetów.
- Edytor kodu wyposażony jest w funkcję sprawdzania i weryfikacji błędów składniowych na bieżąco w trakcie pisania kodu.
- Bogate narzędzia do refaktoryzacji.
- Zestaw narzędzi wspierających tworzenie aplikacji internetowych w Django.
- Edycja i wsparcie dla HTML, CSS oraz JavaScript.
- Wsparcie dla innych frameworków Flask, Pyramid, web2py.
- Wsparcie dla bibliotek SQLAlchemy, wxPython, PyQt, PyGTK.
- Programowanie na platformie Google App Engine.
- Zintegrowane testy jednostkowe z testami pokrycia.
- Graficzny debugger dla Pythona i Django.
- Integracja z systemami kontroli wersji Mercurial, Subversion, Git, CVS, Perforce.
- Wbudowany terminal.
- Możliwość instalacji dodatkowych pluginów.

Wireshark

Wireshark to sniffer pakietów danych należący do grupy darmowego i wolnego oprogramowania. Umożliwia przechwytywanie i nagrywanie pakietów sieciowych, a także ich dekodowanie.



Dzięki dużej ilości dodatków potrafi rozpoznać i zdekodować ogromna ilość protokołów komunikacyjnych. W głównej mierze jest wykorzystywany przez administratorów sieci, służby specjalne, hakerów i pentesterów do śledzenia pakietów. Wireshark potrafi rozpoznawać i dekodować również wyspecjalizowane protokoły przemysłowe np. Profinet, Powerlink.

Jedną z wielu zalet Wiresharka jest wykorzystanie graficznego interfejsu użytkownika. Dzięki niemu użytkownik może odczytać informacje takie jak np. listę adresów mac urządzeń komunikujących się w sieci. Wireshark pozwala na wyłuskiwanie rejestrowanych danych. Mogą to być nagrania audiowizualne, przesyłane hasła, dokumenty czy nawet klucze pośredniczące w wymianie informacji.

Wykorzystane systemy z rodziny Linux podczas testów *Evilpostman*:

- **Debian**
- **Ubuntu**
- **Arch.**

Krótki opis systemów linuxowych wykorzystanych do testów aplikacji *Evilpostman*:

Debian

Debian to wolny system operacyjny (OS). System operacyjny to zestaw podstawowych programów i narzędzi, które umożliwiają komputerowi działanie. Debian używa jądra (jest to rdzeń systemu operacyjnego) Linux, ale większość podstawowych narzędzi systemu pochodzi z projektu GNU. Dlatego system nazywa się GNU/Linux.



Debian to coś więcej, niż sam system operacyjny: zawiera ponad 50000 pakietów, które są skompilowanymi programami spakowanymi w sposób, który umożliwia łatwą instalację. Ciekawostka dotycząca nazewnictwa kolejnych wersji tego systemu - Każda kolejna odsłona wersji Debiana nosi nazwę będącą imieniem bohatera z animowanego filmu Toy Story.

Dystrybucja ta cieszy się opinią najbardziej stabilnego systemu operacyjnego o bardzo wysokiej jakości. Posiada także wsparcie dla wielu języków. Obecnie jest tworzony przez bardzo dużą grupę ochotników komunikujących się ze sobą poprzez szereg list dyskusyjnych oraz system śledzenia błędów. Projekt posiada rozbudowaną strukturę wewnętrzną: z wyborami, konstytucją a także formalnymi dokumentami określającymi zasady postępowania.

Arch Linux

Arch Linux jest dystrybucją Linuksa stworzoną przez Judda Vineta, która podkreśla łatwość użytkowania. Łatwość ta nie jest jednak osiągana przez mnogość graficznych konfiguratorów, a poprzez rozsądnie rozmieszczone i konstruowane pliki konfiguracyjne, skrypty i programy. Dlatego też Arch Linux, mimo swojej prostoty, może nie być odpowiednim systemem dla osób nie mających wcześniej styczności z Linuksem.



Ubuntu

Ubuntu - dystrybucja systemu operacyjnego Linux oparta na Debianie. Popularność zdobył głównie przez swoją prostotę, możliwości i łatwość obsługi. Instalacja tego systemu jest bardzo prosta (wygląda jak instalacja programów w systemie Windows tzn. przechodzimy od kroku do kroku).



Pierwotna nazwa projektu Ubuntu to no-name-yet.com. Pierwsze wydanie Ubuntu ukazało się 20 października 2004 roku jako tymczasowa odmiana dystrybucji Debian GNU/Linux. Jej początkowym celem było stworzenie własnego, 6-miesiecznego cyklu wydawniczego. Projekt stopniowo ewoluował i uniezależniał się. W przeciwieństwie do innych dystrybucji opartych na Debianie, Ubuntu pozostało jednak wierne jego filozofii, która zakłada wykorzystanie wyłącznie wolnego oprogramowania.

13. Opis ważniejszych klas w programie

Nazwa	Opis
QueuePacketCatcher	Jest to klasa odpowiedzialna za tworzenie kopii zapasowych i przywracania zasad iptables, uruchamiania przechwytywania i wywoływanie dodatkowych metod do obsługi złapanych metod.
NFQController	Klasa dziedzicząca po threading.Thread, uruchamiana z klasy QueuePacketCatcher jako osobny wątek. To ta klasa monitoruje kolejkę NFQUEUE i wywołuje metodę z QueuePacketCatcher kiedy złapany zostaje nowy pakiet.
Packet_handler	Jak nazwa wskazuje jest to klasa odpowiedzialna za obsługę przechwyconych pakietów. Tutaj znajdują się metody filtrujące i modyfikujące nowo przybyłe pakiety. Odpowiada także za uruchamianie okien tworzenia filtrów i modyfikatorów oraz pobieranie ich wyników w postaci gotowych struktur, które można zastosować do obróbki pakietów. Wyświetla, też odpowiednie pakiety w wybranych tabelach interfejsu.
Filters/Modifiers	Klasa tworząca okno dialogowe do tworzenia filtrów/modyfikatorów. Odpowiada nie tylko za wyświetlanie samego okna ale tworzenia proceduralnie układu na podstawie tego jakie protokoły wybiera użytkownik. Odczytuje wartości wypełnionych pól i zapisuje do specjalnego słownika, i przekazuje ten wynik do klasy Packet_handler gdzie na jego podstawie pakiety są filtrowane/modyfikowane.

14. Filtrowanie Pakietów

Po tym jak pakiet dostanie się do NFQUEUE tworzymy na jego podstawie odpowiedni obiekt pakietu Scapy i przekazujemy go funkcji handle_my_packet(pkt) w klasie **Packet_handler**.

W tej klasie schwytany pakiet jest dodawany do listy złapanych pakietów w interfejsie. Następnie sprawdzamy czy dany pakiet jest obejmowany przez, któryś z filtrów. Dokonuje tego funkcja filter(self, packet). Przekazujemy do niej pakiet, który chcemy sprawdzić. Wewnątrz niej literujemy najpierw po wszystkich filtrach. Filtry są zapisywane w słowniku, gdzie klucz to nazwa filtra a wartość to lista wszystkich filtrowanych warstw w danym filtrze i poszczególne pola według których filtrujemy w formacie: nazwa pola, wartość pola.

Pierwszym testem jest próba odwołania się do konkretnego protokołu w pakiecie, jeżeli to się powiedzie i znajdziemy odpowiedni protokół, wykorzystujemy funkcje getattr, która w pythonie pozwala na uzyskanie wartości danego pola na podstawie jego nazwy i porównujemy uzyskaną wartość z wartością oczekiwanaą z filtru.

Jeżeli, na którymkolwiek z etapów uzyskaliśmy wartość inną niż oczekiwana, sprawdzanie jest przerwane aby zmniejszyć czas przetwarzania pakietów. Jeżeli jednak wszystkie testy się powiodły i pakiet pasuje do jednego z filtrów, jako wynik zwracamy dwie wartości, pierwszą boolowską oznaczającą czy udało się dopasować pakiet do filtru i drugą z nazwą filtru, który został dopasowany do tego pakietu. Dzięki temu wiemy jakie modyfikacje, chcemy wprowadzić dla jakiego filtru.

15. Modyfikowanie Pakietów

Jeżeli pakiet został przefiltrowany i został oznaczony jaki filtr go dotyczy, może on być zmodyfikowany. Dzieje się to dzięki funkcji `modify(self, packet, filter_name)`. Działa ona w podobny sposób do funkcji filtrującej.

Słownik przechowujący modyfikacje(które należy wykonać na pakiecie) używa jako klucza nazwy filtru, do którego ma się odnosić. Tak więc pobierana jest ze słownika lista protokołów, które mają zostać zmodyfikowane i na ich podstawie edytowane są odpowiednie pola, wykorzystując `setattr` pozwalający na ustawienie wartości atrybutu tylko posiadając jego nazwę. W ten sposób możliwa jest zmiana wszystkich potrzebnych atrybutów bez znaczenia czy dany protokół został przewidziany podczas pisania programu.

16. Budowa interfejsu:

Proces budowy interfejsów bywa często żmudny i niedoceniany. Ze względu na wybór języka Python rozważano wybór jednej spośród trzech najpopularniejszych bibliotek graficznych dostępnych dlatego języka:

- PyQt5
- Tkinter
- Pyside

Ze względu na gotowe narzędzia do projektowania interfejsów takie jak np. `QtDesigner`, ostatecznie zastosowano bibliotekę PyQt5.

Poniższa lista przedstawia większość komponentów wykorzystanych do budowy graficznego interfejsu użytkownika:

a. **QApplication:**

Klasa **QApplication** zarządza przepływem sterowania i głównymi ustawieniami aplikacji. Dla każdej aplikacji graficznej wykorzystującej Qt istnieje dokładnie jeden obiekt **QApplication**, bez względu na to, czy aplikacja ma w danym momencie posiada 0, 1, 2 lub więcej okien.

b. **QWidget:**

Klasa **QWidget** jest podstawową klasą wszystkich obiektów interfejsu użytkownika. Widżet jest atomem interfejsu użytkownika: tzn. zawiera w sobie funkcje odbierania sygnałów od myszy, klawiatury i innych zdarzeń z systemu, a także maluje reprezentację samego siebie na ekranie.

c. **QLabel:**

Widżet **QLabel** zapewnia wyświetlanie danych tekstowych i graficznych. Nie są dostępne żadne funkcje interakcji z użytkownikiem. Wygląd etykiety można skonfigurować na różne sposoby. Pomimo, że ten komponent biblioteki Qt nie posiada zdefiniowanych funkcji/metod do interakcji z użytkownikiem, istnieje możliwość zdefiniowania takich interakcji przez programistę.

d. **QLineEdit:**

Widżet **QLineEdit** jest jednowierszowym edytorem tekstu. Edycja linii pozwala użytkownikowi na wprowadzanie i edycję pojedynczej linii zwykłego tekstu z przydatnym zbiorem funkcji edycyjnych, takich jak cofanie i edytowanie, wycinanie i wklejanie oraz przeciąganie czy upuszczanie.

e. **QPlainTextEdit**

Klasa **QPlainTextEdit** zapewnia widżet, który jest używany do edycji i wyświetlania zwykłego tekstu. **QPlainTextEdit** działa na paragrafach

i znakach. Akapit jest sformatowanym ciągiem znaków, który jest dopasowywany do szerokości widżetu. Domyślnie podczas czytania zwykłego tekstu, jeden nowy wiersz oznacza akapit.

Dokument składa się z co najmniej zeroowej liczby paragrafów. Paragrafy są oddzielone przerwami w wierszu. Każdy znak w paragrafie ma swoje własne atrybuty, na przykład czcionkę, kolor itp.

f. QPushButton

Widżet przycisków QPushButton zawiera dwa rodzaje przycisków zwykły przycisk lub przycisk poleceń. Przycisk zwykły lub przycisk poleceń jest prawdopodobnie najczęściej używanym widżetem w graficznym interfejsie użytkownika. Naciśnij (kliknij) na ten widżet, aby polecić komputerowi wykonanie jakiejś czynności lub przekazanie odpowiedzi na zadane pytanie.

Typowe przyciski to:

OK, Zastosuj, Anuluj, Zamknij, Tak, Nie, Pomoc czy Więcej informacji.

17. Użytkowanie aplikacji

Po uruchomieniu interfejsu użytkownik ma do dyspozycji wiele kart z różnymi funkcjonalnościami. Pierwsza z nich to krótki opis działania aplikacji z instrukcją obsługi. Użytkownik posiada możliwość przełączania się pomiędzy kartami za pomocą kliknięcia na daną kartę, bądź za pomocą przycisków *dalej* i *wstecz*.

Po kliknięciu w przycisk i uruchomieniu przechwytywania pakietów, tworzona jest kopia zapasowa tabeli iptables użytkownika i zapisywana do pliku tak, aby w przypadku nieprzewidzianego zamknięcia aplikacji, bądź samego systemu można było przywrócić prawidłową konfigurację reguł iptables systemu.

Następnie po utworzeniu backupu oryginalnych reguł dodawana jest nowa reguła iptables której zadaniem jest przekierowanie pakietów przychodzących z odpowiedniego NFQUEUE do aplikacji tak, by mogła z nich korzystać.

Próba zamknięcia aplikacji powoduje wykonanie automatycznego przywrócenia oryginalnych reguł iptables systemu, gdy reguły zostaną przywrócone aplikacja zostaje zamknięta. Przywrócenie backupu wykonywane jest również, gdy aplikacja zostanie zamknięta nagle przy użyciu przerwania systemowego np. CTRL+C.

18. Wygląd Interfejsu użytkownika

The screenshot shows the NetworkMiner application window. At the top, there is a navigation bar with tabs: 'Informacje' (selected), 'Przechwytywanie', 'Filtr', 'Modyfikatory', and 'Lista zmodyfikowanych'. Below the navigation bar is a table titled 'Pakiet' (Packet) which lists 19 captured network packets. The table has two columns: a numerical index column and a description column. The descriptions are as follows:

	Pakiet
1	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw
2	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw
3	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw
4	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw
5	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw
6	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw
7	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
8	IP / UDP / DNS Qry "b'BRWACD1B84F0D49.local.'"
9	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
10	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
11	IP / UDP / DNS Qry "b'BRWACD1B84F0D49.local.'"
12	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
13	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
14	IP / TCP 104.16.58.5:https > 192.168.1.14:45280 A
15	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
16	IP / UDP / DNS Qry "b'BRWACD1B84F0D49.local.'"
17	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
18	IP / UDP 192.168.1.11:netbios_ns > 192.168.1.255:netbios_ns / NBNSQueryRequest
19	IP / TCP 104.16.60.37:https > 192.168.1.14:58474 PA / Raw

At the bottom of the interface, there are four buttons: 'Powrót' (Back), 'Utwórz filtr na podstawie pakietu' (Create filter based on packet), 'Włącz przechwytywanie pakietów' (Enable packet capture), and 'Dalej' (Next).

Rys.1 Widok karty przechwytywanie.

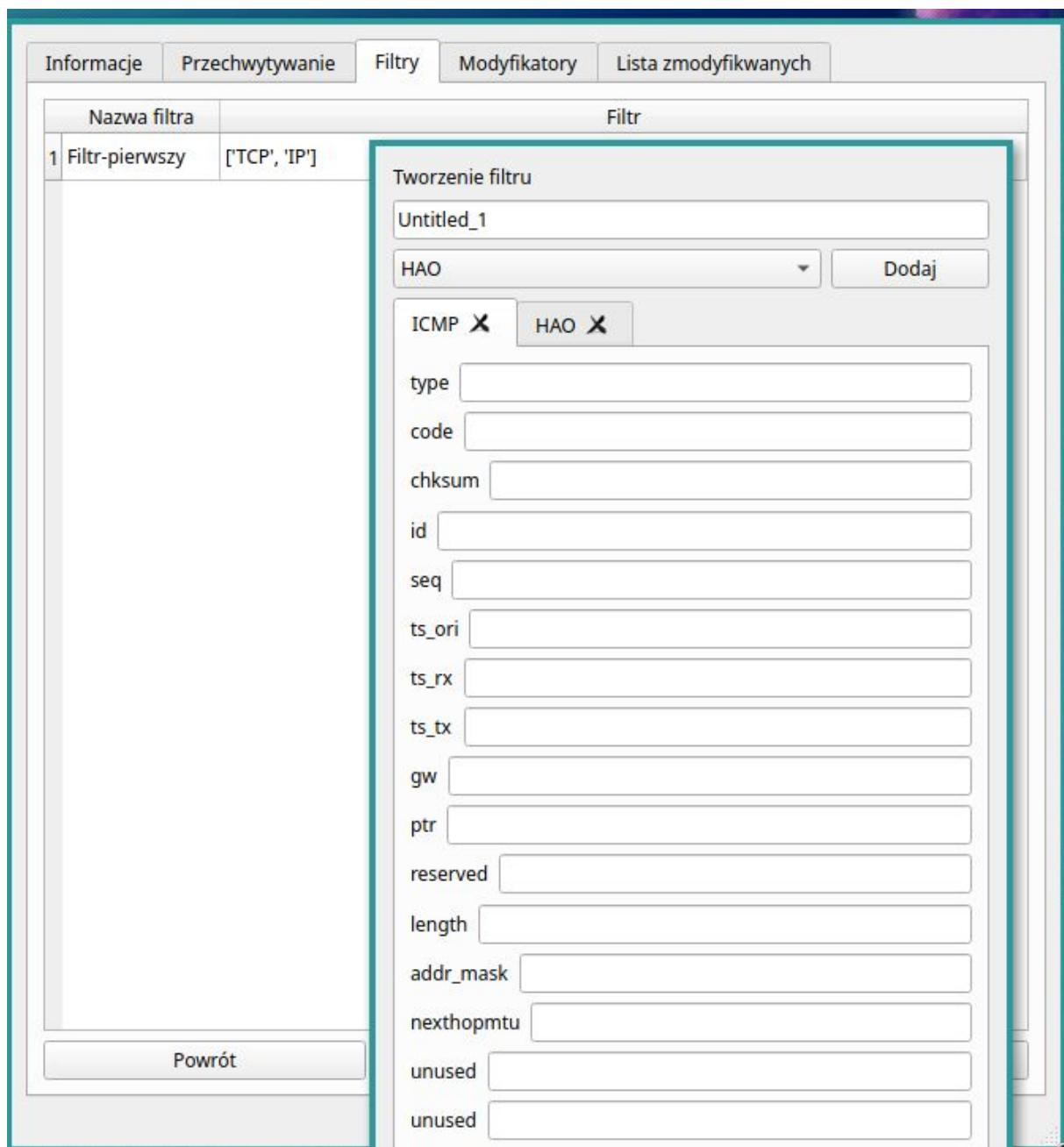
Interfejs użytkownika podzielony jest na pięć kart, które reprezentują typowy przebieg pracy użytkownika. Użytkownik może się pomiędzy nimi poruszać dzięki przyciskom "Dalej" oraz "Powrót" lub wskazywać bezpośrednio na wybrane karty aby przejść do nich poza ustaloną kolejnością. Pozwala to na bardzo intuicyjne poruszanie się po interfejsie.

Pierwszą kartą, którą użytkownik widzi po uruchomieniu interfejsu programu jest karta “Informacje” zawiera ona dokładny opis działania i obsługi programu.

Kolejna karta to “Przechwytywanie”, tutaj znajduje się przycisk do zarządzania przechwytywaniem pakietów. Większa część interfejsu zajęta jest przez okno wyświetlające listę pakietów przechwytywanych, które pojawiają się po kliknięciu w przycisk “Włącz przechwytywanie pakietów”. Pojedyncze pakiety pojawiają się w nowych liniach i posiadają skrót najważniejszych informacji na ich temat.

Celem było stworzenie swego rodzaju “liniowego” programu. Przyciski “Dalej” i “Powrót” przypominają te znane ze wszelkiej maści instalatorów. Ustanawia to pewnego rodzaju kolejność wykonywania czynności, co jest pomocne dla nowych użytkowników ale i także dla osób, które dobrze znają program i chcą szybko przejść przez wszystkie etapy pracy z programem.

Karty zostały dodatkowo zaimplementowane aby nie ograniczać użytkownika tylko do wybranej ścieżki działania, i umożliwić szybkie zmiany w funkcjonowaniu programu, ale także aby zaprezentować graficznie postęp, przez swego rodzaju workflow programu.



Rys.2 Widok karty Filtry.

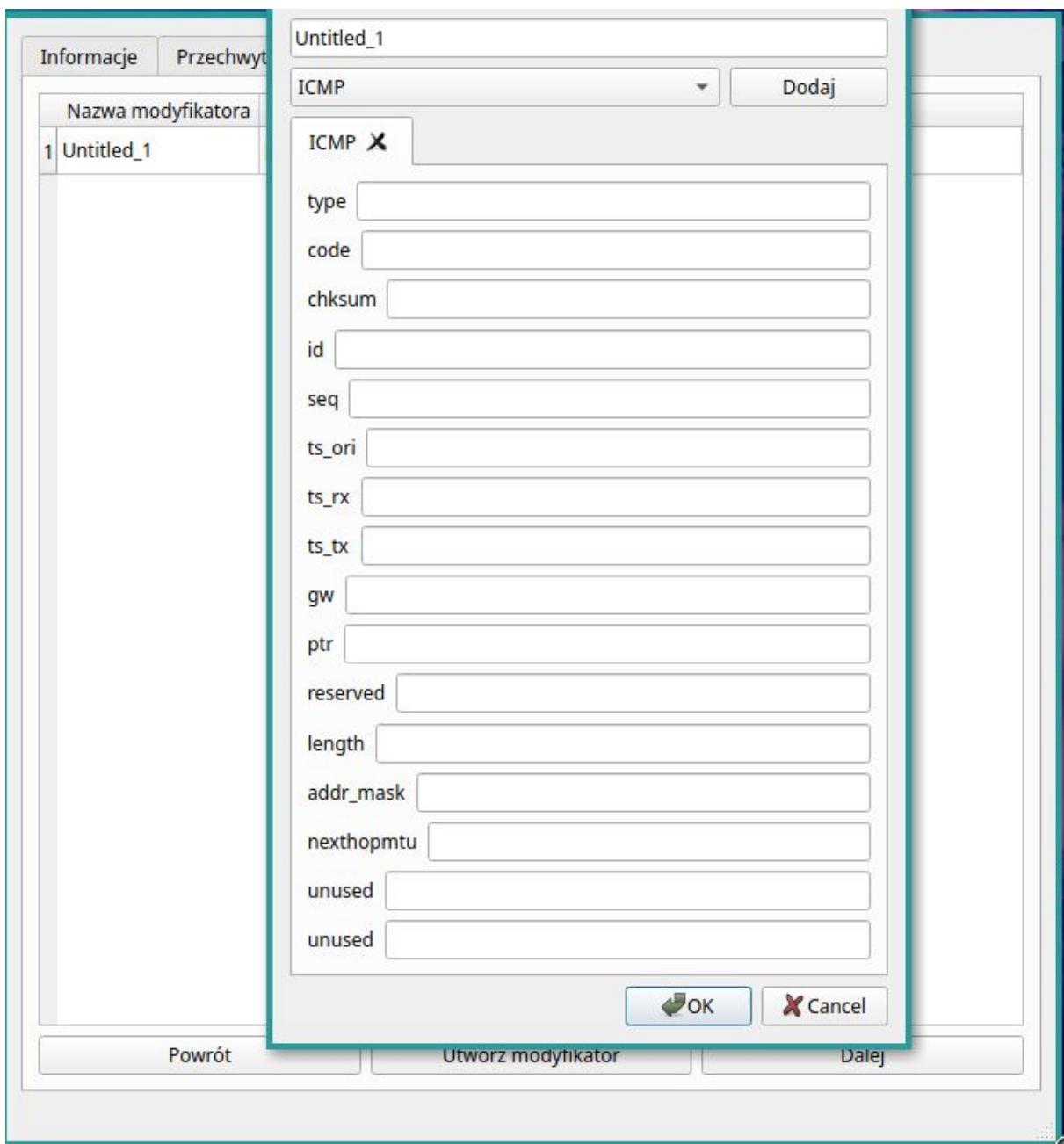
W karcie numer trzy mamy dostęp do systemu zarządzania filtrami. W tej karcie użytkownik ma możliwość tworzenia i przeglądania utworzonych filtrów. Na środku podobnie tak jak w poprzedniej karcie znajduje się tabela z listą filtrów, wyświetlająca nazwę oraz skrót protokołów, których dotyczy dany filtr. Przyciski u dołu ekranu podobnie jak w poprzednich kartach pozwalają na przechodzenie pomiędzy kartami.

Po kliknięciu w przycisk “Utwórz filtr” pokazuje się okno kreatora nowego filtra, gdzie możemy stworzyć nowy filtr. W tym oknie możemy wybrać nazwę i dodać protokoły według, których chcemy filtrować. Dokonujemy tego wybierając je z listy proceduralnie generowanych na podstawie tych, które są aktualnie wspierane przez bibliotekę Scapy.

Po wybraniu odpowiedniego protokołu tworzy się nowa karta z jego nazwą i dodają się wszystkie możliwe pola dla danego protokołu. Wpisując dane w pola możemy filtrować pakiety z wybranymi wartościami w konkretnych protokołach. Jeżeli pole pozostawimy puste, nie będzie ono miało wpływu na proces filtrowania.

Do pojedynczego filtru możemy dodawać wiele protokołów. Wszystkie protokoły w pojedynczym filtrze mają między sobą związek logiczny and. Natomiast pomiędzy osobnymi filtrami istnieje związek or. Pozwala to na tworzenie relatywnie skomplikowanych filtrów.

Po kliknięciu w przycisk potwierdzający, tworzony jest słownik, który jest później wykorzystywany w systemie filtrowania opisany w punkcie czternastym tego dokumentu. Po zakończeniu tworzenia filtr dodawany jest on do listy filtrów w interfejsie i od tego momentu obowiązuje on przekazując pakiety do systemu modyfikacji.



Rys.3 Widok karty Modyfikatory.

Karta “Modyfikatory” umożliwia tworzenie modyfikatorów. Przypomina ona interfejs do tworzenia filtrów jednak, działa na trochę innej zasadzie. Tworząc nowy modyfikator wykorzystujemy okno identyczne jak przy tworzeniu filtrów, różnicą jest tutaj fakt, że nazwa odnosi się do już istniejącego filtra, z którego pakiety będą poddawane modyfikacji. Wybieramy jakie pola w jakich warstwach będą poddawane zmianom i ich wartości. Pozostawiona pusta wartość spowoduje pominięcie danego pola, a w efekcie jego nie modyfikowanie.

Po kliknięciu w przycisk potwierdzenia tworzony jest podobny słownik jak przy systemie filtrów, jednak pola są używane do zmiany pól w pakietach a nie filtrowania.

19. Możliwości rozwoju

Interfejs programu jest w pełni responsywny i funkcjonalny, choć wciąż jest niedokończony posiadając drobne niedociągnięcia.

W dalszych pracach począwszy od tych najpilniejszych jakie należy wykonać znajduje się: poprawa opisów tabel czy dodanie możliwości podglądu, usuwania oraz edycji istniejących filtrów, a także modyfikatorów. Brakuje również funkcji tworzenia filtrów i modyfikatorów na podstawie istniejącego przechwyconego pakietu, czy podglądu samego pakietu.

Planuje się też zastąpienie w oknie kreatora modyfikatorów widżetu QLineEdit na QCombox tak, aby nazwę filtra wybierana była z dostępnej listy już istniejących filtrów. Rozbudowa programu Evilpostman w przyszłości ma umożliwić zapis przechwyconego ruchu do pliku w formacie pcap/pcapng.

Rozważano również zapewnienie funkcjonalności zapisu i odczytu z/do pliku wybranych filtrów i modyfikatorów. Jedną z kolejnych planowanych funkcjonalności jest możliwość skonfigurowania aplikacji tak, aby działała w trybie Fuzzer'a. Ma to na celu dostarczenie użytkownikom (przede wszystkim pentesterom i osobom zajmującym się bezpieczeństwem technologii internetowych) narzędzia pozwalającego na testowanie zabezpieczeń poprzez przesyłanie losowych, niepoprawnych, nadmiarowych lub niekompletnych danych we wskazane miejsce (np. do routera, aplikacji, strony www itd.) na podstawie wybranego protokołu czy pakietu danych.

Najtrudniejszą planowaną funkcjonalnością jest możliwość wyłuskiwania konkretnych danych przesyłanych przez sieć (pliki, dokumenty, grafiki, strony www itp.) podobnie jak w przypadku programów typu *Network Miner*.

Dokumentacja projekt będzie rozwijana wraz ze zmianami i poprawkami jakie będą wprowadzane do aplikacji Evilpostman. Jest to dokument przedstawiający pierwszą działającą wersję aplikacji, lecz wciąż wymagającą wprowadzenia wielu poprawek i funkcjonalności.

20. Repozytorium

Repozytorium:

https://github.com/jaroslaw-wieczorek/Modifing_Packets_On-The-Fly_In_SCAPY

21. Bibliografia

<https://sekurak.pl/generator-pakietow-scapy/>

<https://helion.pl/ksiazki/black-hat-python-jazyk-python-dla-hakerow-i-pentesterow-justin-seitz.blahap.htm#format/d>

<http://scapy.readthedocs.io/en/latest/usage.html>

<https://theitgeekchronicles.files.wordpress.com/2012/05/scapyguide1.pdf>

<https://helion.pl/ksiazki/zapory-sieciowe-w-systemie-linux-kompendium-wiedzy-o-nftables-wydanie-iv-steve-suehring.zasili.htm#format/d>

<https://wiki.archlinux.org>