# SCHT - Lab3

Stanisław Kwiatkowski, Bartosz Ziemba

30 marca 2024

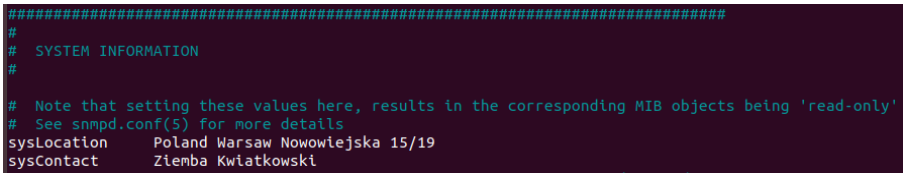
## Spis treści



## 1. 4.3.R1

SNMDP - (Simple Network Management Protocol Daemon) - jest to proces uruchamiany w tle na urządzeniu zarządzanym (agent). Nasłuchuje na żądania wysyłane przez menagera protokołem UDP na porcie 161, odpowiada na nie na porcie 162. W pliku snmpd.conf możemy przypisać pewne wartości, o które manager będzie mógł się "zapytać". Możemy też zdefiniować zdarzenia, w których agent wyśle pewne informacje, po spełnieniu określonych warunków.

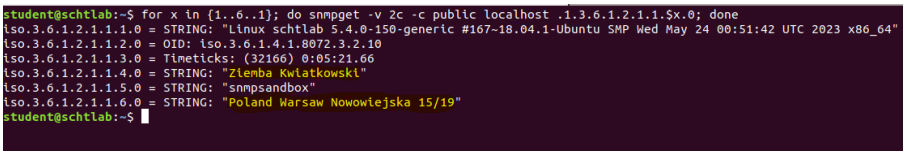
Ustawiliśmy wartości obiektów sysContact na nasze nazwiska, a sysLocation na adres naszego wydziału.




Rys. 1: Dodane elementy konfiguracji snmpd

## 2. 5.1.R1

Po wypisaniu 6 kolejnych wartości węzłów, będących obiektami skalarnymi, widzimy wpisane przez nas do konfiguracji dane.




Rys. 2: Wynik 6 operacji SNMP typu GET

## 3. 5.1.R2

Definicje obiektów będących bezpośrednimi dziećmi węzła system:

```
sysDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A textual description of the entity.  This value
            should include the full name and version
            identification of the system's hardware type,
            software operating-system, and networking
            software.  It is mandatory that this only contain
            printable ASCII characters."
    ::= { system 1 }

sysObjectID OBJECT-TYPE
    SYNTAX  OBJECT IDENTIFIER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The vendor's authoritative identification of the
            network management subsystem contained in the
            entity.  This value is allocated within the SMI
            enterprises subtree (1.3.6.1.4.1) and provides an
            easy and unambiguous means for determining 'what
            kind of box' is being managed.  For example, if
            vendor 'Flintstones, Inc.' was assigned the
            subtree 1.3.6.1.4.1.4242, it could assign the
            identifier 1.3.6.1.4.1.4242.1.1 to its 'Fred
            Router'."
    ::= { system 2 }
```

```
sysUpTime OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The time (in hundredths of a second) since the
            network management portion of the system was last
            re-initialized."
    ::= { system 3 }

sysContact OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The textual identification of the contact person
            for this managed node, together with information
            on how to contact this person."
    ::= { system 4 }

sysName OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "An administratively-assigned name for this
            managed node.  By convention, this is the node's
            fully-qualified domain name."
    ::= { system 5 }

sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The physical location of this node (e.g.,
            `telephone closet, 3rd floor')."
    ::= { system 6 }

sysServices OBJECT-TYPE
    SYNTAX  INTEGER (0..127)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A value which indicates the set of services that
            this entity primarily offers.

            The value is a sum.  This sum initially takes the
            value zero, Then, for each layer, L, in the range
            1 through 7, that this node performs transactions
            for, 2 raised to (L - 1) is added to the sum.  For
            example, a node which performs primarily routing
            functions would have a value of 4 (2^(3-1)).  In
            contrast, a node which is a host offering
            application services would have a value of 72
```

```
        (2^(4-1) + 2^(7-1)).  Note that in the context of
        the Internet suite of protocols, values should be
        calculated accordingly:

             layer  functionality
                 1  physical (e.g., repeaters)
                 2  datalink/subnetwork (e.g., bridges)
                 3  internet (e.g., IP gateways)
                 4  end-to-end  (e.g., IP hosts)
                 7  applications (e.g., mail relays)

        For systems including OSI protocols, layers 5 and
        6 may also be counted."
    ::= { system 7 }
```

## 4. 5.1.R3

Wartości różnią się od siebie, ponieważ, jak mówi opis obiektu SysUpTime, jest to czas w milisekundach od momentu uruchomienia usługi. Jako, że wywołujemy kolejne zapytnaia GET w pewnych odstępach czasowych, zwracana wartość będzie się różniła właśnie o te wartości czasowe.

```
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (213394) 0:35:33.94
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (213713) 0:35:37.13
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (213912) 0:35:39.12
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (214085) 0:35:40.85
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (214192) 0:35:41.92
```

Rys. 3: Kolejne zapytania o SysTimeUp

## 5. 5.1.R4

### 5.1. get-request

#### 5.1.1. Poziom UDP

```
User Datagram Protocol, Src Port: 45960, Dst Port: 161
    Source Port: 45960
    Destination Port: 161
    Length: 51
    Checksum: 0xfe46 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 30]
```

Struktura poziomu UDP jest płaska i składa się z pól określających po kolei:

- portu, z którego wysłano zapytanie (45960),
- portu, z którego wysłano zapytanie, portu, na który zapytanie ma trafić (161),
- długości pakietu (51)
- sum kontrolnych (0xfe46, niezweryfikowana)

#### 5.1.2. Poziom SNMP

```
Simple Network Management Protocol
    version: v2c (1)
    community: public
    data: get-request (0)
        get-request
            request-id: 786601973
            error-status: noError (0)
            error-index: 0
            variable-bindings: 1 item
                1.3.6.1.2.1.1.3.0: Value (Null)
```

Struktura pakietu na poziomie protokołu SNMP nie jest płaska, zawiera zagnieżdzone elementy.

- wersja protokołu (v2c)
- community string - ten parametr jest swoistym hasłem, które pozwala na odczyt i ewentualnie zapis danych w urządzeniu,
- parametr określający, że jrst to zapytanie (get-request),
- id zapytania (786601973), dzięki któremu można chociażby rejestrować i logować zapytania,
- pole error-status określające fakt wystąpienia ewentualnego błędu (noError),
- pole error-index określające miejsce wystąpienia ewentualnego błędu (0),
- variable-bindings - w tym polu podany jest obiekt o OID elementu o który odpytujemy, posiada wartość Null, która zmieni się w *get-response*.

### 5.2. get-response

#### 5.2.1. Poziom UDP

```
User Datagram Protocol, Src Port: 161, Dst Port: 45960
    Source Port: 161
    Destination Port: 45960
    Length: 54
    Checksum: 0xfe49 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 30]
```

Struktura poziomu UDP get-response jest niemal identyczna jak w przypadku get-reques:

- portu, z którego wysłano zapytanie (161),
- portu, z którego wysłano zapytanie, portu, na który zapytanie ma trafić (45960),
- długości pakietu (54),
- sum kontrolnych (0xfe46, niezweryfikowana).

Numery portów zamieniły się względem wcześniejszego zapytania.

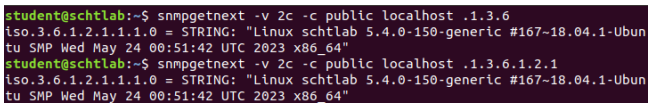
#### 5.2.2. Poziom SNMP

```
Simple Network Management Protocol
    version: v2c (1)
    community: public
    data: get-response (2)
        get-response
            request-id: 786601973
            error-status: noError (0)
            error-index: 0
            variable-bindings: 1 item
                1.3.6.1.2.1.1.3.0: 214192
                    Object Name: 1.3.6.1.2.1.1.3.0 (iso.3.6.1.2.1.1.3.0)
                    Value (Timeticks): 214192
```

Dzięki temu, że zarówno get-request i get-response są "owiniętę" w ten sam protokół, ich struktura pozostanie taka sama.

- wersja protokołu (v2c),
- community string - taki sam jak w zapytaniu
- id zapytania (786601973), takie samo jak w zapytnaiu
- pole data, które w tym przypadku określa fakt, że jest to odpowiedź (get-response),
- pole error-status, tak samo jak w zapytaniu,
- pole error-index, tak samo jak w zapytaniu,
- variable-bindings - w przeciwieństwie do zapytania, tutaj pojawiła nam się konkretna wartość obiektu i w tym przypadku jest to czas działania usługi SysUpTime.

## 6. 5.2.R1




Rys. 4: Wywołanie polecenia SMTP GET-NEXT dla podanych wartości węzłów OID

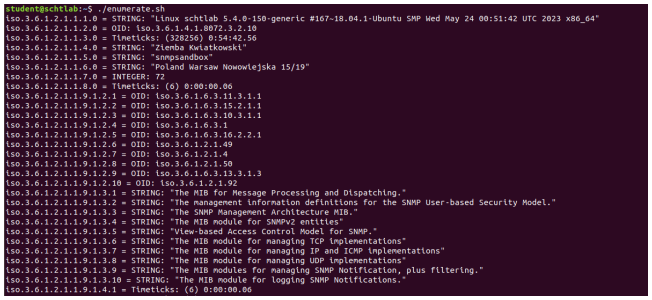
## 7. 5.2.R2

Wyniki obydwu zapytań są takie same, ponieważ polecenie GET-NEXT zwraca wartość **następnego, niepustego** węzła występującego po podanym w argumencie. Następnym niepustym węzłem po zarówno .1.3.6 oraz .1.3.6.1.2.1 jest węzeł o OID = 1.3.6.1.2.1.1.1.0. Wszystkie potencjalne węzły znajdujące się pomiędzy .1.3.6 a .1.3.6.1.2.1 są puste. Dodatkowo, jest to pierwszy węzeł, więc na pewno nie istnieje żaden inny, który mógłby się okazać "kolejnym" dla .1.3.6.

## 8. 5.2.R3

Wykonując wiele operacji GET-NEXT i podając jako argumenty OID wartości uzyskanej w poprzedniej iteracji możemy zbudować narzędzie pozwalające nam enumerować wszystkie zdefiniowane i niepuste obiekty. Taka rekursywna funkcja jest zaimplementowana w SNMP i nosi nazwę SNMP-WALK. Nasza customowa implementacja takiej funkcji znajduje się poniej.

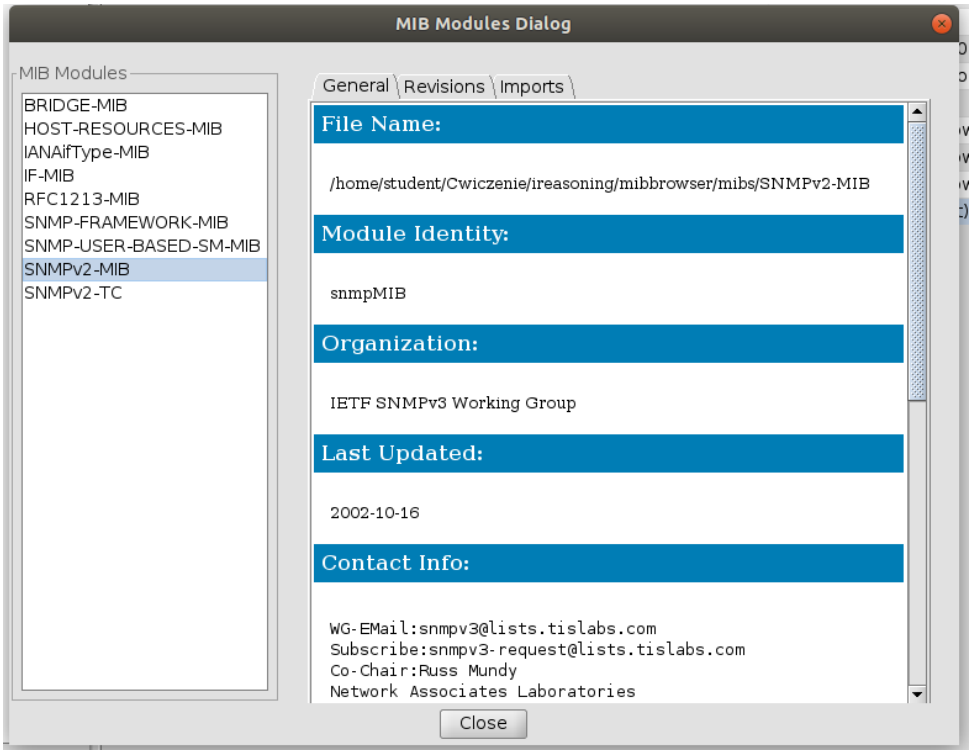
```
host="localhost"
community="public"
oid=".1"

while true; do
    result=$(snmpgetnext -v 2c -c "$community" "$host" "$oid")
    if [ -z "$result" ]; then
        break
    fi
    next_oid=$(echo "$result" | awk '{print $1}')
    echo "$result"
    oid="$next_oid"
done
```




Rys. 5: Efekt uruchomienia customowego SNMP-WALK

## 9. 6.2 R1

Po uruchomieniu przeglądarki w trybie interfejsu graficznego mamy dostęp do modułów MIB. Z którego każdy opisuje jakąś część/funkcje sieci.




Rys. 6: Modułów MIB

## 10. 6.3 R1

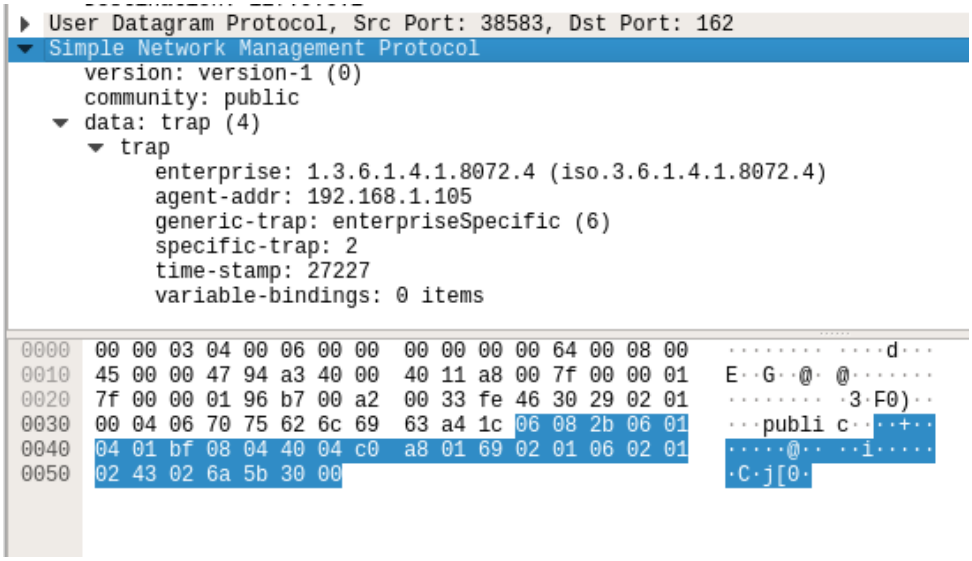
W sekcji Trap Receiver widzimy jak zarówno zamykanie oraz uruchamiania serwisu SNMPD generuje wiadomości typu trap o odpowiednich nazwach i parametrami, które możemy zobaczyć po przechwyceniu pakietu w Wiresharku. W ogólności wiadomość ta przenosi informacje o błędach, awariach oraz nie prawidłowościach w działaniu sieci.

| Description | Source | Time |
|---|---|---|
| coldStart | 192.168.1.105 | 2023-11-30 14:35:32 |
| Specific: 2; .1.3.6.1.4.1.8072.4 | 192.168.1.105 | 2023-11-30 14:35:13 |

Rys. 7: Zawartość pola Trap Receiver

| | | | | | |
|---|---|---|---|---|---|
| **Source:** | 192.168.1.105 | **Timestamp:** | 37 minutes 31.64 seconds | **SNMP Version:** | 1 |
| **Enterprise:** | .1.3.6.1.4.1.8072.4 | | | **Community:** | public |
| **Specific:** | 2 | **Generic:** | enterpriseSpecific | | |
| **Description:** | | | | | |

Rys. 8: Podgląd wiadomości trap



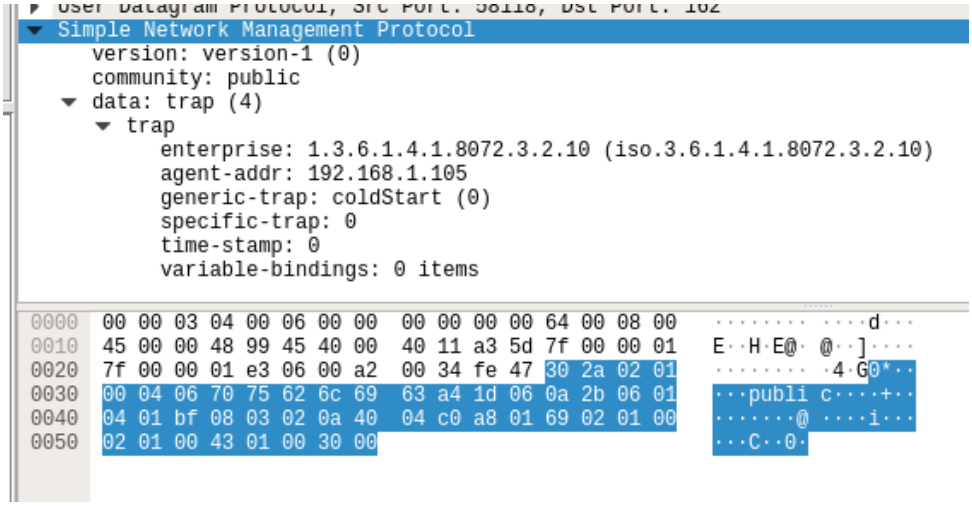

Rys. 9: Podgląd pakietu w Wiresharku

Jak widzimy zamknięcie usługi powoduje wygenerowanie wiadomości trap typu "enterpriseSpecific (6)". Jest to specyficzny typ wiadomości, stworzony na potrzeby naszej sieci przez jej projektanta. Dokładna struktura wiadomości wygląda następująco:

- enterprise: 1.3.6.1.4.1.8072.4 -> Identyfikator obiektu, które utworzyło tą wiadomość
- agent-addr: 192.168.1.105 -> adres IP agenta, który wysłał ten trap
- generic-trap: enterpriseSpecific (6) -> rodzaj wiadomości
- specific-trap: identyfikator konkretnej kategorii, sugerującej rodzaj zdarzenia, które wystąpiło
- time-stamp: 27227 -> znacznik w czasie, liczony od początku działania usługi, można go interpretować jako czas działania.
- variable-bindings: 0 items-> zmienne powiązane z trapem, w naszym przypadku brak takich zmiennych.

| | | | | | |
|---|---|---|---|---|---|
| **Source:** | 192.168.1.105 | **Timestamp:** | 0 millisecond | **SNMP Version:** | 1 |
| **Enterprise:** | .1.3.6.1.4.1.8072.3.2.10 | | | **Community:** | public |
| **Specific:** | 0 | **Generic:** | coldStart | | |
| **Description:** | coldStart | | | | |

Rys. 10: Podgląd wiadomości trap

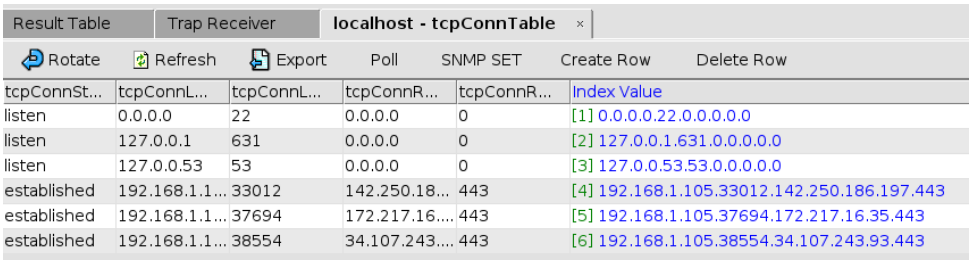



Rys. 11: Podgląd pakietu w Wiresharku

Natomiast uruchomienie usługi powoduje rozesłanie wiadomości trap typu "coldStart (0)", który jest używany do poinformowania o rozpoczęciu działania usługi SNMP. Struktura wiadomości jest tożsama z pierwszym trapem, a jedne różnice zachodzą w

- generic-trap: coldStart (0)
- specific-trap: 0
- time-stamp: 0 -> 0 ponieważ dopiero uruchamiamy usługę

## 11. 6.4 R1




Rys. 12: Tabela połączeń TCP

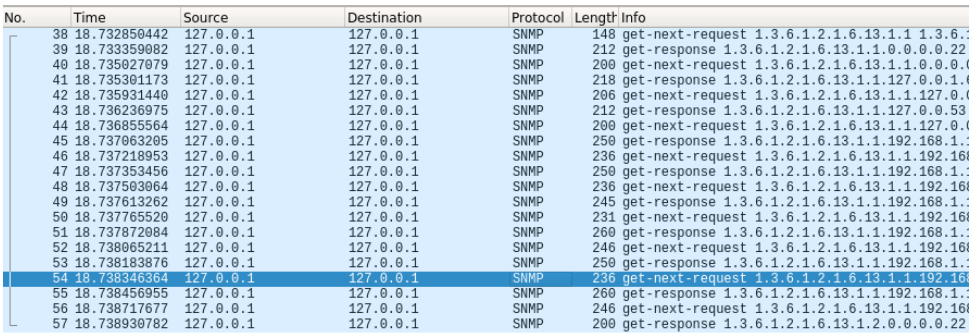
## 12. 6.4 R2

Tablica ta ukazuje utworzone połączenia TCP w tej sieci wraz z ich parametrami. Jej atrybuty można opisać opisać jako:

- tcpConnState -> status połączenia
- tcpConnLocalAddress -> adress IP hosta
- tcpConnLocalPort -> port na, którym działa połączenia u hosta
- tcpConnRemoteAddress -> adress IP serwera
- tcpConRemPort -> port na serwerze obsługujący połączenie

## 13. 6.4 R3

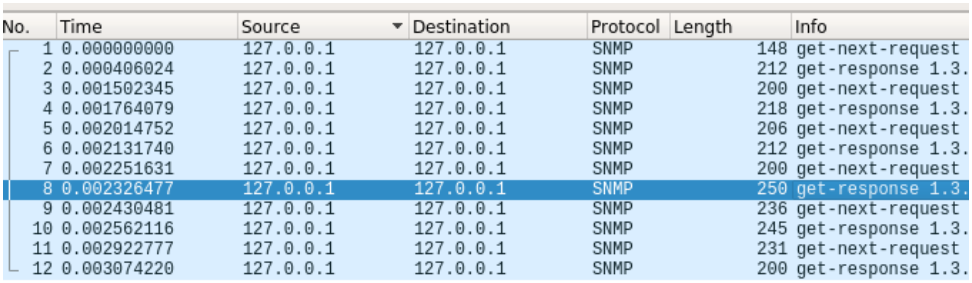
By dowiedzieć się w jaki sposób przeglądarka odczytała dana wyświetlone w tabli skorzystaliśmy z wiresharka, by przechwycić ruch sieciowy.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 38 | 18.732850442 | 127.0.0.1 | 127.0.0.1 | SNMP | 148 | get-next-request 1.3.6.1.2.1.6.13.1.1 1.3.6. |
| 39 | 18.733359082 | 127.0.0.1 | 127.0.0.1 | SNMP | 212 | get-response 1.3.6.1.2.1.6.13.1.1.0.0.0.0.22 |
| 40 | 18.735027079 | 127.0.0.1 | 127.0.0.1 | SNMP | 200 | get-next-request 1.3.6.1.2.1.6.13.1.1.0.0.0.0 |
| 41 | 18.735301173 | 127.0.0.1 | 127.0.0.1 | SNMP | 218 | get-response 1.3.6.1.2.1.6.13.1.1.127.0.0.1.6 |
| 42 | 18.735931440 | 127.0.0.1 | 127.0.0.1 | SNMP | 206 | get-next-request 1.3.6.1.2.1.6.13.1.1.127.0.0 |
| 43 | 18.736236975 | 127.0.0.1 | 127.0.0.1 | SNMP | 212 | get-response 1.3.6.1.2.1.6.13.1.1.127.0.0.53 |
| 44 | 18.736855564 | 127.0.0.1 | 127.0.0.1 | SNMP | 200 | get-next-request 1.3.6.1.2.1.6.13.1.1.127.0.0 |
| 45 | 18.737063205 | 127.0.0.1 | 127.0.0.1 | SNMP | 250 | get-response 1.3.6.1.2.1.6.13.1.1.192.168.1.1 |
| 46 | 18.737218953 | 127.0.0.1 | 127.0.0.1 | SNMP | 236 | get-next-request 1.3.6.1.2.1.6.13.1.1.192.168 |
| 47 | 18.737353456 | 127.0.0.1 | 127.0.0.1 | SNMP | 250 | get-response 1.3.6.1.2.1.6.13.1.1.192.168.1.1 |
| 48 | 18.737503064 | 127.0.0.1 | 127.0.0.1 | SNMP | 236 | get-next-request 1.3.6.1.2.1.6.13.1.1.192.168 |
| 49 | 18.737613262 | 127.0.0.1 | 127.0.0.1 | SNMP | 245 | get-response 1.3.6.1.2.1.6.13.1.1.192.168.1.1 |
| 50 | 18.737765520 | 127.0.0.1 | 127.0.0.1 | SNMP | 231 | get-next-request 1.3.6.1.2.1.6.13.1.1.192.168 |
| 51 | 18.737872084 | 127.0.0.1 | 127.0.0.1 | SNMP | 260 | get-response 1.3.6.1.2.1.6.13.1.1.192.168.1.1 |
| 52 | 18.738065211 | 127.0.0.1 | 127.0.0.1 | SNMP | 246 | get-next-request 1.3.6.1.2.1.6.13.1.1.192.168 |
| 53 | 18.738183876 | 127.0.0.1 | 127.0.0.1 | SNMP | 250 | get-response 1.3.6.1.2.1.6.13.1.1.192.168.1.1 |
| 54 | 18.738346364 | 127.0.0.1 | 127.0.0.1 | SNMP | 236 | get-next-request 1.3.6.1.2.1.6.13.1.1.192.16 |
| 55 | 18.738456955 | 127.0.0.1 | 127.0.0.1 | SNMP | 260 | get-response 1.3.6.1.2.1.6.13.1.1.192.168.1.1 |
| 56 | 18.738717677 | 127.0.0.1 | 127.0.0.1 | SNMP | 246 | get-next-request 1.3.6.1.2.1.6.13.1.1.192.168 |
| 57 | 18.738930782 | 127.0.0.1 | 127.0.0.1 | SNMP | 200 | get-response 1.3.6.1.2.1.6.13.1.2.0.0.0.0.22 |

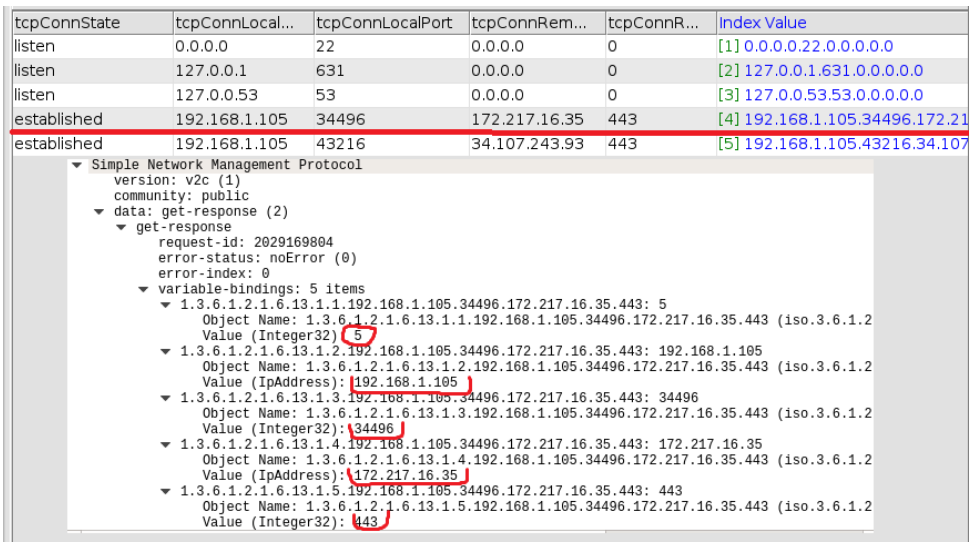
Rys. 13: Pozyskiwanie informacji o obiekcie tcpConnTable

Jak widać przeglądarka wysyłała serie zapytań typu GET-NEXT do kolejnych obiektów, które zwracają wartości zapisywane do rekordów tablicy. Co możemy zobaczyć w treści odpowiedzi na wysłane żądanie.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 127.0.0.1 | 127.0.0.1 | SNMP | 148 | get-next-request |
| 2 | 0.000406024 | 127.0.0.1 | 127.0.0.1 | SNMP | 212 | get-response 1.3. |
| 3 | 0.001502345 | 127.0.0.1 | 127.0.0.1 | SNMP | 200 | get-next-request |
| 4 | 0.001764079 | 127.0.0.1 | 127.0.0.1 | SNMP | 218 | get-response 1.3. |
| 5 | 0.002014752 | 127.0.0.1 | 127.0.0.1 | SNMP | 206 | get-next-request |
| 6 | 0.002131740 | 127.0.0.1 | 127.0.0.1 | SNMP | 212 | get-response 1.3. |
| 7 | 0.002251631 | 127.0.0.1 | 127.0.0.1 | SNMP | 200 | get-next-request |
| 8 | 0.002326477 | 127.0.0.1 | 127.0.0.1 | SNMP | 250 | get-response 1.3. |
| 9 | 0.002430481 | 127.0.0.1 | 127.0.0.1 | SNMP | 236 | get-next-request |
| 10 | 0.002562116 | 127.0.0.1 | 127.0.0.1 | SNMP | 245 | get-response 1.3. |
| 11 | 0.002922777 | 127.0.0.1 | 127.0.0.1 | SNMP | 231 | get-next-request |
| 12 | 0.003074220 | 127.0.0.1 | 127.0.0.1 | SNMP | 200 | get-response 1.3. |

Rys. 14: Proces pobierania danych do tabeli



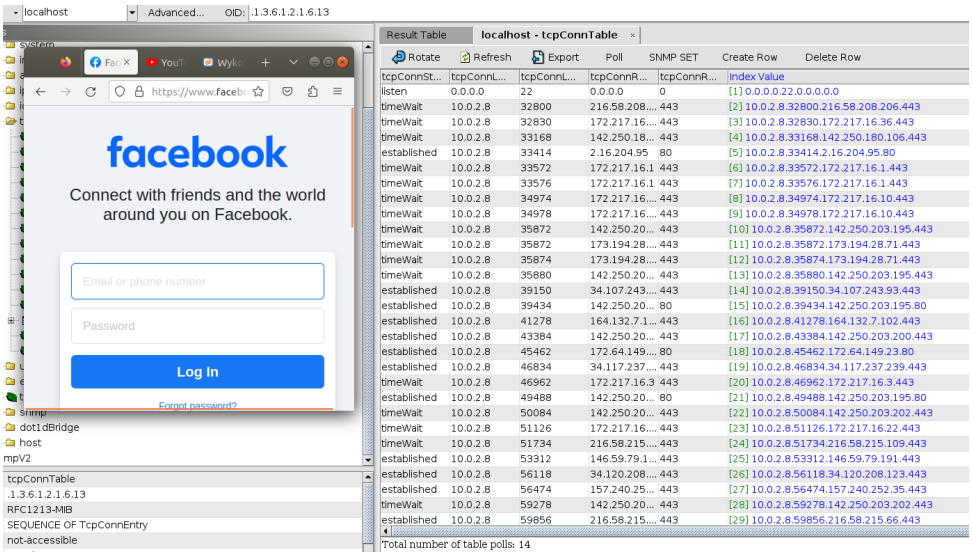
| tcpConnState | tcpConnLocal... | tcpConnLocalPort | tcpConnRem... | tcpConnR... | Index Value |
|---|---|---|---|---|---|
| listen | 0.0.0.0 | 22 | 0.0.0.0 | 0 | [1] 0.0.0.0.22.0.0.0.0.0 |
| listen | 127.0.0.1 | 631 | 0.0.0.0 | 0 | [2] 127.0.0.1.631.0.0.0.0.0 |
| listen | 127.0.0.53 | 53 | 0.0.0.0 | 0 | [3] 127.0.0.53.53.0.0.0.0.0 |
| established | 192.168.1.105 | 34496 | 172.217.16.35 | 443 | [4] 192.168.1.105.34496.172.21 |
| established | 192.168.1.105 | 43216 | 34.107.243.93 | 443 | [5] 192.168.1.105.43216.34.107 |

```
Simple Network Management Protocol
    version: v2c (1)
    community: public
  data: get-response (2)
    get-response
        request-id: 2029169804
        error-status: noError (0)
        error-index: 0
      variable-bindings: 5 items
        1.3.6.1.2.1.6.13.1.1.192.168.1.105.34496.172.217.16.35.443: 5
            Object Name: 1.3.6.1.2.1.6.13.1.1.192.168.1.105.34496.172.217.16.35.443 (iso.3.6.1.2
            Value (Integer32): 5
        1.3.6.1.2.1.6.13.1.2.192.168.1.105.34496.172.217.16.35.443: 192.168.1.105
            Object Name: 1.3.6.1.2.1.6.13.1.2.192.168.1.105.34496.172.217.16.35.443 (iso.3.6.1.2
            Value (IpAddress): 192.168.1.105
        1.3.6.1.2.1.6.13.1.3.192.168.1.105.34496.172.217.16.35.443: 34496
            Object Name: 1.3.6.1.2.1.6.13.1.3.192.168.1.105.34496.172.217.16.35.443 (iso.3.6.1.2
            Value (Integer32): 34496
        1.3.6.1.2.1.6.13.1.4.192.168.1.105.34496.172.217.16.35.443: 172.217.16.35
            Object Name: 1.3.6.1.2.1.6.13.1.4.192.168.1.105.34496.172.217.16.35.443 (iso.3.6.1.2
            Value (IpAddress): 172.217.16.35
        1.3.6.1.2.1.6.13.1.5.192.168.1.105.34496.172.217.16.35.443: 443
            Object Name: 1.3.6.1.2.1.6.13.1.5.192.168.1.105.34496.172.217.16.35.443 (iso.3.6.1.2
            Value (Integer32): 443
```

Rys. 15: Porównanie pakietu przechwycone przez Wireshark z tablicą w przeglądarce

## 14. 6.4 R4

Najprostszym sposobem na otwarcie wielu połączeń TCP, jest otwarcie dowolnej strony WWW. Aby została ona wyświetlona, host musi utworzyć wiele sesji połączeń TCP, aby pobrać różne pliki, wykonać zapytania, uwierzytelnić się itd. Im więcej danych będziemy chcieli wyświetlić tym więcej zapytań będzie potrzebnych, więc dla lepszego efektu otworzyliśmy kilka stron internetowych tuż po sobie. Co więcej, gdy strony się załadują a różne niewidzialne dla użytkownika przekierowania i połączenia dodatkowe zakończą, ilość wierszy w tabeli obniża się, eliminując zakończone połączenia.




Rys. 16: Zwiększona liczba sesji TCP po rozpoczęciu przeglądania Internetu

## 15. 6.5 R1

Tłumacząc opis na Polski można wywnioskować, że $ifNumber$ określa liczbę dostępnych interfejsów w systemie niezależnie od ich obecnego stanu.

| | |
|---|---|
| Name | ifNumber |
| OID | .1.3.6.1.2.1.2.1 |
| MIB | RFC1213-MIB |
| Syntax | INTEGER |
| Access | read-only |
| Status | mandatory |
| DefVal | |
| Indexes | |
| Descr | The number of network interfaces (regardless of their current state) present on this system. |

Rys. 17: Definicja obiektu ifNumber

## 16. 6.5 R2

| ifIndex | ifDescr | ifType | ifMtu | ifSpeed | ifPhysAddress |
|---|---|---|---|---|---|
| 1 | lo | softwareLoo... | 65536 | 10000000 | |
| 2 | Intel Corpor... | ethernetCs... | 1500 | 1000000000 | 08-00-27-55-64-B5 |

Rys. 18: Tablica interfejsów

Jej kolumny można opisać w następujący sposób

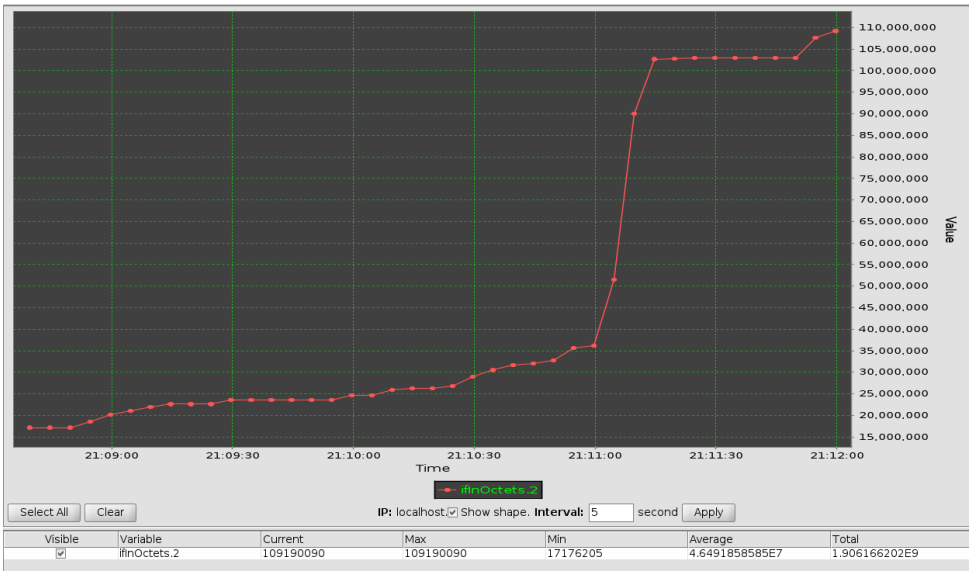
- ifDescr - krótki tekstowy opis interfejsu, powinien zawierać nazwę producenta, nazwę produktu oraz jego wersje. Jak widać u nas jeden z interfejsów jest interfejsem wirtualnym (lo), a drugi wyprodukowany przez firmę Intel.
- ifType - określa typ protokółu pod warstwą sieciową. U nas jest to odpowiednio Loopback oraz CSMA/CD.
- ifMtu - rozmiar największego pakietu, który może zostać wysłany/odebrany przez ten interface.
- ifSpeed - szacowana dostępna przepływność interfejsu, wartość podawana w bitach na sekunde.
- ifPhysAddress - fizyczny adress interfejsu, dla localhosta nie istnieje ponieważ nie jest to interfejs fizyczny

## 17. 6.5 R3

- ifAdminStatus - flaga wskazująca jaki "powinien być" (czego oczekuje zarządca sieci) stan interfejsu. Przyjmuje trzy stany
  - 1 - up, oznacza, że interfejs jest aktywny
  - 2 - down, oznacza, że interfejs jest wyłączony
  - 3 - testing, oznacza, że interfejs jest testowany i żadne "operacyjne" pakiety nie mogą przejść
- ifOperStatus - flaga wskazująca jaki jest rzeczywisty stan interfejsu. Przyjmuje siedem stanów, z czego pierwsze 3 są tożsame z omawianymi wcześniej:
  - 1 - up
  - 2 - down
  - 3 - testing
  - 4 - unknown, stan niezany
  - 5 - dormant, uruchomiony ale posiadający ograniczeni na to jaki ruch może przetważać
  - 6 - notPresent, interfejs nie jest obecny fizycznie
  - 7 - lowerLayerDown, brak niższej warstwy, może to oznaczać awarie medium transportowego.
- iflnOctects - liczba oktetów (bajtów) otrzymanych przez ten interface.

## 18. 6.5 R4

Aby spowodować znaczne zmiany na wykresie, włączyliśmy test prędkości internetu, który naturalnie pobiera dane z jak największą prędkością. Tym samym sprawiliśmy, że wartość funkcji na wykresie w pewnym punkcie czasu gwałtownie rośnie.




Rys. 19: Ładny przebieg wzrostu sumy pobranych oktetów

## 19. Podsumowanie

Laboratorium pozwoliło nam zrozumieć istotę protokołu SNMP i jego roli w zarządzaniu sieciami. Ustaliliśmy wartości dla różnych obiektów, jak sysContact czy sysLocation, umożliwiając menadżerowi ich odczyt. Pobierając informacje za pomocą operacji GET, zdobyliśmy wiedzę o opisie systemu i czasie jego działania. Badając obiekty SNMP, w tym tablice połączeń TCP czy informacje o interfejsach, zgłębiliśmy praktyczne zastosowania monitorowania i zarządzania siecią. Analiza zmian w ruchu sieciowym pozwoliła nam lepiej zrozumieć wpływ aktywności na przesyłane dane, co przybliżyło nas do praktycznego wykorzystania SNMP w kontrolowaniu infrastruktury sieciowej.