Laboratorium 4. SNMP v.2 i v.3

Laboratorium to ma za zadanie przedstawienie zmian jakie zaszły w nowszych wersjach protokołu SNMP oraz zmianach w sposobie obsługi narzędzi i agentów. Dodatkowo przedstawiona zostanie częściowa konfiguracja agenta pod systemem Linux.

Celem zajęć jest skonfigurowanie agenta SNMP, w taki sposób aby wykorzystywał uwierzytelnianie zastosowane w wersji trzeciej protokołu. A także wykorzystanie poznanych wcześniej narzędzi do odczytania danych używając tej metody uwierzytelniania. Dodatkowo należy się zapoznać z nowymi narzędziami, które wspierają nowsze wersje protokołu.

1. Wprowadzenie

W 1992 roku zaczęto pracować nad poprawą bezpieczeństwa protokołu oraz rozszerzeniem jego możliwości tak, aby mógł kontrolować dowolne zasoby, również te nie związane z sieciami. W 1996 roku ustandaryzowano nową wersję protokółu nazwaną SNMPv2 lub SNMPv2c, drugi skrót pochodzi z określenia sposobu autoryzacji (Community-Based SNMPv2). Niestety wersja ta nie poprawia aspektów bezpieczeństwa. Następujące dokumenty RFC 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908 opisują standard. Nowością w SNMPv2 jest dodanie możliwość współpracy między zarządcami.

W 1998 roku ustandaryzowano SNMPv3, publikując jednocześnie dokumenty RFC numer 2271, 2273, 2273, 2274, 2275. Głównym usprawnieniem wersji trzeciej standardu jest dodanie aspektów związanych z bezpieczeństwem, w tym szyfrowanie i autoryzację.

2. Różnice w protokole SNMP wersji pierwszej i drugiej

Zmiany, które wprowadzono w nowszej wersji protokołu można pogrupować na modyfikację dokonane w następujących kategoriach:

- struktura informacji zarządzania (SMI),
- współpraca między zarządcami,
- działanie protokołu.

SMI zostało rozszerzone o nowe możliwości oraz niewiele zmienione w stosunku do starszej wersji. SMI protokołu SNMPv2 można podzielić na następujące części:

- definicje obiektów,
- tabele,
- definicje powiadomień,
- moduły informacyjne.

Definicje obiektów, czyli ogólnie ujmując tworzenie baz MIB. Różnice i zmiany zostaną zaprezentowane w poniższej tabeli.

SNMPv2	***SNMPv1***
Typy podstawowe:	
INTEGER (-2147483648..2147483647)	INTEGER
OCTET STRING (SIZE (0..65535))	OCTET STRING
OBJECT IDENTIFIER	OBJECT IDENTIFIER
	NULL

SNMPv2	*SNMPv1*
Typy aplikacyjne:	
IpAddress	IpAddress
Counter32	Counter
TimeTicks	Timeticks
Opaque	Gauge
Counter64	Opaque
Unsigned32	
RowStatus	
MAX-ACCESS	**ACCESS**
read-only	read-only
read-write	read-write
read-create	write-only
not-accessible	not-accessible
accessible-for-notify	
STATUS	
current	mandatory | optional
obsolete	obsolete
deprecated	deprecated
INDEX | AUGMENTS	**INDEX**
DESCRIPTON	**DESCRIPTON**

Tabela 1.Różnice w tworzeniu baz MIB pomiędzy SNMPv1 i SNMPv2c

Typy proste zostały ograniczone, w ten sposób, że należy podawać zakres wartości, które mogą przyjmować. Rozszerzono typy aplikacyjne o 64 bitowe wartości.

Klauzula ACCESS została zastąpiona klauzulą MAX-ACCESS, której wartości zostały rozszerzone, i tak ”read-create” oznacza obiekt do odczytu, zapisu i tworzenia, natomiast ”accessible-for-notify” określa obiekt dostępny tylko przez agenta podczas tworzenia komunikatu typu pułapki (eng. trap).

Opcje klauzuli STATUS zostały uszczuplone o dwie opcje, które zastąpiono jedną. Dlatego nie ma zapisu o obowiązkowości lub opcjonalności obiektu, tylko informacje czy istnieje w bieżącym standardzie.

Kolejne rozszerzenie dotyczy tablic, które należało dopracować. Dopuszczalne są dwa typy tablic:

- zabraniające tworzenia i usuwanie wierszy przez zarządcę – wszystkie obiekty tablicy mogą być co najwyżej o dostępie typu ”read-write”,
- pozwalające na tworzenie i usuwanie wierszy przez zarządcę – taka tablica może być pusta i dopiero zarządca będzie ją wypełniał.

Indeksowanie tablic zostało zmodyfikowane, pierwszą zmianą jest to, że obiekty, które są indeksem nie mogą być dostępne (atrybut ”not-accessible”). Dodatkowo zamiast klauzuli INDEX może wystąpić AUGMENT. Klauzula ta jest wykorzystywana w przypadku rozszerzenia istniejącej tabeli o nowe obiekty. Wartością tej klauzuli jest nazwa wiersza bazowego tablicy, która ma zostać rozszerzona. Innymi słowy jeśli wytwórca urządzenia chce dodać jakieś możliwości monitorowania do istniejącej tablicy (która jest w standardowej bazie MIB) może ją rozszerzyć o własne pola.

Usuwanie wierszy i tworzenie nowych wierszy tablic zostało ustandaryzowane, do tego celu

wykorzystywane są podstawowe operacje (głównie ”set”). Poniżej omówiono szczegółowiej powyższe operacje.

Tworzenie nowego oraz usuwanie istniejącego wiersza tablicy wymaga odpowiedniego przygotowania tablic do tych operacji. Po pierwsze tabela musi zawierać obiekt typu ”RowStatus” o dostępie ”read-create”, jest to tak zwana kolumna statusowa. Powinna także zawierać obiekty o dostępie ”read-create”, aczkolwiek nie jest to konieczne.

Wyróżniamy dwie metody tworzenia nowego wiersza:

- createAndWait,
- createAndGo

Pierwsza metoda rozpoczyna się od wysłania żądania utworzenia nowego wiersza z określoną, przez zarządcę, wartością indeksową, która nie powinna występować w danej tabeli. Jeśli agent utworzy wiersz, przypisuję wszystkim zmiennym, o dostępie ”read-create”, wartości domyśle (jeśli są one określone). W przypadku ustawienia wszystkim takim obiektom wartości początkowych stan wiersza określony będzie na ”notInService”, jednak jeśli jakiś obiekt nie ma przypisanej wartości domyślnej wówczas stan wiersza ustawiony zostanie na ”notReady”. Kolejnym krokiem zarządcy jest wysyłanie żądania Get tego wiersza, w celu otrzymania wartości domyślnych lub „noSuchInstance”, jeśli wartość nie jest określona. Dodatkowo, jeśli agent nie implementuje jakiegoś obiektu, który jest zdefiniowany w bazie MIB, odpowiada wartością „noSuchObject”. Następnie zarządca ustawia wartości wszystkich nie uaktywnionych obiektów oraz może zmieniać wartości domyśle. Ostatnim krokiem jest zmiana wartość kolumny statusowej na ”active”, w celu uaktywnienia wiersza tabeli.

Druga metoda jest prostsza, ale posiada pewne ograniczenia:

- obiekty tabeli muszą się zmieścić w pojedynczej jednostce PDU,
- zarządca nie poznaje wartości domyślnych obiektów w sposób automatyczny.

Zarządca wybiera numer instancji (wartość indeksu), a następnie może sprawdzić żądaniem typu Get, które obiekty nie mają przypisanych wartości domyślnych (lub jeśli posiada tą wiedzę pomija ten krok). Ostatecznie wysyła rozkaz Set, który tworzy wiersz i przypisuje wartości obiektom, które mają zdefiniowany dostęp typu ”read-create” oraz nie mają przypisanej wartości domyślnej. Oczywiście można zmienić w tej operacji te wartości poprzez załączenie wartości tych obiektów. Jeśli operacja ta powiedzie się wiersz zostanie utworzony i automatycznie zostanie uaktywniony.

Obie metody powinny być implementowane w agentach, każda ma swoje wady i zalety, wybór używanej metody zależy od ograniczeń oraz od możliwego obciążenia agenta.

Jeszcze jedną ważną nowością jest nowa operacja protokołu, którą jest żądanie zarządcy wysyłane do innego zarządcy. Informuje ona o stanie informacji zarządzania na innym zarządcy. Ten typ komunikatu nazywamy ”InformRequest”.

Występuje jeszcze jeden nowy typ komunikatu ”GetBulkRequest” służy on do odczytu wielu wartości. Działa on na podobnej zasadzie, co ”GetNextRequest”, jednak może zwrócić wiele następników obiektu lub wielu obiektów. Komunikat tez powstał, aby zmniejszyć liczbę przesyłanych komunikatów pomiędzy zarządcą a agentem.

InformRequest wysyła wartość określonego obiektu do zdalnego zarządcy, który tą wiadomość przekazuje do docelowej aplikacji i odsyła odpowiedź Response PDU zawierającą informacje co stało się z zawartością żądania.

Zmiany z bazie mib-2 są częściowo zaprezentowane w dodatku.

3. Nowości w SNMPv3

Ostatnia wersja protokołu SNMPv3 wprowadza brakujące mechanizmy bezpieczeństwa. W protokole SNMPv2c brakowało ich, gdyż uwierzytelnianie na podstawie nazwy społeczności, przy braku szyfrowania, umożliwiało w bardzo prosty sposób przejmowanie takiej transmisji. Nowa wersja protokołu nie zmienia formatów jednostek PDU, dlatego zachowuje zgodność formatu jednostek PDU z wcześniejszymi wersjami protokołów.

Wykorzystywane są dwa modele bezpieczeństwa:

- oparty na użytkownikach (USM – User-based Security Model) – zapewnia uwierzytelnianie i szyfrowanie wiadomości;
- oparty na widokach (VACM – View-base Access Control Model) – ustala dostęp do baz MIB dla danego zarządcy, działa na poziomie jednostek PDU.

Poniżej zostaną dokładniej omówione oba modele bezpieczeństwa.

- Model USM

Specyfikacja modelu obejmuje:

- uwierzytelnianie – kod uwierzytelniający wykorzystuje funkcję HMAC, funkcja kodująca MD5 lub SHA-1,
- poprawność czasowa – chroni przed opóźnieniem i powielaniem komunikatów,
- prywatność – algorytm DES do szyfrowania zasadniczej części komunikatu,
- format komunikatu – definicja pola ”msgSecurityParameters”, która obejmuje trzy powyższe informacje,
- wykrywanie – umożliwia otrzymanie informacji na temat komunikatu poprzez inny komunikat SNMP,
- zarządzanie kluczami – zawiera procedury tworzenia, modyfikacji i używania kluczy.

Używane funkcje kryptograficzne potrzebują dwóch kluczy, jednego do szyfrowania (privKey), a drugiego do uwierzytelniania (authKey). Wyróżniamy także dwa rodzaje użytkowników:

- lokalny – użytkownik, dla którego operacje SNMP w danym agencie są uwierzytelnione;
- zdalny – użytkownik zdalnego agenta, do którego możemy wysłać żądanie.

Uwierzytelnianie wykorzystuje jedną z dwóch metod, które tworzą 96-bitowy kod uwierzytelniania:

- HMAC-MD5-96 – wykorzystuje 128-bitowy klucz (authKey),
- HMAC-SHA-96 – wykorzystuje 160-bitowy klucz (authKey),

wyniki obu funkcji są obcinane do 96-bitowego ciągu.

Szyfrowanie natomiast wykorzystuje standardową funkcję DES z kluczem (privKey) 128-bitowym.

W celu zmniejszenia liczby haseł i kluczy wprowadzono system, który pozwala na przechowywanie tylko jednej pary (lub dwóch par) hasła i klucza. Mechanizm ten działa w następujący sposób:

- hasło użytkownika jest wykorzystywane do stworzenia klucza, tworzony jest on poprzez powiększenie klucza (poprzez konkatenację) do wielkości 2^{20} oktetów i następnie wykorzystując funkcje jednokierunkową MD5 (dla klucza 16-oktetowego) lub SHA-1 (20-oktetowego) tworzony jest klucz (digest1);
- dla każdego agenta generowany jest jego klucz (digest2), który powstaje poprzez konkatenację klucza (digest1) z wartością obiektu ”snmpEngineID” (identyfikującym

agenta) i jeszcze raz z kluczem (digest1); oczywiście tak utworzony ciąg należy skrócić wykorzystując funkcję jednokierunkową (jedną z dwóch wykorzystywanych w poprzednim etapie).

Problemem pozostaje przesyłanie kluczy między zarządcą a agentami, protokół SNMP nie umożliwia dostępu do wartości klucza (ani hasła), dlatego pierwsze przekazanie hasła należy do administratora (oczywiście musi on wykorzystywać bezpieczne przekazanie klucza). Bezpieczeństwo transmisji i przekazywanych danych zależy od siły klucza i jego ewentualnej okresowej zmiany, dlatego wprowadzono możliwość zmiany klucza wykorzystując do tego protokół SNMPv3. Proces ten może zainicjować zdalny użytkownik (agent) lub stacja zarządzająca. Wykorzystywany w tym celu jest jeden z dwóch mechanizmów:

- zaszyfrowanie nowego klucza starym kluczem;
- wykorzystując funkcje jednokierunkowe (hash) i operacje XOR:
  do_wysłania = random || (hash(keyOld || random) XOR keyNew);
  wartość losowa (random) jest przesyłana jawnie, keyOld jest znany przez obie strony, dlatego strona odbierająca nie ma żadnego problemu z odczytaniem wartości nowego klucza.

- Model VACM

Określa uprawnienia do baz MIB dla grup użytkowników, ogólnie wyróżnia się pięć elementów odpowiedzialnych za tworzenie modelu VACM:

- grupy – określają uprawnienia całej grupy,
- poziomy bezpieczeństwa – określa uprawnienia grupy do obiektów (na przykład tylko-do-odczytu),
- konteksty – określają podgrupy obiektów,
- widoki MIB – ogranicza podzbiór dostępnych obiektów,
- zasady dostępu – określa je administrator posługując się powyższymi możliwościami.

Szczegóły tego modelu powinny zostać wytłumaczone i wyjaśnione przy omawianiu przykładów.

4. Przebieg laboratorium

Uruchomić komputer z systemem operacyjnym Linux.

Zapoznać się z poniższymi przykładami.

I. Przykłady

Konfiguracja usługi SNMPD

dostępne dyrektywy:

- rouser | rwuser <user> [noauth|auth|priv] [OID]

określa użytkownika w modelu USM i ewentualnie prawa do kreślonego obiektu; oczywiście można łączyć użytkownika z uprawnieniami modelu VACM

- rocommunity[6] | rwcommunity[6] <społeczność> [<źródło>] [OID]

określenie nazw społeczności oraz praw do nich przypisanych (dla wersji IPv4 i IPv6)

- com2sec[6] <sec.nazwa> <źródło> <społeczność>

przypisuje nazwę społeczności do nazwy wykorzystywanej w modelu bezpieczeństwa

- group <nazwa> <model> <sec.nazwa>

przypisuje nazwę modelu bezpieczeństwa z modelem bezpieczeństwa do nazwy grupy; model przyjmuje jedną z trzech wartości: v1, v2c, usm

- view <widok> <typ> <poddrzewo> [<maska>]

typ może być: included lub excluded; poddrzewo określa obiekt od którego będzie ważna reguła w zależności od maski (tworzenie jej zostanie opisanie poniżej)

- access <nazwa> <kontekst> <model> <poziom> <prefix> <read> <write> <notify>

wiąże nazwy grup i widoków; model może przyjmować wartości: usm, v1, v2c, any; poziom – noauth, auth, priv; prefix określa jak kontekst powinien się zgadzać z przychodzącym PDU, możliwości: exact, prefix; read, write i notify określają widok, który ma być przyporządkowany dla określonych uprawnień

Maska jest liczbą szesnastkową rozdzielaną kropkami, jednak do jej zrozumienia potrzebny jest zapis bitowy. Długość maski nie może być dłuższa (znacząca część zawierająca jedynki) niż nazwa obiektu (liścia drzewa), licząc tylko liczbę liczb rozdzielanych kropkami. Maska ma takie znaczenie, że jeśli występuje bitowa jedynka, to odpowiadający mu obiekt (węzeł) nie może ulec zmianie. Natomiast jeśli w masce występuje zero to odpowiadający mu obiekt może zmieniać wartość na sąsiednie. Jeśli długość maski jest krótsza niż wskazywanego obiektu wówczas domyślnie przyjmuje się, że pozostałe bity są zerami.

przykład:

```
view cust1 included interfaces.ifTable.ifEntry.ifIndex.1 ff.a0
```

interfaces.ifTable.ifEntry.ifIndex.1 == .1.3.6.1.2.1.2.2.1.1.1

ff.a0 == 1 1 1 1 1 1 1 1.1 0 100000

określa, że zamiast ifIndex może wystąpić dowolny obiekt z jego sąsiadów.

II. Zadania

Zapoznaj się z następującymi aplikacjami systemu Linux:

- snmpbulkget
- snmpbulkwalk
- snmpinform
- snmptable
- snmpnetstat
- snmpstatus
- snmptest
- net-snmp-config –create-snmpv3-user
- snmpusm
- snmpvacm

Stwórz konfigurację (lub zmień) usługi snmpd, która będzie zawierać elementy modelów usm i vacm, stwórz użytkownika dla uwierzytelniania i szyfrowania; musisz umożliwić sprawdzenie konfiguracji zdalnie (z całego laboratorium), dodatkowo powinieneś przygotować komendy którymi można sprawdzić poprawne działanie stworzonej konfiguracji:

- dostęp użytkownika z poprawnym hasłem do dowolnego obiektu,
- odmowa dostępu z powodu błędnego hasła uwierzytelniającego (oraz szyfrującego – druga komenda)
- dostęp do konkretnego widoku oraz pokazanie, że nie można otrzymać informacji nie należących do niego,
- podobnie do powyższego z tą zmianą, że widok musi zakazywać dostępu,

Wszystko przygotowane i działające należy przedstawić prowadzącemu, do tego celu należy wykorzystać wyłącznie komputery znajdujące się w sali laboratoryjnej.

Struktura mib-2 dla wersji SNMPv2c (część zmian)

- system (mib-2 1)
  - sysDescr (1)
  - sysObjectID (2)
  - sysUpTime (3)
  - sysContact (4)
  - sysName (5)
  - sysLocation (6)
  - sysServices (7)
  - sysORLastChange (8)
  - sysORTable (9) – NA
    - sysOREntry (1) – NA
      - sysORIndex (1) – NA <-
      - sysORID (2)
      - sysORDescr (3)
      - sysORUpTime (4)
- interfaces (mib-2 2)
  - ifNumber (1)
  - ifTable(2) - NA
    - ifEntry (1) - NA
      - ifIndex (1) <-
      - ifDescr (2)
      - ifType (3)
      - ifMtu (4)
      - ifSpeed (5)
      - ifPhysAddress (6)
      - ifAdminStatus (7) - RW
      - ifOperStatus (8)
      - ifLastChange (9)
      - ifInOctets (10)
      - ifInUcastPkts (11)
      - ifInNUcastPkts (12)
      - ifInDiscards (13)
      - ifInErrors (14)
      - ifInUnknownProtos (15)
      - ifOutOctets (16)
      - ifOutUcastPkts (17)
      - ifOutNUcastPkts (18)
      - ifOutDiscards (19)
      - ifOutErrors (20)
      - ifOutQLen (21)
      - ifSpecific (22)
- at (mib-2 3) – grupa translacji adresów
  - atTable (1)
    - atEntry (1)
      - atIfIndex (1) – RW <-
      - atPhysAddress (2) – RW
      - atNetAddress (3) – RW <-
- ip (mib-2 4)
  - ipForwardung (1)
  - ipDefaultTTL (2)
  - ipInReceives (3)
  - ipInHdrErrors (4)
  - ipInAddrErrors (5)
  - ipForwDatagrams (6)
  - ipInUnknownProtos (7)
  - ipInDiscards (8)
  - ipInDelivers (9)

- ipOutRequests (10)
- ipOutDiscards (11)
- ipOutNoRoutes (12)
- ipReasmTimeout (13)
- ipReasmReqds (14)
- ipReasmOKs (15)
- ipReasmFails (16)
- ipFragOKs (17)
- ipFragFails (18)
- ipFragCreates (19)
- ipAddrTable (20) - NA
  - ipAddrEntry (1) - NA
    - ipAdEntAddr (1)
    - ipAdEntIfIndex (2) <-
    - ipAdEntNetMask (3)
    - ipAdEntBcastAddr (4)
    - ipAdEntReasmMaxSize (5)
- ipRouteTable (21) - NA
  - ipRouteEntry (1) – NA
    - ipRouteDest (1) – RW <-
    - ipRouteIfIndex (2) – RW
    - ipRouteMetric1 (3) – RW
    - ipRouteMetric2 (4) – RW
    - ipRouteMetric3 (5) – RW
    - ipRouteMetric4 (6) – RW
    - ipRouteNextHop (7) – RW
    - ipRouteType (8) – RW
    - ipRouteProto (9) – RW
    - ipRouteAge (10) – RW
    - ipRouteMask (11) – RW
    - ipRouteMetric5 (12) – RW
    - ipRouteInfo (13)
- ipNetToMediaTable (22) – NA
  - ipNetToMediaEntry (1) – NA
    - ipNetToMediaIfIndex (1) – RW <-
    - ipNetToMediaPhysAddress(2) – RW
    - ipNetToMediaNetAddress (3) – RW <-
    - ipNetToMediaType (4) – RW
- ipRoutingDiscards (23)
- ipForward (24)
  - ipForwardNumber (1)
  - ipForwardTable (2) – NA
    - ipForwardEntry (1) – NA
      - ☑ ipForwardDest (1) <-
      - ☑ ipForwardMask (2)
      - ☑ ipForwardPolicy (3) <-
      - ☑ ipForwardNextHop (4) <-
      - ☑ ipForwardIfIndex (5)
      - ☑ ipForwardType (6)
      - ☑ ipForwardProto (7) <-
      - ☑ ipForwardAge (8)
      - ☑ ipForwardInfo (9)
      - ☑ ipForwardNextHopAS(10)
      - ☑ ipForwardMetric1 (11)
      - ☑ ipForwardMetric2 (12)
      - ☑ ipForwardMetric3 (13)
      - ☑ ipForwardMetric4 (14)
      - ☑ ipForwardMetric5 (15)

- icmp (mib-2 5)
  - icmpInMsgs (1)
  - icmpInErrors (2)
  - icmpInDestUnreachs (3)
  - icmpInTimeExcds (4)
  - icmpInParmProbs (5)

- icmpInSrcQuenchs (6)
- icmpInRedirects (7)
- icmpInEchos (8)
- icmpInEchoRers (9)
- icmpInTimestamps (10)
- icmpInTimestampReps (11)
- icmpInAddrMasks (12)
- icmpInAddrMaskReps (13)
- icmpOutMsgs (14)
- icmpOutErrors (15)
- icmpOutDestUnreachs (16)
- icmpOutTimeExcds (17)
- icmpOutParmProbs (18)
- icmpOutSrcQuenchs (19)
- icmpOutRedirects (20)
- icmpOutEchos (21)
- icmpOutEchoRers (22)
- icmpOutTimestamps (23)
- icmpOutTimestampReps (24)
- icmpOutAddrMasks (25)
- icmpOutAddrMaskReps (26)

- tcp (mib-2 6)
  - tcpRtoAlgorithm (1)
  - tcpTtoMin (2)
  - tcpRtoMax (3)
  - tcpMaxConn (4)
  - tcp ActiveOpens (5)
  - tcpPassiveOpens (6)
  - tcpAttemptFails (7)
  - tcpEstabResets (8)
  - tcpCurrEstab (9)
  - tcpInSegs (10)
  - tcpOutSegs (11)
  - tcpRetransSegs (12)
  - tcpConnTable (13) – NA
    - tcpConnEntry (1) – NA
      - tcpConnState (1) – RW
      - tcpConnLocalAddress (2) <-
      - tcpConnLocalPort (3) <-
      - tcpConnRemAddress (4) <-
      - tcpConnRemPort (5) <-
  - tcpInErrors (14)
  - rcpOutRsts (15)
- udp (mib-2 7)
  - udpInDatagrams (1)
  - udpNoPorts (2)
  - udpInErrors (3)
  - udpOutDatagrams (4)
  - udpTable (5) – NA
    - udpEntry (1) – NA
      - udpLocalAddress (1) <-
      - udpLocalPort (2) <-
- egp (mib-2 8)
  - egpInMsgs (1)
  - egpInErrors (2)
  - egpOutMsgs (3)
  - egpOutErrors (4)
  - egpNeighTable (1) – NA
    - egpNeighRntry (1) – NA
      - egpNeighState (1)
      - egpNeighAddr (2) <-
      - egpNeighAs (3)
      - egpNeighInMsgs (4)
      - egpNeighInErrs (5)

        - egpNeighOutMsgs (6)
        - egpNeighOutErrs (7)
        - egpNeighInErrMsgs (8)
        - egpNeighOutErrMsgs (9)
        - egpNeighStateUps (10)
        - egpNeighStateDowns (11)
        - egpNeighIntervalHello (12)
        - egpNeighIntervalPoll (13)
        - egpNeighMode (14)
        - egpNeighEventTrigger (15)
    - egpAs (6)
- snmp (mib-2 11)
    - snmpOnPkts (1)
    - snmpInBadVersions (3)
    - snmpInBadCommunityNames (4)
    - snmpInBadCommunityUses (5)
    - snmpInASNParseErrs (6)
    - snmpEnableAuthenTraps (30)
    - selenotropizmie (31)
    - snmpProxyDrops (32)