SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1751

UGRADNJA SNMP PODRŠKE U PROTOKOL ZA RAZMJENU KLJUČEVA

Miljen Mikić

Zagreb, rujan 2008.

Sažetak

U ovom radu opisan je jednostavni mrežni upravljački protokol (Simple Network Management Protocol) te pripadajuća arhitektura sustava za upravljanje mrežom. Također, opisan je protokol za razmjenu ključeva (Internet Key Exchange Protocol). U praktičnom dijelu rada definirane su podatkovne strukture i varijable ikev2 implementacije koje se mogu koristiti za potrebe nadziranja u produkcijskim sustavima. Izdvojeni elementi opisani su korištenjem ASN.1 notacije te je razrađen i ostvaren mehanizam ugradnje podrške za SNMP u ikev2 implementaciju.

Abstract

This diploma thesis describes Simple Network Management Protocol (SNMP) and appropriate architecture of the network managing system. It is also described Internet Key Exchange Protocol. Data structures and variables of the ikev2 implementation which can be used for monitoring purposes are defined in the practical part of the diploma thesis. Particular elements are described using Abstract Syntax Notation One (ASN.1). Finally, mechanism for SNMP support in the ikev2 implementation is elaborated and developed.

Sadržaj

1. Uvod

Od brojnih problema koje današnji Internet ima, jedan od najvećih je svakako sigurnost. Rješenja je mnogo, od kojih su najpoznatija Kerberos, SSL, TLS i IPsec koji je posebno zanimljiv. Radi se o arhitekturi [2] i skupu protokola, trenutno u trećoj generaciji, koji djeluju na trećem sloju OSI modela . Dva protokola u sklopu IPsec arhitekture koja nude zaštitu prometa su *Encapsulating Security Payload* (ESP) [9] i *Authentication Header* (AH) [10]. ESP je obavezan za implementaciju i nudi tajnost dok je AH opcionalan a nudi autentičnost i besprijekornost. Treći protokol je *Internet Key Exchange version 2* (IKEv2) [1] koji se koristi za razmjenu ključeva, autentifikaciju i autorizaciju. Tijekom rada na implementaciji tog protokola, ikev2, pojavila se ideja o uvođenju dodatne funkcionalnosti – podrške za jednostavni mrežni upravljački protokol (*Simple Network Management Protocol*, SNMP) [3].

SNMP je protokol koji se danas najviše koristi u upravljanju mrežama i trenutno je u trećoj generaciji. Iako u početku korišten uglavnom za administraciju mrežnih uređaja (npr. usmjernika), ovaj protokol se danas koristi za nadzor i upravljanje brojnim entitetima prisutnima na mreži, pa tako i ikev2 implementacijom. Za ugradnju SNMP podrške korištena je danas najrasprostranjenija implementacija ovog protokola s licencom otvorenog koda, "Net-SNMP“. [6]

Rad je organiziran na sljedeći način. U sljedećem poglavlju opisan je SNMP protokol kao i pripadajući pojmovi i notacija. U trećem i četvrtom poglavlju objašnjeni su IPsec arhitektura i IKEv2 protokol. Peto poglavlje donosi opis ikev2 implementacije s naglaskom na arhitekturi. Praktični dio ovog diplomskog rada nalazi se u šestom poglavlju. Rad završava zaključkom, popisom literature i sažetkom.

2. Jednostavni mrežni upravljački protokol

Potaknuti naglim razvojem računalnih mreža (pogotovo Interneta), dizajneri su počeli zamišljati svijet gdje će biti umreženi svi elektronski uređaji; od računala preko video i audio sustava pa čak do tostera. U tu svrhu razvijan je protokol sposoban za administraciju bilo kojeg mrežnog uređaja. Rezultat je SNMP, protokol predstavljen 1988. godine koji sada uključuje tri različite verzije: SNMPv1, SNMPv2 i SNMPv3. Zbog svoje zrelosti, mnogi proizvođači mrežne opreme u startu uključuju podršku za ovaj protokol.

U prvoj fazi razvoja Interneta nije postojao sofisticirani protokol za upravljanje mrežama. Iako mu to uopće nije namjena, za te potrebe je korišten *Internet Control Message Protocol* (ICMP). Taj protokol je i danas jako korišten (ali u druge svrhe), posebice zbog svojih eho poruka (*Echo-Request* i *Echo-Reply)*. Na njima se temelji i najjednostavniji i vrlo popularni alat PING koji služi za dijagnostiku rada računalnih mreža. Ipak, potreba za upravljanjem složenim mrežama iznjedrila je prvi pravi upravljački protokol - *Simple Gateway Monitoring Protocol* (SGMP) [17] koji je bio namijenjen nadzoru usmjernika. Godinu dana od njegovog predstavljanja stvoren je i protokol SNMP koji je nastao poboljšanjem SGMP protokola. Treba spomenuti da je pored SNMP-a razvojem SGMP-a nastao i protokol *Common Management Information Protocol* (CMIP) koji je složeniji te ujedno i ISO standard. CMIP ima više naredbi, bolju sigurnost ali i puno veći *overhead* od SNMP-a. SNMP je tako ostao daleko najrasprostranjeniji i najkorišteniji te je to glavni razlog zašto je podrška baš za taj upravljački protokol ugrađena u ikev2.

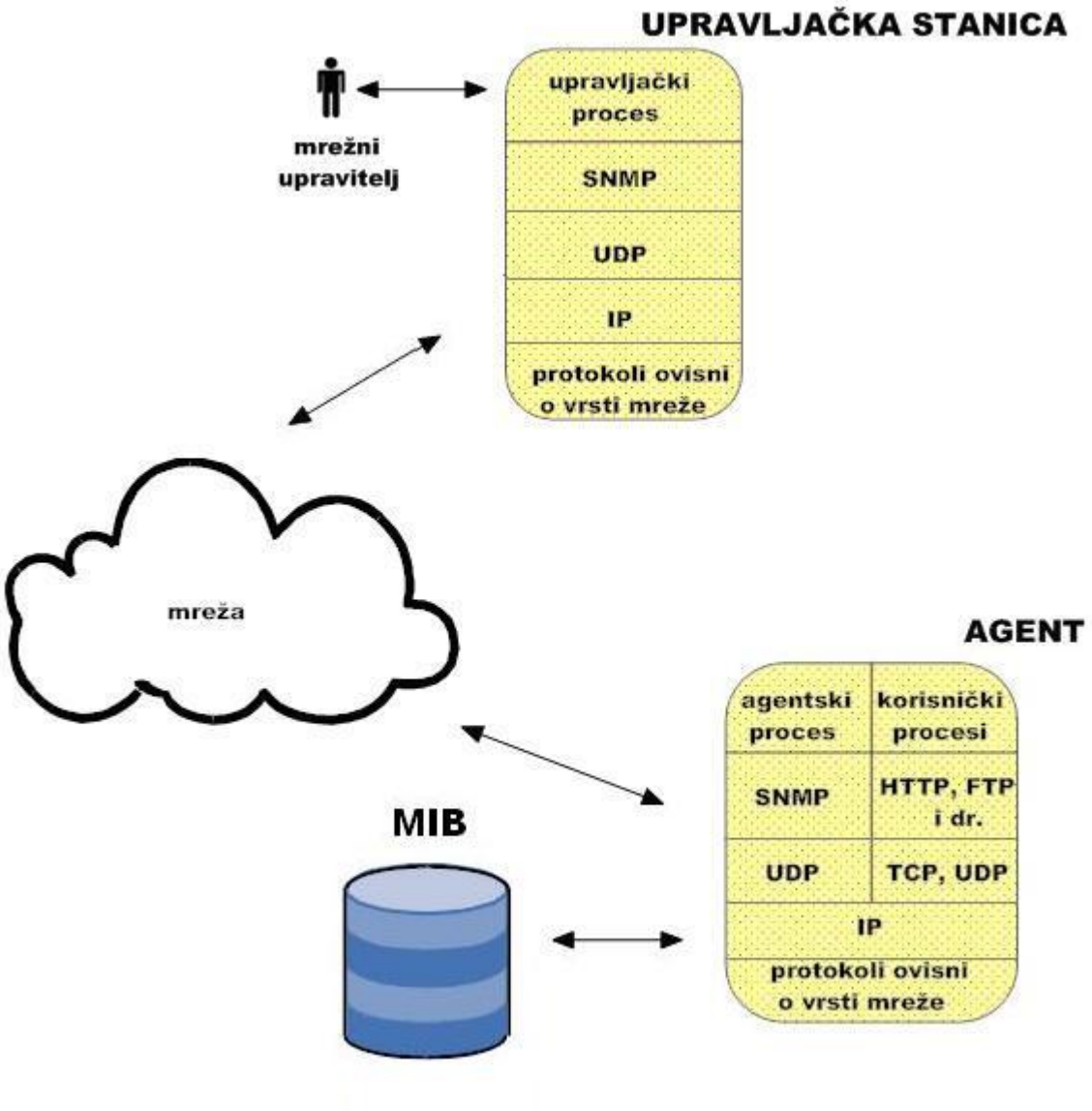
2.1 Osnovne značajke

Sustav za upravljanje mrežom čine:

- mrežni entiteti – entiteti koji su uključeni u mrežu; mogu biti sklopovski (usmjernik, računalo, poslužitelj..) ili programski (protokol..)
- agenti – programska podrška koja se nalazi u mrežnim entitetima, a služi skupljanju i spremanju upravljanih informacija kao što je npr. broj neispravnih paketa
- upravljani objekti – dijelovi mrežnog entiteta koji se mogu nadzirati i/ili upravljati
- baza upravljanih informacija (*Management Information Base,* MIB) [5] – kolekcija upravljanih objekata
- upravljač (*controller*) – entitet koji upravlja i nadzire upravljane objekte agenta

SNMP je protokol koji omogućuje upravljačkoj stanici (**upravljaču**) da upravlja upravljanim sustavom (**agentom**) kroz razmjenu SNMP poruka. SNMP poruka je paket poslan korištenjem bespojnog prijenosnog protokola (*User Datagram Protocol*, UDP) na pristup (*port)* 161. U ISO/OSI modelu SNMP spada u sedmi, odnosno

aplikacijski sloj. Na prvi pogled je možda neobičan izbor UDP protokola s obzirom da na prijenosnom sloju postoji i spojni prijenosni protokol (*Transmission Control Protocol*, TCP) koji pruža pouzdani prijenos informacija uz potvrdu primitka paketa. SNMP na transportnoj razini namjerno koristi UDP budući da se u ovom slučaju prednost daje brzini, odnosno što manjem zauzeću mreže. UDP višim slojevima nudi bespojnu uslugu bez potvrde primitka paketa te se time značajno ubrzava prijenos i manje opterećuje mreža što je bio jedan od glavnih zahtjeva prilikom dizajna protokola za upravljanje mrežom. Slika 2.1 prikazuje konfiguraciju sustava za upravljanje mrežom pomoću SNMP protokola.



Slika 2.1 Konfiguracija sustava za upravljanje mrežom

Upravljani podaci prikazani su u obliku varijabli (npr. `ime sustava`, `broj pokrenutih procesa`) na agentima. Tim varijablama se onda mogu slati upiti za dohvat vrijednosti, a ako je dozvoljeno mogu se i postavljati od strane upravljača.

SNMP služi prvenstveno definiranju načina komunikacije između upravljača i agenta. Koristi se dohvati-i-spremi paradigma (*fetch and store)*. SNMP poruka dakle služi postavljanju (*store)* ili nadzoru (*fetch*) varijabli SNMP agenta. Sama varijabla je instanca generičkog objekta – kao što će se vidjeti u nastavku, ikev2 može imati više instanci objekta "sjednica", za svaku sjednicu po jedan. Varijable su organizirane hijerarhijski u obliku stabla. SNMP koristi identifikatore objekta (*Object Identifier*, OID) za određivanje mjesta pojedine varijable u stablu. Svaki OID sastoji se od brojeva međusobno odvojenih točkom i naravno, jedinstven je. Primjerice, identifikator objekta za varijablu `dos treshold` je .1.3.6.1.3.411249.1. Identifikatori objekata nalaze se, zajedno s još nekim informacijama, u MIB-u (prikazano plavo na slici 2.1), odnosno bazi upravljanih informacija koja je opisana u sljedećem potpoglavlju.

2.2 Baza upravljanih informacija

Svaki SNMP agent sadrži popis svih svojih upravljanih objekata i taj se popis zove baza upravljanih informacija. Baza sadrži sljedeće zapise: ime, OID, tip podatka, dozvole čitanja i pisanja te kratki opis za svaki objekt SNMP agenta. S pomoću informacija o objektu i vrijednosti pojedine instance – varijable, SNMP upravljač može slati SNMP poruke za dohvat ili postavljanje pojedine varijable SNMP agenta.

SNMP ne definira unaprijed skup MIB varijabli pa je dizajn vrlo prilagodljiv. Čim se pojavi potreba, mogu se definirati i objaviti nove MIB varijable. U slučaju pojave novog protokola (primjerice, protokola za razmjenu ključeva), osobe zadužene za dizajn protokola izdvoje i definiraju varijable bitne za upravljanje programskom podrškom protokola (što je bitan dio i ovog diplomskog rada). Jednako tako, u slučaju pojave novog sklopovlja, ljudi koji su ga razvijali definiraju MIB varijable za nadzor i upravljanje tim uređajem. Svaki bitni protokol danas definira i odgovarajući skup MIB varijabli:

- RFC 3418 opisuje bazu upravljanih informacija za protokol SNMP
- RFC 4022 opisuje bazu upravljanih informacija za protokol TCP
- RFC 4113 opisuje bazu upravljanih informacija za protokol UDP
- RFC 4293 opisuje bazu upravljanih informacija za protokol IP

Kao što se vidi, čak i sam SNMP protokol definira svoj skup MIB varijabli koje su standardizirane odgovarajućim RFC-om. To su npr. varijabla `sysDescr` koja pruža puno ime i verziju sistemskog sklopovlja, operacijskog sustava i mrežne programske podrške, varijabla `sysContact` koja predstavlja podatke o osobi za kontakt upravljanog čvora (agenta) te kako ju kontaktirati i primjerice varijabla `sysLocation` koja pruža podatke o fizičkoj lokaciji upravljanog čvora.

2.3 ASN.1

Postoje dvije poteškoće koje SNMP poruke moraju riješiti ako se želi postići interoperabilnost, tj. da svi SNMP uređaji razumiju i znaju interpretirati SNMP poruke.

Prvi problem nastaje zbog toga što različiti programski jezici imaju različite tipove podataka (cjelobrojne, znakove, nizove znakova, oktete itd.). Ukoliko SNMP upravljač pošalje poruku punu tipova podataka iz jednog programskog jezika (npr. Jave) a SNMP agent je napisan u drugom programskom jeziku (npr. C-u), oni se neće razumjeti. Da bi se riješio ovaj problem SNMP koristi ASN.1 (*Abstract Syntax Notation One*) [7] za definiranje tipova podataka korištenih za konstrukciju SNMP poruke. Budući da je ASN.1 sintaksa nezavisna od izbora programskog jezika, SNMP agenti i upravljači mogu biti pisani u bilo kojem programskom jeziku. Međutim, čak i uz korištenje ASN.1 sintakse i dalje ostaje neriješen još jedan problem. Ako se šalju određeni podaci preko prijenosnog medija, kako ih kodirati? Dobar primjer su nizovi znakova – u programskom jeziku C oni moraju imati znak NULL na svom kraju, u nekim drugim jezicima ne. U nekim jezicima logičke vrijednosti kodirane su sa osam bita, a u nekim sa šesnaest itd. ASN.1 sintaksa uključuje osnovna pravila kodiranja (*Basic Encoding Rules*, BER) [10] za rješavanje ovog problema. Neovisno o izboru programskog jezika, svi tipovi podataka su kodirani na isti način prije slanja prijenosnim medijem. Dakle, svi tipovi podataka u SNMP poruci moraju biti ispravni ASN.1 tipovi podataka i moraju biti kodirani u skladu sa osnovnim pravilima kodiranja.

ASN.1 sintaksa je vrlo moćna i kompleksna ali zato pati od nedostatka učinkovitosti. Glavna snaga ASN.1 sintakse je u definiranju jednoznačnih pravila kodiranja na samoj razini bita. No, to je ujedno i slabost ASN.1 sintakse. Pravila kodiranja su takva da je cilj postići što manje bita na prijenosnom mediju, a to se plaća vrlo slabom učinkovitošću korištenja procesora na komunikacijskim krajevima i to prilikom kodiranja i dekodiranja.

ASN.1 je definirana standardom ISO 8824, a pravila kodiranja standardom ISO 8825.

2.3.1 Osnovni pojmovi

Za ASN.1 sintaksu važni su sljedeći pojmovi:

- apstraktna sintaksa
- tip podataka
- kodiranje
- pravila kodiranja
- sintaksa prijenosa

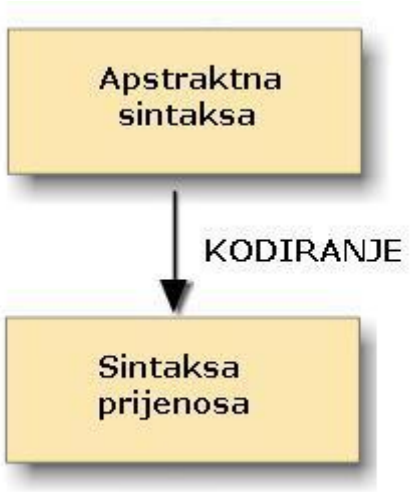
Apstraktna sintaksa definira strukturu podataka neovisnu o načinu kodiranja korištenom za prikaz podataka. Kroz apstraktnu sintaksu je moguće definirati tipove podataka i vrijednosti istih.

Tip podataka uključuje veći broj vrijednosti, a osnovna podjela je na jednostavni i složeni tip. Tipovi podataka detaljnije su opisani u nastavku.

Kodiranjem se dobiva niz okteta koji se koristi za prikaz podatkovnih vrijednosti.

Pravila kodiranja definiraju način preslikavanja iz jedne u drugu sintaksu, tj. iz apstraktne sintakse u sintaksu prijenosa. Drugim riječima, pravilima kodiranja određen je način na koji će se skup podatkovnih vrijednosti iz apstraktne sintakse prikazati u sintaksi prijenosa.

Sintaksa prijenosa definira slijed bita koji će biti prenijeti krajnjem odredištu. Radi se dakle o fizičkom prikazu podataka, za razliku od prikaza podataka definiranog apstraktnom sintaksom. Slika 2.2 prikazuje preslikavanje iz apstraktne sintakse u sintaksu prijenosa.



Slika 2.2 Preslikavanje iz apstraktne sintakse u sintaksu prijenosa

2.3.2 Tipovi podataka

Konstrukcija ispravne poruke zahtijeva znanje o tipovima podataka specificiranima ASN.1 sintaksom. Kao što je prethodno rečeno, tipove možemo podijeliti u dvije osnovne kategorije – jednostavni i složeni. Jednostavni tipovi su *Integer, Octet String, Bit String, Null, Boolean* i *Object Identifier*. *Object Identifier* je najvažniji za SNMP poruke, budući da to polje čuva identifikator objekta nužan za određivanje varijable u SNMP agentu. Tablica 2.1 prikazuje osnovne tipove podataka i njihov kratki opis. U SNMP okolini koriste se svi ranije pobrojani tipovi, osim tipa *Boolean*.

Tablica 2.1 Osnovni ASN.1 tipovi podataka

Naziv tipa	Kod	Kratki opis
INTEGER	2	Cijeli broj proizvoljne duljine
BIT STRING	3	Niz koji sadrži 0 ili više bita
OCTET STRING	4	Niz koji sadrži 0 ili više okteta bez predznaka (*unsigned*)
NULL	5	*Place holder*
OBJECT IDENTIFIER	6	Tip za označavanje objekata

ASN.1 pruža nekoliko složenih tipova podataka potrebnih za izgradnju SNMP poruka. Jedan od tih složenih tipova je i niz (*Sequence*). Niz nije ništa drugo nego lista podatkovnih polja. Svako polje u nizu može sadržavati drugi tip podatka. ASN.1 također definira i SNMP PDU tipove podataka. Ovdje se radi o složenim tipovima specifičnim za SNMP. SNMP PDU polje sadrži tijelo SNMP poruke. Dva dostupna SNMP PDU tipa podatka su i *GetRequest* i *SetRequest* koji sadrže sve potrebne podatke za dohvat i postavljanje varijabli, respektivno. O njima će detaljnije biti riječi kasnije.

Prilikom korištenja ASN.1 tipova podataka, potrebno je znati i pridržavati se određenih leksičkih pravila. Važnija su:

- ugrađene tipove podataka treba pisati velikim slovima (npr. INTEGER)
- korisnički definirana imena tipova počinju s velikim slovom i moraju sadržavati barem jedan znak koji nije veliko slovo abecede
- identifikatori objekta moraju imati malo početno slovo, a mogu sadržavati velika i mala slova, znamenke i posebne znakove kao što su npr. crtica ili podvlaka
- komentari započinju sa kombinacijom -- i protežu se do kraja retka ili sljedećeg pojavljivanja te kombinacije

Primjer deklaracije i inicijalizacije varijable `brojac`:

```
brojac INTEGER ::= 1000
```

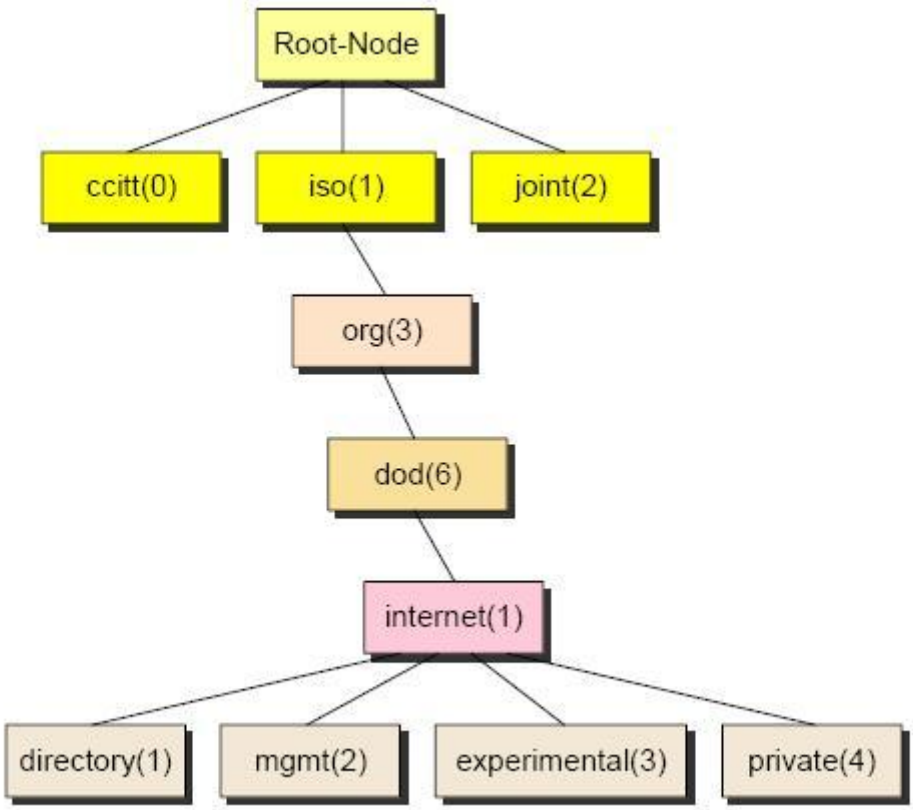
Iako varijabla tipa *Integer* može poprimiti bilo koju vrijednost, SNMP pravila ograničavaju skup mogućih vrijednosti. Ponekad je korisno definirati i podtipove čije su vrijednosti ograničene na specifične vrijednosti ili određeni interval, primjerice:

```
VelicinaPaketa ::= INTEGER (0..1023)
```

Varijable koje su tipa *Bit String* ili *Octet String* sadrže 0 ili više bita, odnosno okteta. Bit poprima vrijednost 0 ili 1, a oktet u rasponu od 0 do 255. Za oba tipa moguće je definirati i duljinu niza znakova kao i početnu vrijednost. Tip *Object Identifier* služi označavanju objekata i pomoću njega gradimo već spomenutu hijerarhijsku strukturu. Ta je struktura u obliku stabla i korištenjem ovog tipa svaki je objekt postavljen na jedinstveno mjesto.

2.3.3 ASN.1 stablo objekata

Kako je već spomenuto, upravljani objekti strukturirani su u obliku stabla. Stablo s imenima objekata vidimo na slici 2.3:



Slika 2.3 ASN.1 stablo objekata

Koristeći se pravilima objašnjenim u prethodnom potpoglavlju lako se mogu definirati pojedini čvorovi stabla (konkretno, podstablo "internet"):

```
internet OBJECT IDENTIFIER ::= {iso org(3) dod(6) 1}

directory OBJECT IDENTIFIER ::= {internet 1}

mgmt OBJECT IDENTIFIER ::= {internet 2}

experimental OBJECT IDENTIFIER ::= {internet 3}
```

```
private OBJECT IDENTIFIER ::= {internet 4}
```

Čvor koji se nalazi na samom vrhu stabla zove se korijenski čvor, odnosno korijen. Ispod njega nalazi se čvorovi roditelji (ukoliko imaju djece), odnosno djeca (listovi) ukoliko ispod njih nema drugih čvorova. U ASN.1 stablu korijenski čvor zove se *Root-Node*, a njegova podstabla su *ccitt(0), iso(1)* i *joint(2)*. Podstablo *iso(1)* povezano je sa SNMP okolinom dok su za ostala dva zadužene odgovarajuće organizacije. *Ccitt* administrira organizacija ITU-T, a za *joint* se brinu zajednički ISO i ITU-T. Podstablo *directory(1)* rezervirano je za buduće uporabe od strane ISO organizacije. *Mgmt(2)* podstablo se koristi za identifikaciju objekata koji su definirani u dokumentima koje je odobrio IAB (*Internet Architecture Board*). Primjerice, SNMP MIB-2 ima OID .1.3.6.1.2.1, odnosno *iso.org.dod.internet.mgmt.1.* Podstablo *experimental(3)* služi za identifikaciju objekata koji se koriste za eksperimente, tj. to je mjesto za stavljanje nestandardiziranih ekstenzija. U to podstablo trenutno spadaju i objekti definirani za IKEv2 protokol. U podstablu *private(4)* nalaze se objekti definirani od strane pojedinaca i organizacija. Potrebno je napomenuti da u novoj verziji strukture upravljačke informacije (opisane kasnije) čvor *internet* ima još dvoje djece – *security(5)* te *snmpv2(6)*.

Kao što je spomenuto na primjeru SNMP MIB-2, svaki OID osim brojčanog, ima i znakovni zapis (.1.3.6.1.2.1 je brojčani zapis za *iso.org.dod.internet.mgmt.1*). To je pristup sličan označavanju Web adresa – pamćenje četiri okteta je bilo nepraktično pa je uveden DNS (*Domain Name Service*). U ovom slučaju ne postoji DNS, već baza upravljanih informacija (MIB) koja sadrži informacije o svakom objektu u stablu.

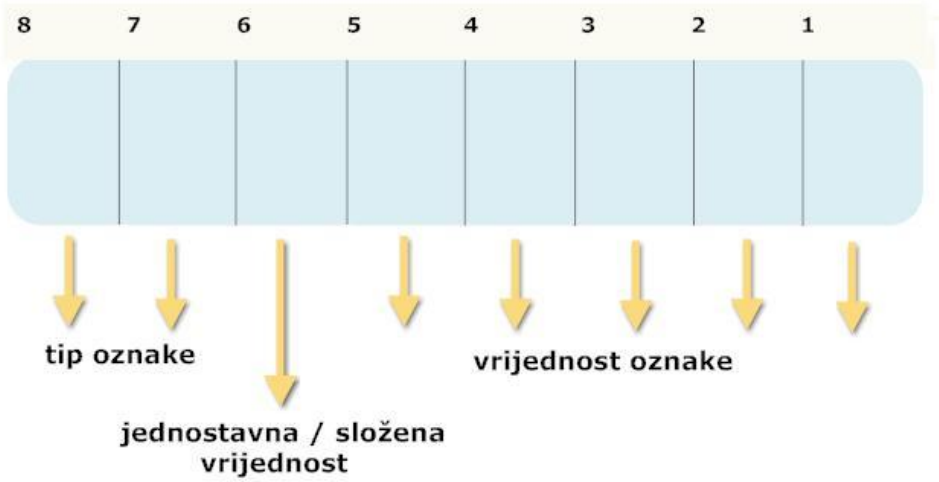
2.3.4 Osnovna pravila kodiranja

SNMP poruka nastaje kompozicijom ASN.1 tipova podataka. Međutim, navođenje ispravnog tipa podatka još uvijek nije dovoljno. U potpoglavlju o ASN.1 tipovima podataka spomenut je i složeni tip koji se zove niz, a sastoji se od više polja koja mogu sadržavati podatke različitog tipa. Sada se nameće pitanje – ako je SNMP poruka niz polja različitih tipova podataka, kako primatelj može znati gdje jedno polje završava a drugo počinje ili koji je tip podatka u svakom polju? Ti problemi izbjegavaju se korištenjem osnovnih pravila kodiranja.

BER predstavlja sintaksu prijenosa koju koristi ASN.1. Standardom ASN.1 obuhvaćena su i druga kodiranja, ali SNMP koristi samo BER. Neka od važnijih koje ASN.1 dopušta su i CER (*Canonical Encoding Rules*) te DER (*Distinguished Encoding Rules*). Glavna razlika između njih i BER kodiranja je u tome što je BER fleksibilniji, odnosno dopušta i alternativna kodiranja kao opciju pošiljatelju. Drugim riječima, primatelj koji želi primati podatke kodirane osnovnim pravilima kodiranja, mora podržati i sve ostale alternative. Primjerice, prilikom kodiranja složenog podatka (podatka koji se sastoji od manjih, već kodiranih podataka), pošiljatelj može koristiti bilo koje od tri različita kodiranja za specificiranje duljine tog podatka. S druge strane, CER i DER su tu puno restriktivniji odnosno ne podržavaju alternativna kodiranja, tj. duljina podatka se mora specificirati (isključivo) u skladu s odgovarajućim kodiranjem. BER se smatra neefikasnim u usporedbi s ostalim kodiranjima budući da proizvodi dulje kodirane podatke nego što je potrebno. Smatra se da je to više zbog loših implementacija nego grešaka u pravilima kodiranja. Bila ta neefikasnost stvarnost ili samo percepcija, dovela je do razvoja novih pravila kodiranja. Jedna od poznatijih su

primjerice PER (*Packet Encoding Rules*) koja nastoje unaprijediti svojstva i duljinu kodiranih podataka koji nastaju primjenom BER kodiranja.

Način na koji se svaki od podataka kodira pomoću BER kodiranja je sljedeći: podatak se kodira pomoću identifikatora tipa, duljine podatka i same vrijednosti podatka te oznake kraja ukoliko je potrebna. Takav način kodiranja uobičajeno se zove TLV (*type-length-value*), odnosno tip-duljina-vrijednost. Identifikator tipa prikazan je slici 2.4.



Slika 2.4 Identifikator tipa podatka kodiranog osnovnim pravilima kodiranja

Bitovi najmanje težine (posljednjih pet bitova) označavaju vrijednost oznake, šesti bit govori da li je vrijednost koja se prenosi jednostavna ili složena (podsjetnik – ASN.1 ima jednostavne i složene tipove podataka) dok je značenje prvih dvaju bitova (bitova najveće težine) koji se zovu tipovi oznake prikazano u tablici 2.2.

Tablica 2.2 Tipovi oznake u identifikatoru tipa

Tip oznake	Bit 8	Bit 7
Universal	0	0
Application	0	1
Context-specific	1	0
Private	1	1

U SNMP okolini jedina dva tipa oznake koja su bitna su *Universal* (vrijednost je ASN.1 ugrađeni tip, npr. INTEGER) te *Context Specific* (obično za složene tipove).

Tablica 2.1 sadrži kodove za jednostavne ASN.1 tipove podataka. Ti kodovi su zapravo vrijednost oznake, odnosno najmanje značajnih pet bitova u oktetu

identifikatora tipa. Slijedi popis kompletnih identifikatora tipa za neke složene ASN.1 tipove podataka (heksadekadski):

- Niz - identifikator tipa je 0x10
- *GetRequestPDU* - identifikator tipa je 0xA0
- *GetResponsePDU* - identifikator tipa je 0xA2
- *SetRequestPDU* - identifikator tipa je 0xA3

Ovi složeni tipovi izgrađeni su pomoću jednostavnih tipova podataka. Prema tome, složeni podaci se kodiraju kao ugniježđena polja. Vizualni prikaz donose sljedeće dvije slike, prva koja prikazuje kodiranje jednostavnog tipa podataka i druga koja isto prikazuje za složeni tip.

Slika 2.5 Format BER kodiranog polja (jednostavni tip podataka)

Slika 2.6 Format BER kodiranog polja (složeni tip podataka)

Postoje još dva BER pravila bitna za kodiranje SNMP poruka. Oba se odnose na kodiranje identifikatora objekata (OID-a). Prvo pravilo odnosi se na kodiranje prva dva broja u OID-u. Prema osnovnim pravilima kodiranja, prva dva broja bilo kojeg OID-a (x.y) kodiraju se kao jedna vrijednost koristeći formulu 40 * x + y. Prva dva broja u svakom SNMP OID-u su .1.3. Dakle, ta dva prva broja kodiraju se kao 43 odnosno heksadekadski 0x2B, budući da je 40 * 1 + 3 = 43. Nakon što su prva dva broja kodirana, sva preostali brojevi u OID-u se kodiraju kao jedan oktet. Jedan oktet može pohraniti vrijednosti iz intervala [0, 255] a primjerice OID varijable `sm threads` iz ikev2 implementacije je .1.3.6.1.3.411249.2, tj. sadrži u sebi broj 411249 koji nikako ne stane u taj jedan oktet. Taj problem rješava drugo pravilo. Ono kaže da se najnižih sedam bitova u oktetu koriste za pohranu vrijednosti broja, a najviši bit predstavlja zastavicu odnosno oznaku primatelju da li se taj broj proteže kroz više okteta. Iz toga slijedi da se svaki broj veći od 127 mora kodirati s dva ili više okteta.

2.4 SNMP poruke

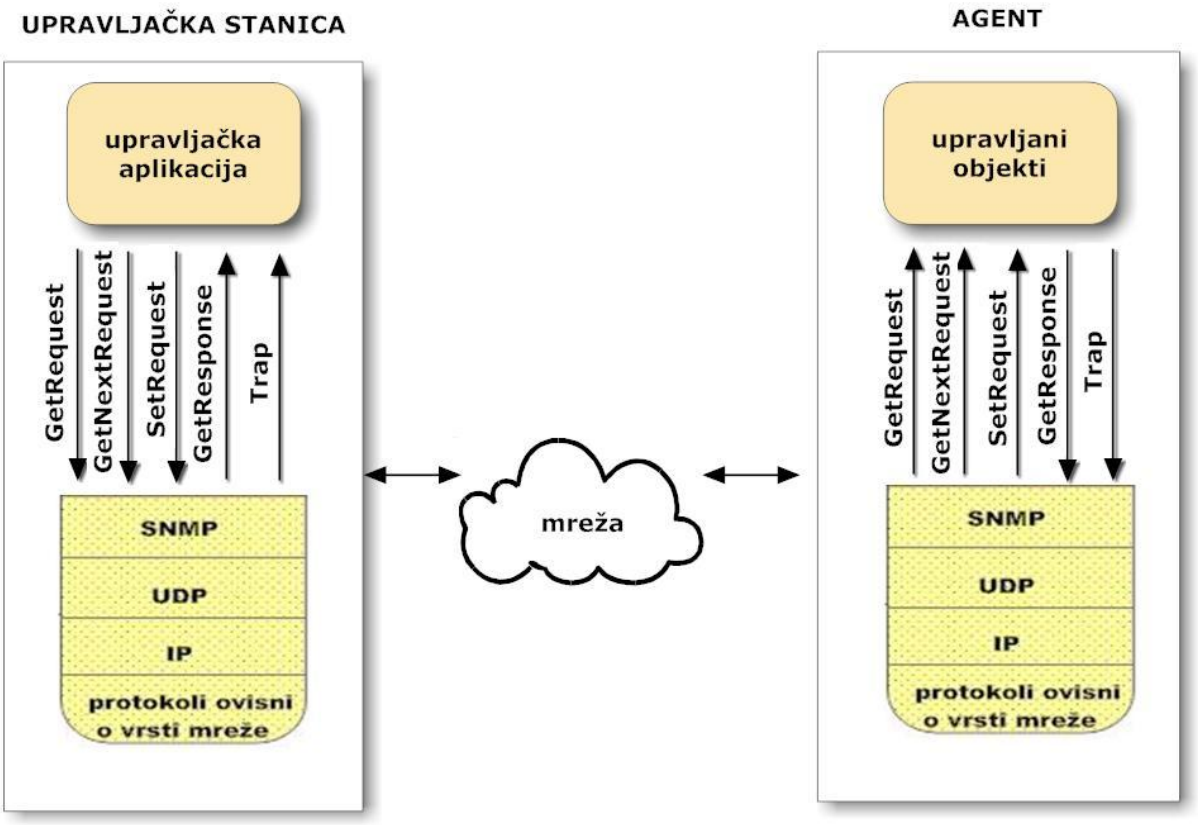
Nakon upoznavanja s ASN.1 sintaksom i osnovnim pravilima kodiranja, vrijeme je za pregled mogućih SNMP poruka te njihovih formata.

Protokol SNMP nudi tri operacije sustavu upravljanja mrežom. To su redom: dohvat (*get),* postavljanje (*set*) i obavijest (*trap*). Prve dvije operacije se odnose na dohvaćanje vrijednosti upravljanih varijabli, odnosno njihovo postavljanje. Postupak slanja poruka agentima u sklopu operacije dohvata nazivamo prozivanje (*polling*) i on se odvija najčešće ciklički u nekom zadanom intervalu. U tom intervalu upravljač proziva agente i nakon što sve prozove kreće ispočetka.

Operacija postavljanja služi upravljaču za postavljanje vrijednosti upravljanje varijable koja se nalazi u MIB-u agenta. I dohvat i postavljanje osim zahtjeva upravljača uključuju i odgovor agenta, odnosno radi se o dvosmjernoj komunikaciji.

Obavijest se sastoji od poruke koju šalje isključivo agent. Pomoću nje se obavještava upravljač o nekom događaju koji se u tom trenutku dešava na strani agenta (to može biti npr. degradacija mrežnih performansi, kvar nekog dijela opreme i slično). Upravljači se mogu programirati da u slučaju primitka obavijesti učine neke automatske akcije (primjerice, ukoliko Ethernet preklopnik pošalje obavijest o čudnom ponašanju nekog pristupa, jedna od mogućih automatskih akcija je skidanje tog pristupa s mreže). Ipak, prozivanje putem operacije dohvata je i dalje nužno budući da u slučaju da SNMP agent iz nekog razloga prestane s radom, jasno da više ne može ni slati obavijesti.

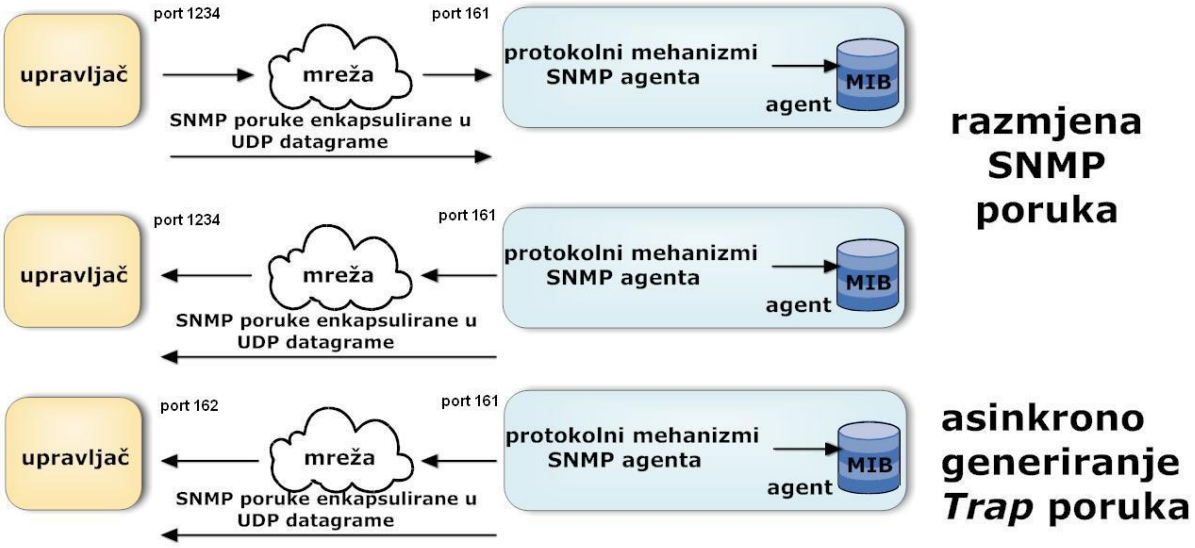
Na temelju ovih mogućnosti koje SNMP pruža, definiraju se SNMP poruke koje prikazuje slika 2.7.



Slika 2.7 SNMP poruke

Upravljač može agentu poslati jednu od poruka *GetRequest, SetRequest* i *GetNextRequest*. Nakon primanja bilo koje od tih poruka, agent upravljaču odgovara sa porukom *GetResponse.* Poruku *Trap* agent šalje upravljaču i time ga obavještava o nekom događaju u svojoj okolini.

SNMP se bazira na UDP protokolu koji je bespojan, pa je time i SNMP protokol takav. Agent osluškuje na pristupu 161, a upravljač može slati s bilo kojeg pristupa. Razmjena poruka prikazana je na slici 2.8.



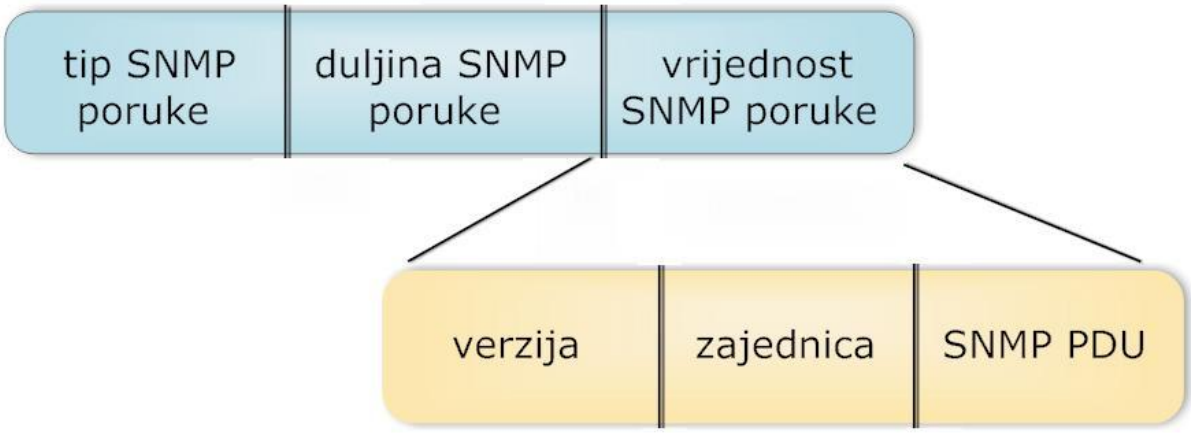
Slika 2.8 UDP enkapsulirane SNMP poruke

SNMP poruke se enkapsuliraju u UDP datagrame. Postupak slanja *GetRequest* ili *SetRequest* poruke počinje upisivanjem vrijednosti izvorišnog pristupa u zaglavlje UDP datagrama od strane upravljača. Na slici je to pristup 1234 i njega je upravljaču dodijelio operacijski sustav. Istovremeno se upisuje i odredišni pristup na kojem agent sluša, a kako je već spomenuto, radi se o UDP pristupu 161. Sličan postupak provodi agent kad šalje odgovor na primljenu poruku. U zaglavlje UDP datagrama upisuje se odredišni pristup kojeg je agent primio u SNMP poruci (u ovom slučaju to je 1234) te izvorišni pristup, odnosno 161. U slučaju generiranja *Trap* poruke stvari su malo drukčije. Tu poruku šalje agent upravljaču pa je prema tome izvorišni pristup ponovo 161, no odredišni pristup je sada 162. To je unaprijed određena vrijednost na kojoj upravljač osluškuje i čeka primitak *Trap* poruka od agenta.

2.4.1 Format poruke

Slika 2.9 prikazuje opći format poruke kodirane osnovnim pravilima kodiranja (objašnjenim u potpoglavlju 2.3.4) , a slika 2.10 prikazuje SNMP poruku kodiranu osnovnim pravilima kodiranja.

Slika 2.9 Opći format BER kodirane poruke



Slika 2.10 BER kodiranje SNMP poruke

SNMP poruku čine tri osnovna polja – verzija, naziv zajednice i PDU.

Verzija je cijeli broj i predstavlja redni broj SNMP protokola. To mogu biti brojevi 1, 2 i 3 koji označavaju redom verzije SNMPv1, SNMPv2 i SNMPv3. Razvoj SNMP protokola i razlika između tih verzija opisani su u posebnom odjeljku, a na ovom mjestu će biti spomenute samo najznačajnije karakteristike bitne za format poruke.

Zajednica se koristi radi sigurnosti i predstavlja šifru. Odmah treba napomenuti da je u današnje vrijeme to slab i gotovo nikakav jamac sigurnosti. U zajednicu koja je tipa *Octet String* upisuje se šifra koja se šalje u čistom (*plain-text*) obliku i ukoliko je paket snimljen, napadač ima otvoren put prema svim ovlastima koje poželi. Ovaj pristup se koristi u verzijama SNMPv1 i SNMPv2 dok SNMPv3 nudi puno bolje mehanizme sigurnosti. Postoje tri zajednice koje se definiraju u konfiguracijskoj datoteci agenta. To su *read-only, read-write* i *trap*. *Read-only* omogućuje upravljačima isključivo čitanje upravljanih varijabli. Nasuprot tome, *read-write* omogućuje i upis, odnosno promjenu vrijednosti upravljane varijable u MIB-u agenta. *Trap* zajednica omogućuje upravljaču primitak *trap* poruke. Zajednice *read-only* i *read-write* su uglavnom unaprijed već postavljene i to kao *public* odnosno *private*. Ovo nosi dodatnu opasnost ukoliko je mrežni administrator neoprezan, tj. nije promijenio postavke

konfiguracijske datoteke. Naime, dovoljno je poslati SNMP poruku u kojoj u polju zajednice piše *private* i pošiljatelj ima pravo promjene bilo koje upravljane varijable. Ukoliko se koriste starije verzije SNMP protokola (dakle, verzije 1 i 2), trebalo bi se barem osigurati sigurnosnom zaštitnom stijenom ili komunikacijom uz korištenje IPsec-a.

2.4.2 SNMP PDU

Za razliku od SNMP verzije i zajednice koji su jednostavni tipova podataka i nisu građeni od manjih polja, SNMP PDU je složeni tip i sastoji se od nekoliko manjih polja (slojeva).

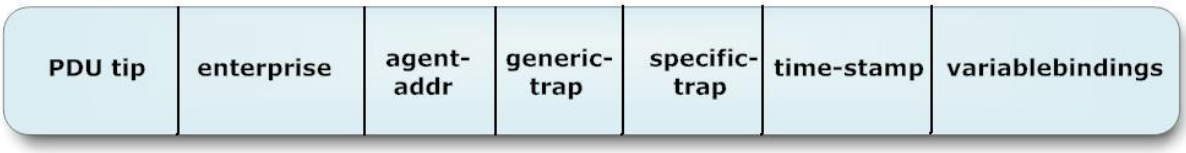
Postoje tri vrste SNMP PDU-a. Prva je prikazana na slici 2.11 i nju koriste poruke *GetRequest, GetNextRequest* i *SetRequest* (dakle, poruke koje generira upravljač).

Slika 2.11 SNMP PDU za GetRequest, GetNextRequest i SetRequest

Slika 2.12 prikazuje drugu vrstu SNMP PDU-a i nju koristi poruka *GetResponse*.

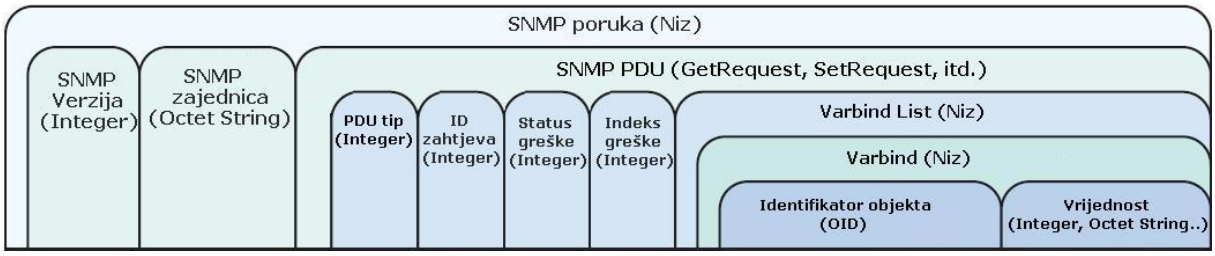
Slika 2.12 SNMP PDU za GetResponse

Treća i posljednja vrsta SNMP PDU-a se nalazi na slici 2.13 i nju koristi poruka *Trap*, generirana naravno od strane agenta.



Slika 2.13 SNMP PDU za Trap

ASN.1, prethodno opisan, definira format SNMP PDU-a. Pregled svih polja koji se mogu naći u prve dvije vrste PDU-a donosi tablica 2.3 a polja su pregledno prikazana na slici 2.14.



Slika 2.14 Polja SNMP poruke

Tablica 2.3 Pregled polja SNMP poruke

Polje	Opis
SNMP poruka	Niz koji predstavlja kompletnu SNMP poruku koja se sastoji od SNMP verzije, zajednice i SNMP PDU-a
SNMP verzija	Cijeli broj koji kazuje verziju SNMP-a (SNMPv1 = 0 itd.)
SNMP zajednica	*Octet String* koji sadrži niz znakova korišten radi sigurnosti
SNMP PDU	SNMP PDU sadrži tijelo SNMP poruke. Postoji više vrsta PDU-a
Tip PDU-a	Cijeli broj koji kazuje o kojoj vrsti PDU-a se radi
ID zahtjeva	Cijeli broj kojim se identificira pojedini SNMP zahtjev. Vraća se natrag u odgovoru SNMP agenta omogućujući SNMP upravljaču da upari odgovor sa odgovarajućim zahtjevom.
Status greške	Cijeli broj koji je postavljen na 0x00 u zahtjevu SNMP upravljača. SNMP agent stavlja kod greške u ovo polje ako se dogodila greška pri obradi zahtjeva. Neki od kodova greške su: 0x00 – nema greške 0x01 – poruka odgovora je prevelika za prenijeti 0x02 – ime zatraženog objekta nije nađeno 0x03 – tip podatka u zahtjevu nije odgovarao tipu podatka u SNMP agentu 0x04 – SNMP upravljač pokušao je postaviti *read-only* varijablu
Indeks greške	Ukoliko se dogodila greška, Indeks greške čuva pokazivač na objekt koji je uzrokovao grešku. Ako nema greške indeks je 0x00
Varbind List	Niz *Varbind*-a
Varbind	*Variable binding,* niz koji se sastoji od dva polja – identifikatora objekta i vrijednosti za taj objekt ili od tog objekta
Identifikator objekta	Pokazivač na pojedinu varijablu u SNMP agentu
Vrijednost	*SetRequest* PDU – vrijednost se postavlja na specifičan OID SNMP agenta *GetRequest* PDU – vrijednost je *null* i služi kao *place holder* za povratne podatke *GetResponse* PDU – povratna vrijednost od specifičnog OID-a SNMP agenta

Iako je tablica vrlo detaljna, potrebna su još neka dodatna pojašnjenja. Kod prvog tipa PDU-a (koji se koristi za poruke *GetRequest, GetNextRequest* i *SetRequest)* polja *Status greške* i *Indeks greške* postavljena su na nulu, budući da njih postavlja isključivo agent (a ove poruke šalje upravljač). *Varbind* je zapravo uređeni par (ime varijable, vrijednost varijable) a njihova lista je popis tih parova. Kod *GetRequest* i *GetNextRequest* poruka vrijednosti svih varijabli postavljene su na nulu (jasno, kod

SetRequest poruke u vrijednost varijable biti će upisana nova vrijednost koja se postavlja).

Ovom tablicom nisu pokrivena polja koja čine *trap* poruku. Ona su popisana u tablici 2.4, zajedno s opisom pojedinog polja.

Tablica 2.4 Polja SNMP PDU-a za trap poruku

Polje	Opis
Enterprise	Vrsta objekta koji generira *trap*
Agent-addr	Adresa objekta koji generira *trap*
Generic-trap	Vrsta generičkog *trapa* (npr. *linkDown* = 2)
Specific-trap	Informacija specifična za određenu vrstu *trapa*
Time-stamp	Vrijeme proteklo između stvaranja *trapa* i posljednje inicijalizacije mrežnog entiteta

Nakon pregleda svih polja, zanimljivo je pogledati sadržaj paketa uhvaćenog programom za snimanje mrežnog prometa Wireshark. Time se još bolje ilustrira kako SNMP zaista funkcionira "kroz žicu". Prikazana je operacija dohvata koja uključuje razmjenu dva paketa, a to su zahtjev upravljača i odgovor agenta:

```
No.     Time        Source      Destination Protocol Info
    112 1.936155    10.10.1.110 10.10.1.224 SNMP     GET SNMP...
Simple Network Management Protocol
    Version: 1 (0)
    Community: public
    PDU type: GET (0)
    Request Id: 0x2a7ee1af
    Error Status: NO ERROR (0)
    Error Index: 0
    Object identifier 1: 1.3.6.1.2.1.1.4.0 (SNMPv2-MIB::sysContact.0)
    Value: NULL


No.     Time        Source      Destination Protocol Info
    113 1.943425    10.10.1.224 10.10.1.110 SNMP     RESPONSE SN...
Simple Network Management Protocol
    Version: 1 (0)
    Community: public
    PDU type: RESPONSE (2)
    Request Id: 0x2a7ee1af
    Error Status: NO ERROR (0)
    Error Index: 0
    Object identifier 1: 1.3.6.1.2.1.1.4.0 (SNMPv2-MIB::sysContact.0)
    Value: STRING: Pero Peric
```

Kao što se vidi, zajednica je *public*, verzija je SNMPv1 a zatražena je vrijednost varijable `sysContact`. Prvi paket je tipa GET, a drugi RESPONSE (to jest, radi se o porukama *GetRequest* i *GetResponse*). U drugom paketu je polje Vrijednost (*Value*) popunjeno i predstavlja vrijednost tražene varijable odnosno u ovom slučaju ime osobe za kontakt.

2.5 SMI standard

SMI (*Structure of Management Information*), definiran u [11], predstavlja standard za izgradnju MIB baze. SMI je zapravo podskup od ASN.1, uz neka proširenja. Ovim standardom određuju se vrste podataka koje možemo koristiti u MIB-u, te način prikaza i imenovanja resursa MIB-a. Osnovni cilj je jednostavnost i proširivost MIB-a pa se stoga dopuštaju samo jednostavni tipovi podataka – skalari i dvodimenzionalna polja skalara. Prijetnju interoperabilnosti predstavlja činjenica da MIB-ovi određenih proizvođača sadrže tipove koji su napravili sami proizvođači. SMI mora ispuniti sljedeće zahtjeve kako bi se postigao standardizirani način prikaza upravljanih informacija:

1. Pružanje standardizirane tehnike definiranja strukture pojedinog MIB-a

2. Pružanje standardizirane tehnike definiranja pojedinih objekata MIB-a

3. Pružanje standardizirane tehnike kodiranja vrijednosti objekata MIB-a

Ti zahtjevi postižu se korištenjem identifikatora objekata, tipova objekata koji su definirani standardom ASN.1 i uporabom BER pravila kodiranja.

2.5.1 Makroi i SNMP varijable

SMI koristi tri važna makroa za definiranje modula, objekata i obavijesti. Prvi od njih je MODULE-IDENTITY koji se koristi prilikom opisivanja informacijskih modula. MIB moduli se inače definiraju kao kolekcije povezanih objekata. Drugi važan makro je OBJECT-TYPE koji se koristi prilikom definiranja objekata i on je ključan prilikom definiranja SNMP varijabli. NOTIFICATION-TYPE makro se koristi za definiranje obavijesnih poruka, najčešće prilikom korištenja operacije obavijesti.

Jedan od zadataka ovog rada je opisivanje elemenata ikev2 implementacije korištenjem ASN.1 notacije. To se nalazi u Dodatku A a isječak u nastavku donosi opis dviju varijabli. Prva zapravo i nije varijabla u pravom smislu riječi, već opis MIB modula uz pomoć makroa MODULE-IDENTITY. To predstavlja zgodan primjer kako treba definirati određeni modul. Druga varijabla je jedna od varijabli specifičnih za sam IKEv2 i ona je tipa *Octet-String*. Ona koristi makro OBJECT-TYPE za definiciju svojih parametara.

Isječak:

```
ikev2MIB MODULE-IDENTITY

      LAST-UPDATED "200809020000Z" -- 02 September 2008, 00:00

      ORGANIZATION "FER, ZEMRIS"

      CONTACT-INFO "email: ikev2-devel@zemris.fer.hr"

      DESCRIPTION "Mib for IKEv2 implementation."

      ::= {experimental 411249 }


ikev2pidFile OBJECT-TYPE

    SYNTAX      OCTET STRING

    MAX-ACCESS  read-only

    STATUS      current

    DESCRIPTION "Name and path of a pid file."

    ::= { ikev2MIB 4 }
```

Definicija svake SNMP varijable počinje sa već spomenutim makroom OBJECT-TYPE ispred kojeg dolazi ime varijable. Taj makro definira svojstva varijable i ima četiri obavezna parametra:

- SYNTAX – definicija podatkovnog tipa varijable
- MAX-ACCESS – informacija o pravu pristupa varijabli (čitanje/pisanje)
- STATUS – verzija SNMP-a s kojom je varijabla u skladu
- DESCRIPTION – kratki opis varijable

SMIv2 podržava 13 tipova podataka, te niz podatkovnih tipova (o ovome je bilo govora u poglavlju o ASN.1 tipovima podataka). Parametar STATUS poprima jednu od sljedećih vrijednosti: *current* (varijabla je u skladu s trenutnom verzijom SNMP-a), *obsolete* (varijabla nije u skladu s trenutnom, već zastarjelom verzijom SNMP-a) i *deprecated* (između *current* i *obsolete*). Nakon zadnjeg parametra (DESCRIPTION), slijedi znak "::=" iza kojeg se nalazi točan položaj dotične varijable u hijerarhijskom stablu (unutar vitičastih zagrada).

2.6 Operacije SNMP protokola

Pomno upoznavanje sa samim detaljima SNMP protokola nalazi se u ovom poglavlju i to kroz opis osnovnih operacija, od kojih su gotovo sve barem jednom spomenute do sada.

2.6.1 Operacija dohvata

Operacija dohvata već je puno puta dosad spominjana te je ovdje navedena radi potpunosti. Upravljač koristi poruku *GetRequest* za dohvat vrijednosti varijabli čiji OID mora zadati u tijelu te iste poruke. Kako je spomenuto potpoglavlju 2.5, postoje skalarni (jednostavni) tipovi i tablice. Svaki objekt MIB-a označen je u obliku x.y, gdje je x OID tog objekta a y oznaka instance. Ukoliko se radi o skalarnom objektu tada je y jednak nuli, a kod tablica je jednak indeksu retka u tablici. Agent odgovara upravljaču porukom *GetResponse* u čiji PDU upisuje parove (OID, vrijednost varijable) za svaku varijablu čiji OID je zadan u zahtjevu *GetRequest*. Naredba `snmpget` iz Net-SNMP paketa implementira operaciju dohvata i njena sintaksa je:

```
snmpget options hostname community objectID…
```

Primjer korištenja:

```
% snmpget -v 2c -c public test.net-snmp.org .1.3.6.1.2.1.1.3.0

system.sysUpTime.0 = Timeticks: (586752671) 67 days,
21:52:06.71
```

Opcije su zajedničke svim naredbama Net-SNMP paketa. `Hostname` predstavlja IP adresu (u numeričkom ili simboličkom obliku) mrežnog uređaja kojem se pristupa. U polje `community` upisuje se znakovni niz koji se (loše) koristi u svrhu sigurnosti u prve dvije verzije SNMP protokola. Konačno, `objectID` čuva identifikator objekta čija se vrijednost dohvaća. Najčešća greška prilikom korištenja ove naredbe je izostavljanje nule na kraju OID-a. Naime, OID varijable `sysUpTime` je .1.3.6.1.2.1.1.3, ali se prilikom dohvata mora dodati još sufiks ".0" na kraj OID-a. Razlog tome je što je varijabla `sysUpTime` skalar te se s tom nulom označava sama instanca.

2.6.2 Operacija dohvata sljedeće instance (*get-next*)

Poruka *GetNextRequest* vrlo je slična poruci *GetRequest* (imaju i isti oblik PDU-a). Ipak, postoji razlika. Porukom *GetRequest* upravljač može dobiti samo jednu instancu pojedinog objekta. Primjerice, ikev2 implementacija ima trenutno četiri sjednice koje održava i upravljač želi dobiti podatke o svakoj. *GetRequest* mu to ne omogućuje, osim ako upravljač eksplicitno ne navede OID svake od njih što nije problem kada su u pitanju četiri sjednice, ali što ako ih ima četiri stotine? Jasno da je takav pristup neefikasan i zato je tu poruka *GetNextRequest*. Njome se također dobiva samo jedna vrijednost, i to vrijednost instance objekta koja je slijedeća po leksikografskom redoslijedu. Leksikografski redoslijed se u ovom slučaju definira preko brojeva u OID-a, s lijeva na desno; prilikom usporedbe dva OID-a leksikografski prije dolazi onaj s većim brojem na istoj poziciji OID-a (ako su im svi

brojevi isti, prije dolazi onaj s dužim OID-om). Kako onda dobiti odjednom vrijednosti svih instanci kad i *GetNextRequest* vraća samo jednu vrijednost? `Snmpwalk` naredba koristi *GetNextRequest* za dohvat cijelog podstabla:

```
% snmpwalk -Os -c public -v 1 zeus system

sysDescr.0 = STRING: "SunOS zeus.net.cmu.edu 4.1.3_U1 1 sun4m"
sysObjectID.0 = OID: enterprises.hp.nm.hpsystem.10.1.1
sysUpTime.0 = Timeticks: (155274552) 17 days, 23:19:05
sysContact.0 = STRING: ""
sysName.0 = STRING: "zeus.net.cmu.edu"
sysLocation.0 = STRING: ""
sysServices.0 = INTEGER: 72
```

`Snmpwalk` zapravo koristi niz *GetNextRequest* poruka u petlji. Prva iteracija *GetNextRequest*-a koristi OID koji je zadan u komandnoj liniji, a svaka slijedeća iteracija koristi OID dobiven u prethodnom odgovoru. Petlja se zaustavlja ako je vraćeni OID izvan podstabla zadanog originalnim OID-om ili ako se došlo do kraja MIB-a. Jasno je da je takva naredba neizvediva samo sa *GetRequest* zahtjevima jer se nikako ne može dobiti OID sljedeće instance. U gornjem primjeru dohvaćene su sve varijable iz podstabla `system` i to s jednom jedinom naredbom.

Naravno, postoji i naredba `snmpgetnext` koja djeluje identično kao i `snmpget` naredba s tom razlikom da vraća slijedeću instancu objekta.

2.6.3 Operacija većinskog dohvata (*get-bulk*)

Prilikom izvršavanja operacije dohvata, agent je dužan upravljaču odgovoriti porukom *GetResponse* u kojoj specificira vrijednosti zatraženih objekata. Tih objekata može biti više, a duljina SNMP poruke je ograničena pa se logično nameće pitanje – što ako sve vrijednosti ne stanu u poruku? U tom slučaju agent vraća poruku o grešci i ne vraća se nikakva vrijednost. Poboljšanje dolazi sa verzijom SNMPv2 i to uvođenjem operacije većinskog dohvata kojom se zahtijeva od agenta da vrati onoliko vrijednosti koliko stane u poruku. PDU je prikazan na slici 2.15:

Slika 2.15 SNMP PDU za operaciju većinskog dohvata

Nova polja su *non-repeaters* i *max-repetitions*. Prvo polje određuje broj varijabli za koje agent mora vratiti po jednu vrijednost (koristi se najčešće za skalare). To znači da ako su u polju *variablebindings* zatražene vrijednosti za pet varijabli, a u polju *non-repeaters* je upisana vrijednost dva, tada za prve dvije varijable (po leksikografskom redoslijedu) agent mora vratiti samo jednu vrijednost. Drugo polje određuje najveći broj vrijednosti koje agent mora vratiti za preostale varijable. U prethodnom primjeru preostale su tri varijable, i za njih agent mora vratiti (najviše)

onoliko vrijednosti koliko je specificirano u polju *max-repetitions*. Jasnije će biti na konkretnom primjeru (naredba Net-SNMP paketa je `snmpbulkget`):

```
%snmpbulkget -v 2c -Cn2 -Cr3 -c public localhost laLoad
ifInOctets ifOutOctets

UCD-SNMP-MIB::laLoad.1 = STRING: 0.63

IF-MIB::ifInOctets.1 = Counter32: 35352440

IF-MIB::ifOutOctets.1 = Counter32: 35352440

IF-MIB::ifOutOctets.2 = Counter32: 297960502

IF-MIB::ifOutOctets.3 = Counter32: 0
```

Opcija –v2c je obavezna i specificira da se radi o SNMPv2 (u SNMPv1 nema ove naredbe). Opcije –Cn2 i –Cr3 definiraju vrijednosti polja *non-repeaters* i *max-repetitions*. U konkretnom slučaju, vrijednost prvog polja je 2 pa će za dvije varijable, `laLoad` i `ifInOctets` biti vraćena jedna vrijednost, a za preostalu varijablu (`ifOutOctets`) najviše tri vrijednosti jer je vrijednost drugog polja 3. U ovom slučaju su zaista i vraćene tri vrijednosti, ali to ne mora biti slučaj bilo da tri instance jednostavno ne postoje ili se premašila maksimalna dozvoljena duljina SNMP poruke.

2.6.4 Operacija postavljanja

Operacija postavljanja je (kao i dohvat) već detaljno opisana pa u ovom dijelu još nekoliko informacija. Koristi se za postavljanje vrijednosti u MIB-u agenta ili za stvaranje novog retka tablice. To se postiže slanjem parova (OID, vrijednost). Nakon primitka poruke *SetRequest*, agent odgovara porukom *GetResponse* u kojoj specificira da li je došlo do pogreške. Pogreška se označava odgovarajućim kodom, navedenim u tablici 2.3 ili se upisuje nula ako je sve prošlo u redu. Verzija SNMPv2 proširila je skup mogućih tipova grešaka (u verziji SNMPv1 bilo ih je samo pet). Odgovarajuća naredba je `snmpset`:

```
snmpset -c private -v 1 test-hub system.sysContact.0 s
pero.peric@fer.hr
```

Sintaksa ove naredbe je slična sintaksi naredbe `snmpget` ali na kraju slijede trojke OID-tip-vrijednost. Tip je u gornjem primjeru "s" što označava znakovni niz. Naravno, tipovi u sintaksi ove naredbe korespondiraju s odgovarajućim ASN.1 tipovima.

2.6.5 Operacija obavijesti

Operacija obavijesti služi za obavještavanje upravljača o bitnim događajima koji se zbivaju na strani agenta. Komunikacija je jednosmjerna, od agenta prema upravljaču. Budući da se koristi protokol UDP, nema garancije da će upravljač zaista i dobiti tu poruku ali se ovaj način obavještavanja svejedno često koristi u svakom sustavu za upravljanje mrežom.

Neke od primjena ove operacije navedene su u poglavlju 2.4. Ovdje će biti spomenuta još jedna – prozivanje na temelju obavijesti. U stvarnim sustavima za upravljanje postoji vrlo veliki broj agenata (s puno objekata) i njihovo prozivanje od strane upravljača je često nepraktično i neefikasno. Zato se koristi prozivanje uz pomoć obavijesti koje rasterećuje mrežu i upravljača. Ono se temelji na tome da upravljač u određenom periodu (ali ne često) proziva sve agente a ostatak vremena se oni sami jave pomoću obavijesti ukoliko dođe do nepredviđenih događaja. U tom slučaju upravljač proziva tog agenta koji mu se javio ili obavi neke automatske radnje. Ovime se rasterećuju i sami agenti jer se štedi njihovo procesorsko vrijeme.

2.7 Razvoj SNMP protokola i sigurnost

Nakon cjelokupnog pregleda ovog protokola, biti će lakše razumjeti novosti koje su uvedene u drugoj i trećoj verziji. Ovaj rad se bavi s mrežnom sigurnosti pa je napravljena i kratka analiza sigurnosnih nedostataka. Time ujedno i završava poglavlje o SNMP-u.

2.7.1 SNMPv2

U ovoj verziji proširene su funkcionalnosti koje nudi SNMPv1. Operacijama dohvata i postavljanja definirala se komunikacija na razini upravljač-agent (i to u ovom slučaju kroz zahtjev upravljača i odgovor agenta). Operacija obavijesti definira komunikaciju od agenta prema upravljaču (bez potvrde upravljača). SNMPv2 to proširuje i dodaje još jednu razinu komunikacije: upravljač-upravljač. To se provodi na način da jedan upravljač drugom pošalje obavijest o informaciji koju posjeduje. U tu svrhu definira se nova operacija, informiranje *(inform*) te se koristi nova poruka, *InformRequest*. Drugi upravljač tada potvrđuje primitak ove poruke slanjem *Response* PDU-a prvom upravljaču.

Promjena je i kod PDU-a za obavijest. Format tog PDU-a identičan je formatu *GetRequest*, *GetNextRequest*, *SetRequest* i *InformRequest* PDU-a korištenog u SNMPv2. Prve tri nabrojane poruke imaju isti format kao u prvoj verziji SNMP protokola. *Response* PDU kojim drugi upravljač odgovara prvom nakon primitka poruke *InformRequest* jednak je *GetResponse* PDU-u iz SNMPv1.

SNMPv2 definira i operaciju većinskog dohvata opisanu u poglavlju 2.6.3. Ona ne postoji u prvoj verziji SNMP-a.

2.7.2 SNMPv3

Najveće promjene koje donosi treća verzija SNMP protokola odnose se na sigurnost i uvođenje entiteta.

Umjesto upravljača i agenta, SNMPv3 uvodi pojam entiteta. Svaki entitet se sastoji od pogona i aplikacija. Pogon se pak dijeli na četiri podsustava: dispečerski, podsustav za obradu poruka, sigurnosni i podsustav za upravljanje pristupom.

Dispečerski podsustav bavi se primanjem i slanjem poruka. Podsustav za obradu poruka priprema poruke za slanje drugim entitetima i izvlači informacije iz primljenih poruka. Sigurnosni podsustav pruža mogućnost enkripcije i autentifikacije. Enkripcija

se obavlja kriptiranjem DES algoritmom. Entiteti se autentificiraju preko zajednica ako se radi o SNMPv1 i SNMPv2 porukama zbog tzv. kompatibilnosti unatrag (*backward compatibility*), odnosno korisnički (*user-based*). Provjera vjerodostojnosti obavlja se pomoću algoritama za sažimanje, MD5 ili SHA. Ovime se rješava glavni nedostatak prethodnih verzija protokola – slanje šifre u čitljivom obliku. Podsustav za upravljanje pristupom bavi se upravljanjem pristupom korisnika pojedinim objektima. Dakle, u novoj arhitekturi dva od četiri podsustava direktno su povezana sa sigurnošću!

Osim SNMP pogona, entitet se sastoji i od aplikacija. One su skupa sa funkcijama koje obavljaju popisane u tablici 2.5.

Tablica 2.5 Aplikacije entiteta

Aplikacija	Funkcija
generator naredbi	generira zahtjeve *get*, *get-next*, *get-bulk* i *set* te obrađuje odgovore
generator odgovora	šalje odgovore na zahtjeve *get*, *get-next*, *get-bulk* i *set*
generator obavijesti	generira obavijesti
prijemnik obavijesti	prima obavijesti i poruke *inform*
proxy forwarder	olakšava prosljeđivanje SNMP poruka između entiteta

2.7.3 Sigurnost

SNMPv3 je donio velike pomake na razini sigurnosti. Ipak, ovaj protokol je i dalje ranjiv. Grupa za etičko hakiranje, GNUCitizen, provela je eksperiment kojim je pokazala da veliki broj mrežnih uređaja koji podržavaju SNMP može odati osjetljive informacije. Eksperiment je proveden na način da je slučajno odabrano 2,5 milijuna IP adresa koje su zatim skenirane SNMP-om. Zabrinjavajuće velik broj uređaja bez problema je otkrio ime, model i verziju operacijskog sustava.

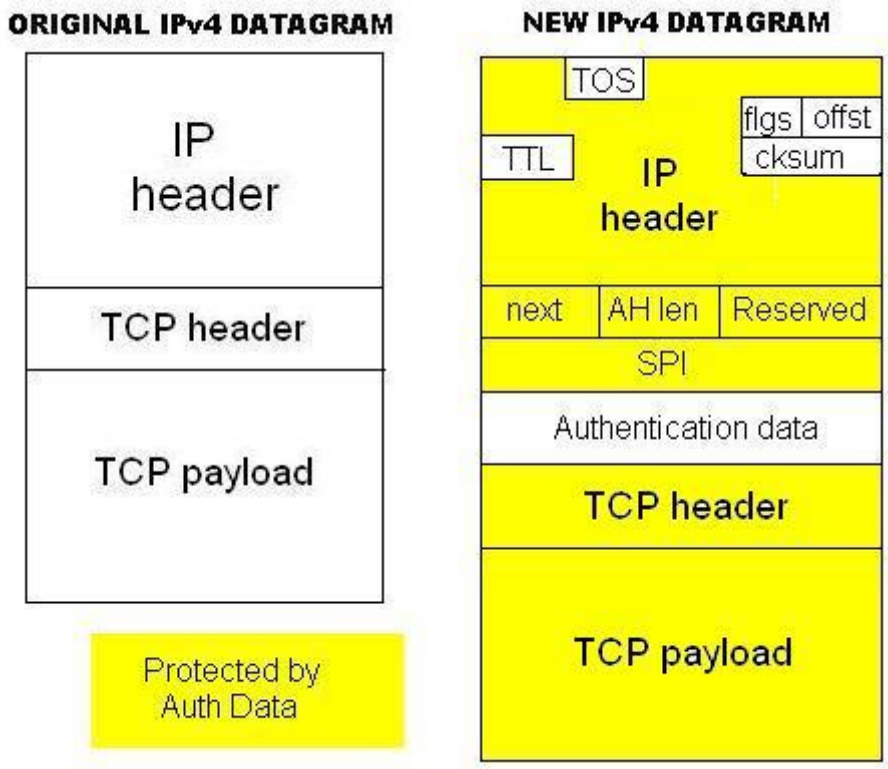
Od brojnih nedostataka koje SNMP ima u aspektu sigurnosti, izdvaja se podložnost napadima grubom silom (*brute force*) te napadima koji se baziraju na rječniku. Kako se SNMP temelji na UDP protokolu, skeniranje sustava može se obaviti puno brže nego što je to slučaj s TCP-om. UDP donosi još jednu ranjivost, a to je napad koji se zasniva na krivotvorenju IP adrese.

3. IPsec

IPsec arhitektura, za razliku od ostalih sigurnosnih rješenja i arhitektura na Internetu, djeluje na mrežnom sloju OSI modela i po tome je specifična. U novoj generaciji Interneta, IPv6, IPsec je obavezna dok je kod IPv4 opcionalna. Osim tri glavna protokola (ESP, AH i IKEv2), IPsec definira i koristi dvije vrlo bitne baze podataka – bazu sigurnosne politike (*Security Policy Database*, SPD) i bazu sigurnosnih udruživanja (*Security Association Database*, SAD). Ove dvije baze podataka kroz svoje zapise odgovaraju na dva važna pitanja – koji promet štititi i kako ga štititi. U SPD bazi administrator definira sigurnosnu politiku (*security policy*) i to na način da definira u kojem rasponu IP adresa je štićeni promet, koji protokol se koristi, razina zaštite itd. Najvažniji dio SPD baze su tzv. selektori prometa (*traffic selectors*) koji specificiraju koje pakete štititi preko izvorišne i odredišne IP adrese, protokola višeg sloja i pristupa. SAD baza čuva pak tzv. sigurnosna udruživanja (*security association*) koja su zapravo skup sigurnosnih parametara, npr. kriptoalgoritmi korišteni u komunikaciji. Svako sigurnosno udruživanje jedinstveno je definirano protokolom (AH ili ESP), odredišnom IP adresom i SPI-em (*Security Parameters Index* – 32-bitni cijeli broj). Zapisi u jednoj i drugoj bazi nalaze se zapravo u samoj jezgri operacijskog sustava. Sami zapisi dolaze u parovima; obje baze čuvaju po jedan zapis za odlazeće i jedan za dolazeće pakete.

3.1 AH

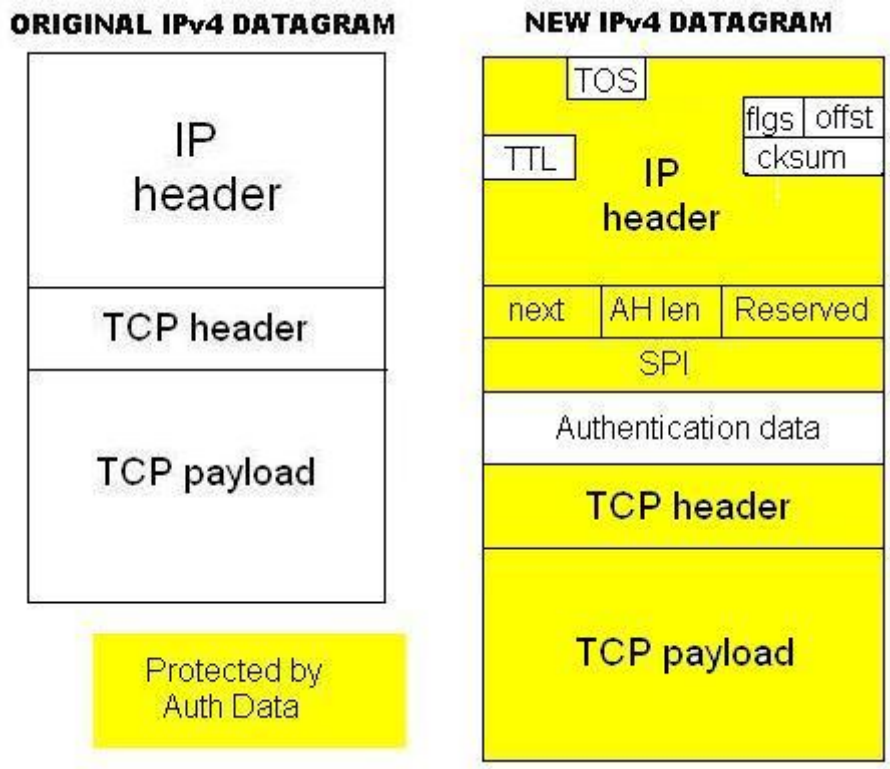
Authentication Header je protokol koji osigurava autentičnost i besprijekornost. Iako je prije bio obavezan za implementaciju, sada je opcionalan i iz tog razloga ESP ima prednost. Kako se može vidjeti na slici 3.1, AH štiti IP sadržaj (*payload*) i cijelo IP zaglavlje osim promjenjivih polja (kao što je npr. *Time To Live* čija vrijednost se dekrementira prolaskom kroz svaki usmjernik). AH zaglavlje smješteno je između IP zaglavlja i IP sadržaja (koji je zapravo zaglavlje i sadržaj viših slojeva). Autentičnost i zaštita integriteta postižu se pomoću tzv. vrijednosti za provjeru integriteta (*Integrity Check Value,* ICV) koja se računa iz čuvanih podataka i dijeljene tajne.



Slika 3.1 AH paket u transportnom načinu rada

3.2 ESP

Encapsulating Security Payload je protokol koji nudi prvenstveno tajnost, a opcionalno i autentičnost te besprijekornost. Tajnost se postiže pomoću kriptiranja nekim od dogovorenih simetričnih kriptoalgoritama. Kriptiraju se IP sadržaj, dopuna (*padding*) te "Next header" polje (slika 3.2). Ovime potencijalni napadač ne zna čak ni koji tip protokola višeg sloja je enkapsuliran. Autentičnost se ostvaruje pokrivanjem manjeg broja polja nego kod AH protokola – samo ESP zaglavlja i IP sadržaja.



Slika 3.2 ESP paket u transportnom načinu rada

3.3 Načini rada

IPsec specificira dva osnovna način rada – transportni (*transport)* i tunelirajući (*tunnel)*. Transportni način rada je prikladan za korištenje kada je komunikacija *end-to-end*. U ovom načinu rada, postoji samo jedna izvorišna i odredišna IPv4 adresa i obje strane u komunikaciji moraju imati IPsec implementaciju.

Tunelirajući način rada je bolji za komunikaciju između sigurnosnih prilaza (*security gateways*). U ovom slučaju računala iza prilaza ne moraju uopće imati IPsec implementaciju, već jedino sami sigurnosni prilazi. U ovom slučaju komunikacija se uspostavlja unutar IPsec tunela. To vodi na još jedan par IP adresa i dalje, na dodatno zaglavlje pored klasičnog, "vanjskog" IP zaglavlja. ESP enkripcija sada pokriva čitavi IPv4 datagram zajedno s unutarnjim zaglavljem. AH čuva i provjerava besprijekornost i od unutarnjeg i od vanjskog zaglavlja te naravno, IP sadržaja. Provjera besprijekornosti uspješno otkriva pokušaje promjene sadržaja paketa od uljeza na nesigurnoj mreži.

3.4 IPsec uporaba – VPN

U budućnosti, kada IPv6 zaživi, IPsec će biti obavezan za korištenje a dotada se njegova upotreba najviše temelji na virtualnim privatnim mrežama (Virtual Private Networks). Iako postoje razne definicije ovisno o veličini mreže, može se reći da je virtualna privatna mreža sigurni, privatni komunikacijski tunel između dva ili više uređaja preko javne mreže (npr. Interneta). Promet unutar VPN tunela je kriptiran i

drugi korisnici ga ne mogu čitati makar sam promet bio i snimljen. Implementiranjem VPN-a, tvrtka može ponuditi pristup internoj mreži korisnicima iz cijelog svijeta. Prije pojave VPN-a, udaljeni (*remote*) zaposlenici pristupali su internoj mreži tvrtke preko iznajmljenih vodova ili preko udaljenih *dial-up* poslužitelja. S VPN-om, oni su dobili najisplativiji način za spajanje, tuneliranjem preko javne mreže.

Internet VPN rješenje bazira se na klijent/poslužitelj arhitekturi. Klijent koji se želi spojiti na internu tvrtkinu mrežu prvo mora ostvariti pristup Internetu preko bilo kojeg davatelja Internet usluga (*Internet Service Provider*). Nakon toga, on uspostavlja VPN konekciju prema tvrtkinom VPN serveru preko VPN klijenta koji je instaliran na samom klijentskom računalu. Jednom kad je komunikacija uspostavljena, udaljeni zaposlenik komunicira sa ostatkom korporacijske mreže preko Interneta baš kao da je lokalno računalo u toj mreži.

Protokoli drugog sloja koji se koriste za uspostavu VPN konekcije su PPTP (Microsoftovo vlasništvo) i L2TP (Ciscovo vlasništvo). L2TP je često korišten upravo u kombinaciji s IPsec-om koji služi za enkripciju i time za bolju sigurnost. Kao takav, IPsec je jedna od najraširenijih VPN tehnologija u današnjim mrežama.

4. IKEv2 protokol

Jedan od tri glavna protokola IPsec arhitekture, IKEv2 protokol, osigurava autentičnost, autorizaciju i najvažnije, razmjenu ključeva. No unatoč tome, IKEv2 protokol nije obavezan za implementaciju u sklopu IPsec-a. Naime, iako ESP i AH protokoli zahtijevaju ključeve u SAD bazi da bi se uspostavila sigurna komunikacija, ti ključevi mogu se unijeti i ručno. Odmah su vidljivi brojni problemi tog pristupa:

- sigurnost - kako na siguran način razmijeniti dijeljeni ključ preko nesigurne mreže
- veliki broj unaprijed dodijeljenih ključeva - za svako sigurnosno udruživanje potreban je drugi ključ
- ručno unošenje ključeva - zamorno, velika mogućnost pogreške
- *rekeying* - nakon nekog vremena potrebni su novi ključevi

Ove, i druge probleme, rješava IKEv2 protokol.

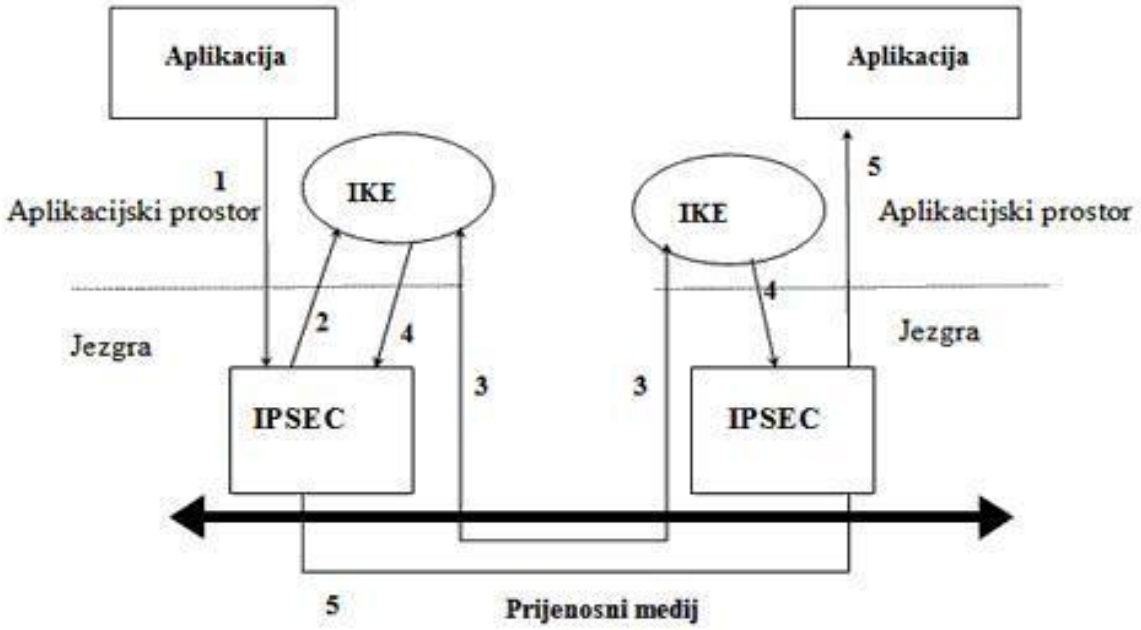
4.1 Osnovne značajke

IKEv2 protokol pokreće se preko *daemona* (na Linux/Unix operacijskim sustavima). Radi se o neinteraktivnom programu koji "leži" u pozadini i čeka odgovarajući okidač da se uključi. U konkretnom slučaju, *daemon* se registrira na UDP pristup 500 i čeka dolazak paketa na taj pristup ili poruku od jezgre operacijskog sustava u slučaju odlazećeg paketa. Sam *daemon* generira simetrične ključeve i nakon nekog vremena izvodi *rekeying*.

Poruke IKEv2 protokola prenose se UDP protokolom u parovima zahtjev-odgovor. IKEv2 za entitete u komunikaciji uvodi termine *initiator* i *responder*. Prvo sigurnosno udruženje, IKE SA, kreira se tijekom IKE_SA_INIT razmjene. To udruženje izraženo je prvim zapisom u SAD bazi koji se sastoji od IP adresa *initiatora* i *respondera*, dogovorenih kriptoalgoritama, SPI brojeva itd. U samoj IKE_SA_INIT razmjeni *initiator* i *responder* nude jedan drugom prihvatljive sigurnosne parametre u komunikaciji. Nakon dogovora oko sigurnosnih parametara na redu je sljedeća razmjena – IKE_AUTH_INIT. Ona je nužna budući da se za vrijeme prve razmjene, IKE_SA_INIT, *initiator* i *responder* nisu jedan drugom predstavili odnosno autentificirali. Sama autentifikacija provodi se dakle također preko IKEv2 protokola. Za autentifikaciju koristi se neka od sljedećih podržanih metoda: unaprijed dodijeljeni ključevi (*pre-shared keys*), digitalni certifikati ili *Extensible Authentication Protocol* (EAP). Ako se koristi EAP onda imamo dodatne poruke. Nakon autentifikacije stvara se i drugo sigurnosno udruženje, tzv. CHILD SA (CSA), kojim se prenosi štićeni promet.

4.2 IPsec scenarij s korištenjem IKEv2

Slika 4.1 pokazuje tipični scenarij korištenja IKEv2 *daemona* prilikom uspostavljanja SA. Na lijevoj strani prikazan je *initiator* a na desnoj *responder*.



Slika 4.1 Uspostava sigurnosnog udruženja uz korištenje IKEv2 daemona

Komunikacija se odvija na sljedeći način:

1. Aplikacija na strani *initiatora* želi komunicirati s aplikacijom na strani *respondera*. Prilikom slanja paketa u jezgri operacijskog sustava provjerava se stanje SPD baze. Ako je pronađen zapis koji kaže da treba štititi promet između tih dviju IP adresa, provjerava se postoje li ključevi u SAD bazi za sigurno komuniciranje.

2. Budući da je SAD baza prazna, treba je napuniti. To je moguće ili ručno, ili bolje – pozivanjem IKE *daemona*. Jezgra operacijskog sustava šalje *daemonu* poruku SADB_ACQUIRE kojom zahtijeva punjenje SAD baze.

3. *Daemon* na strani *initiator*a započinje komunikaciju s *daemonom* na strani *respondera.* Jedan drugom nude sigurnosne parametre koji im odgovaraju za komunikaciju (npr. *initiatoru* odgovara AES algoritam za kriptiranje, a MD5 za

autentifikaciju i sada se traži presjek s algoritmima koji odgovaraju *responderu*). Ukoliko se dogovore, moguća je sigurna komunikacija

4. *Daemoni* na obje strane upisuju sigurnosna udruženja (SA) u SAD baze. Stvorena su sigurnosna udruženja IKE SA i CHILD SA i sve je spremno za sigurnu komunikaciju.

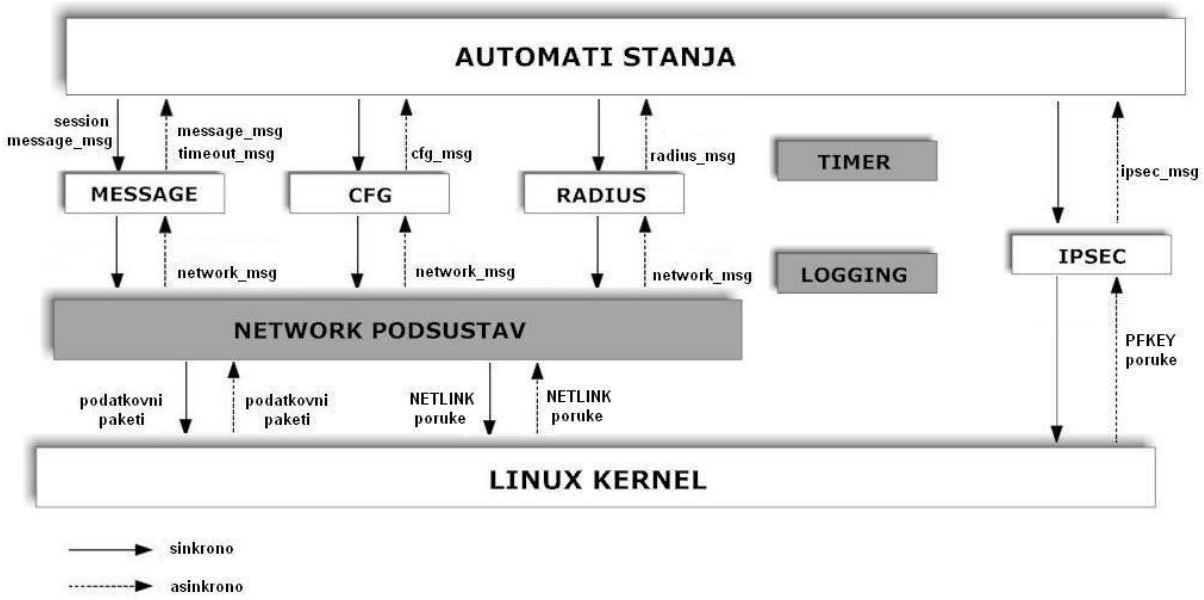
5. Prenosi se zaštićeni promet i u ovom koraku *daemoni* nisu aktivni osim u slučaju izvanrednih okolnosti (napadi i sl.). Nakon isteka određenog vremenskog perioda potrebno je provesti *rekeying* odnosno stvoriti nove ključeve.

5. Arhitektura ikev2 implementacije

Ikev2 implementacija razvijena je modularno te je strukturalno podijeljena na nekoliko neovisnih podsustava. Najvažniji dio cjelokupne implementacije su automati stanja. Glavni automati stanja su oni koji se odnose na ponašanje *initatora* (sedam stanja), *respondera* (također sedam stanja) te automati stanja za posebne situacije, npr. stvaranje dodatnog CHILD SA (osam stanja). Ova implementacija razvijena je u programskom jeziku C za operacijski sustav Linux a njen detaljni opis nalazi se u [12]. Razumijevanje osnova ikev2 arhitekture nužno je radi efikasne ugradnje SNMP modula.

5.1 Statički pogled

Slika 5.1 prikazuje statički pogled na implementaciju protokola za razmjenu ključeva.



Slika 5.1 Statički pogled na ikev2 arhitekturu

Jasno se razlučuju pojedini sustavi – najvažniji su RADIUS, CFG i MESSAGE, te mrežni podsustav (NETWORK) koji predstavlja posrednika između spomenutih podsustava i okoline. Podsustavi komuniciraju asinkrono putem redova i u nekim slučajevima, pomoću *callback* mehanizma. Kratki opis funkcija svih podsustava nalazi se u nastavku.

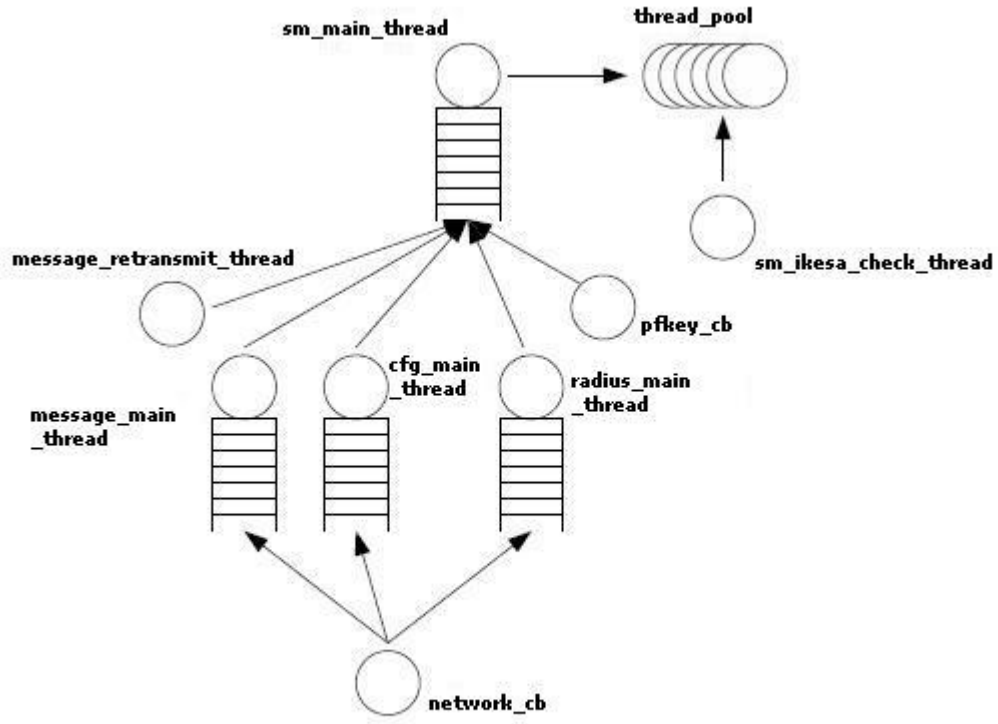
Sjenčano su označeni dijelovi implementacije koji su vezani za *framework*, a bez sjenčanja dijelovi vezani direktno za implementaciju. Prema tome, TIMER i LOGGING podsustavi mogu se koristiti u bilo kojoj implementaciji nekog protokola koja se temelji na ovom *frameworku*. Ta proširivost i mogućnost ponovne uporabe koda bio je jedan od važnijih ciljeva prilikom implementiranja IKEv2 protokola.

Kako se vidi na slici 5.1, dva su glavna ulaza u aplikaciju. Prvi je sama mreža i paketi koji dolaze s mreže. Oni se primaju u NETWORK podsustavu koji ih nakon kraće obrade enkapsulira u *network_msg* strukturu (posebno definirana struktura u ikev2 implementaciji) te šalje višim slojevima, odnosno odgovarajućim podsustavima. To konkretno znači da ako je primjerice primljena poruka IKEv2 poruka, tada se šalje MESSAGE podsustavu, ako je RADIUS poruka šalje se RADIUS podsustavu a ako je pak poruka vezana za DHCP, šalje se u CFG podsustav. Svaki od tih podsustava nakon što primi poruku i obavi potrebno procesiranje stvara svoj odgovarajući format poruke koji se dalje šalje automatu stanja.

Drugi ulaz u aplikaciju dolazi iz jezgre operacijskog sustava, i to konkretno iz SPD i SAD baze. Jedan od primjera je zahtjev jezgre operacijskog sustava prema *daemonu* da pribavi odgovarajuće ključeve nakon što se utvrdilo da se određeni promet mora štititi. IPsec podsustav komunicira s jezgrom preko PF_KEY sučelja. Mrežni podsustav razmjenjuje poruke s jezgrom preko NETLINK sučelja. Treba spomenuti da je IKEv2 *daemon* zamišljen da koristi bilo koje od ta dva sučelja, već prema potrebama.

5.2 Dinamički pogled

Dinamički pogled na arhitekturu prikazan je na slici 5.2.



Slika 5.2 Dinamički pogled na ikev2 arhitekturu

Pod "dinamičkim" se podrazumijevaju obje dretve (prikazane kružićima na slici 5.2) i komunikacijski kanali između njih. Dretve se dijele na dva tipa: prve, koje su aktivne tijekom cijelog životnog ciklusa *daemona* i druge, koje se aktiviraju pozivom neke druge dretve. Na slici 5.2 prve su označene sufiksom "_thread" a druge sufiksom "_cb". Dok su dretve označene kružićima, redovi su prikazani pravokutnicima. Prefiks ispred imena dretve i reda označava kojem podsustavu pripadaju.

Kako je objašnjeno u dijelu o statičkom pogledu, NETWORK podsustav "hvata" sve poruke s mreže i dalje ih šalje višim slojevima nakon kratke obrade. Sada je jasno da taj posao obavlja dretva *network_cb*, s time da nakon obrade te poruke završavaju u redovima pojedinih podsustava. U svakom podsustavu poruka se dekodira i obrađuje te zatim šalje u *thread pool* u kojem se obavlja glavno procesiranje zahtjeva i generiraju odgovori. Veličina *thread poola* može se konfigurirati te je sam izbor broja dretvi vrlo bitan budući da o njemu ovisi efikasnost i brzina *daemona*.

5.3 Podsustavi

U ovom odjeljku dan je kratki opis funkcija pojedinog podsustava.

5.3.1 NETWORK podsustav

Jedna od važnijih funkcionalnosti koje mrežni podsustav nudi ostalim dijelovima sustava je mogućnost registracije, čime se otvara utičnica (*socket*). Ona može biti anonimna tj. služiti samo slanju ili se može vezati za konkretnu adresu i pristup. Prilikom registracije (odnosno otvaranja utičnice), važan parametar je red u koji se šalju pristigle poruke nakon obrade. Utičnica se može zatvoriti bilo kad tijekom rada NETWORK podsustava. Prilikom slanja koje je sinkrono (jer se ne očekuje blokiranje) ne treba stvoriti utičnicu budući da se ona stvara dinamički.

Osim slanja i primanja poruka, uloga ovog podsustava je i nadzor nad mrežnim sučeljima i IP adresama. Razlog tomu je bolja kontrola aplikacije nad podacima koje razmjenjuje, kao i to da neke aplikacije zahtijevaju informaciju o dostupnim IP adresama i sučeljima.

5.3.2 TIMER podsustav

Mehanizmi za podešavanje isteka vremena nalaze se u ovom podsustavu. Nakon isteka konfiguriranog vremena, sustav koji se registrirao obavještava se o nastupu isteka. Sami alarmi mogu se podesiti da se jave jednom, ili u intervalima. Obavještavanje se može namjestiti preko redova ili *callback* funkcija koje se alociraju u odvojenoj dretvi. U tom slučaju procesiranje mora biti brzo i efikasno da se izbjegne zakrčenje.

5.3.3 LOGGING podsustav

Za logiranje je zadužen LOGGING podsustav. Pod logiranjem se podrazumijeva ispis obavijesti i poruka koje pojedina funkcija generira. Mogućnosti su zaista široke, a najpreglednije je koristiti ispis u posebnu datoteku. Obavijesti se razlikuju ovisno o

tome da li se radi o obavijestima o greškama, upozorenjima, početku ili kraju funkcije itd. Primjer dijela ispisa iz datoteke:

```
1211473732.448 proposals DEBUG - Authentication proposals
1211473732.448 proposals DEBUG - AUTH_HMAC_SHA1_96
1211473732.448 proposals DEBUG - Encryption proposals
1211473732.448 proposals DEBUG - PRF proposals
1211473732.448 proposals DEBUG - DH group proposals
1211473732.448 proposals DEBUG - Entering proposal_list_find_transform: 694
1211473732.448 proposals TRACE - proposal_list_find_transform: 695: type=3, protocol=2
1211473732.448 proposals DEBUG - PROPOSAL DUMP: IKEV2_PROTOCOL_AH (2)
1211473732.448 proposals DEBUG - PROPOSAL DUMP: spi_size=4, spi=2712737978
1211473732.448 proposals DEBUG - Authentication proposals
1211473732.448 proposals DEBUG - AUTH_HMAC_SHA1_96
1211473732.448 proposals DEBUG - Encryption proposals
1211473732.448 proposals DEBUG - PRF proposals
1211473732.448 proposals DEBUG - DH group proposals
1211473732.448 sm DEBUG - PROPOSAL DUMP: IKEV2_PROTOCOL_AH (2)
1211473732.448 sm DEBUG - PROPOSAL DUMP: spi_size=4, spi=2712737978
1211473732.448 sm DEBUG - Authentication proposals
1211473732.448 sm DEBUG - AUTH_HMAC_SHA1_96
1211473732.448 sm DEBUG - Encryption proposals
1211473732.448 sm DEBUG - PRF proposals
1211473732.448 sm DEBUG - DH group proposals
1211473732.448 ts DEBUG - Entering ts_lists_are_subset: 707
1211473732.448 ts DEBUG - (172.16.21.0/32 - 172.16.21.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - (192.168.0.0/32 - 192.168.0.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - (172.16.21.0/32 - 172.16.21.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - (192.168.0.0/32 - 192.168.0.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - Leaving ts_lists_are_subset: 825
1211473732.448 ts DEBUG - Entering ts_lists_are_subset: 707
1211473732.448 ts DEBUG - (172.16.21.0/32 - 172.16.21.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - (192.168.0.0/32 - 192.168.0.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - (172.16.21.0/32 - 172.16.21.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - (192.168.0.0/32 - 192.168.0.255/32, icmp, 0 - 65535)
1211473732.448 ts DEBUG - Leaving ts_lists_are_subset: 825
1211473732.449 crypto DEBUG - Entering integ_keylen_get:1027
1211473732.449 crypto TRACE - integ_keylen_get:1037: sha1 - 20
1211473732.449 crypto DEBUG - Entering prf_plus: 435
1211473732.449 crypto TRACE - prf_plus: 436: keylen=16
```

Slika 5.3 Dio ispisa datoteke za logiranje

5.3.4 MESSAGE podsustav

Funkcije za stvaranje i obradu poruka su grupirane u poseban podsustav. Nakon što se IKEv2 poruka primi s mreže, NETWORK podsustav šalje poruku ovome podsustavu koji ju verificira i obrađuje te stvara odgovarajuću strukturu koja se dalje šalje automatu stanja. Za vrijeme slanja IKEv2 poruka, ponovo se poziva MESSAGE podsustav ali ovaj put od strane automata stanja. Postupak slanja poruke počinje generiranjem poruke i to pozivanjem funkcije za stvaranje odgovarajućeg sadržaja a nastavlja se prosljeđivanjem poruke NETWORK podsustavu koji onda zaista šalje poruku na predviđenu destinaciju.

Generiranje poruka je složen zadatak zbog velikog broja mogućih kombinacija sadržaja. Zbog toga je generiranje složenih poruka često iterativan postupak i odvija se u više koraka.

5.3.5 CFG podsustav

IKEv2 *daemon* pomoću ovog podsustava može dohvatiti IP konfiguracijske parametre iz vanjskih izvora. Svaki potencijalni izvor predstavljen je pojedinim DHCPv4 pružateljem, DHCPv6 pružateljem ili pak privatnim pružateljem. DHCPv4 pružatelj je specifičan po tome što se ponaša kao kombinacija DHCPv4 klijenta i posrednika.

5.3.6 RADIUS podsustav

RADIUS podsustav koristi se tijekom EAP autentifikacije i služi kao posrednik RADIUS poruka od *respondera* prema RADIUS serveru i obratno. Ovaj podsustav sadrži funkcije za primanje/slanje RADIUS paketa od/k RADIUS poslužitelju, kao i funkcije za slaganje poruka, izvlačenje EAP paketa iz poruka, parsiranje poruka i slično.

5.3.7 Ostali podsustavi

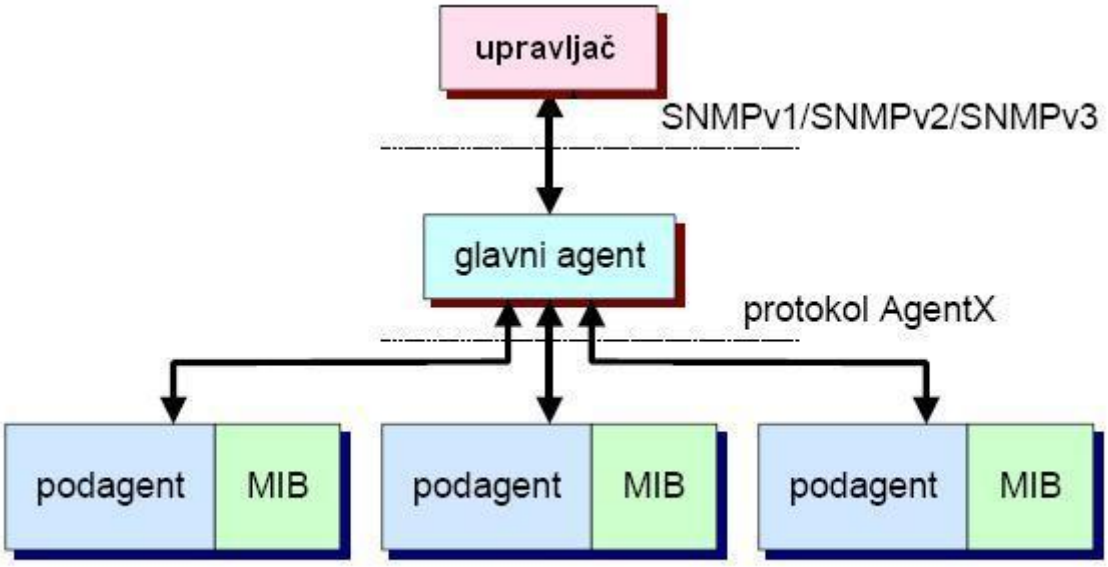
Na slici 5.1 nisu prikazani, ali ikev2 koristi i još neke podsustave. To su NETLIB, koji se bavi manipulacijom mrežnim adresama, CONFIG koji se koristi za čitanje i obradu konfiguracijske datoteke *daemona* i CRYPTO koji sadrži često korištene kriptografske funkcije.

6. Praktični rad – SNMP podrška u IKEv2 okruženju

6.1 Ugradnja SNMP modula

6.1.1 AgentX protokol

Prije same ugradnje modula u ikev2 implementaciju, potrebno je promotriti ponašanje modula u odnosu na "vanjski svijet". U sustavu za upravljanje mrežom opisanom u poglavlju 2.1, glavni entiteti su upravljač i agent. Jasno je da će ikev2 implementacija i njen SNMP modul imati ulogu agenta, budući da je modul taj koji registrira upravljane varijable i njihove vrijednosti nudi upravljačima na mreži. Međutim, to nije sve. AgentX (*Agent eXtensibility*) [15] protokol razrađuje tu arhitekturu uvođenjem hijerarhije među agentima. Glavni agent (*master*) je agent koji služi kao posrednik između upravljača i podagenata (*subagent*) koji se brinu o održavanju MIB baze i upravljanim varijablama. U novoj hijerarhiji, SNMP modul u sklopu ikev2 preuzima ulogu podagenta i obrađuje zahtjeve koje mu glavni agent proslijedi.



Slika 6.1 AgentX arhitektura sustava za upravljanje mrežom

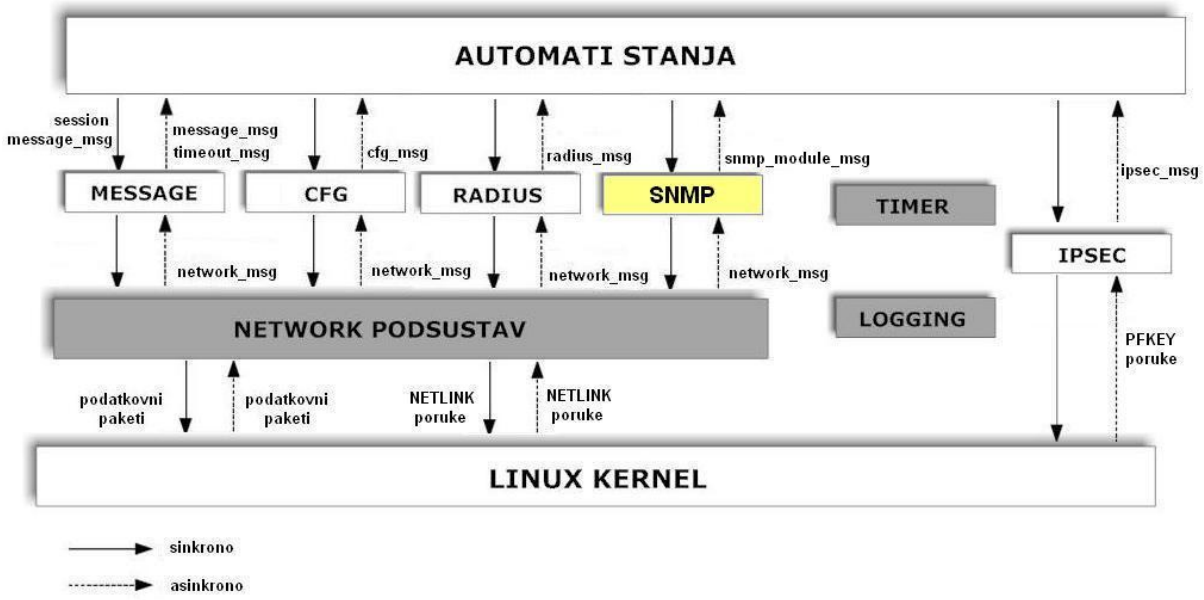
Zašto uopće ovo naizgled nepotrebno kompliciranje? Dva su razloga. Prvi je mogućnost više podagenata na istom stroju, odnosno više agenata u klasičnom smislu. Bez AgentX-a to je bilo nemoguće jer se samo jedan agent mogao registrirati i osluškivati na pristupu 161. U novoj arhitekturi imamo glavnog agenta koji je na pristupu 161 i prima SNMP poruke s mreže te proizvoljan broj podagenata koji s njim komuniciraju putem AgentX protokola. Drugi razlog je da je u standardnom sustavu za upravljanju mrežom nemoguće dodavati i uklanjati objekte MIB-a dok je agent aktivan. Sličan protokol AgentX-u je SMUX (*SNMP Multiplexing Protocol*) ali on je pojavom AgentX-a zastario i ne preporučuje se njegovo korištenje.

Što se samih svojstava arhitekture AgentX-a tiče, već spomenuti glavni agent je zadužen za održavanje tablice sa zapisima o tome koji je podagent odgovoran za koji dio MIB područja. Kada glavni agent primi poruku putem SNMP-a, on pronalazi podagenta odgovornog za pojedino MIB područje i proslijedi mu odgovarajući AgentX zahtjev. Glavni agent ne sadrži gotovo nikakve upravljane informacije s eventualnom iznimkom informacija o trenutno spojenim agentima i njihovim MIB područjima.

Podagenti su zaduženi za pružanje pristupa upravljanim informacijama. Kada je podagent pokrenut, on kontaktira glavnog agenta i registrira MIB područje za koje čuva upravljane informacije. Podagenti u pravilu nemaju doticaja jedan s drugim pa je glavni agent zadužen za eventualne konflikte između njih. Glavni agent mora nastojati izbjeći konflikte i preklapanja MIB područja, a ukoliko do njih dođe mora razriješiti konflikte na način opisan u AgentX protokolu.

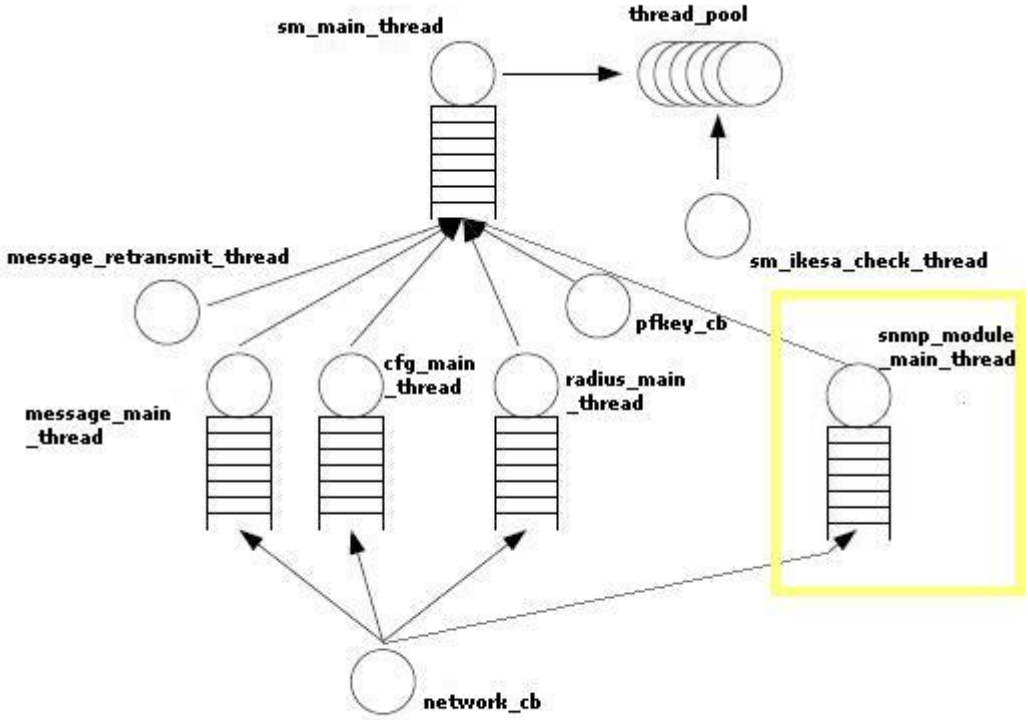
6.1.2 Ugradnja u postojeću ikev2 arhitekturu

SNMP modul napravljen je slično ostalim modulima koji su opisani u poglavlju 5.3. Slika 6.2 prikazuje novi statički pogled na arhitekturu.



Slika 6.2 Statički pogled na ikev2 sa SNMP modulom

I dinamički pogled je sada promijenjen budući da je uvedena nova dretva `snmp_module_main_thread` s pripadajućim redom.



Slika 6.3 Dinamički pogled na ikev2 sa SNMP modulom

Struktura poruke koju `snmp_module_main_thread` šalje prema `sm_main_thread` je sljedeća:

```
struct snmp_module_msg {
      guint8 msg_type;
      guint8 message;
      GMutex *mutex;
};
```

Slika 6.4 Struktura SNMP poruke

Ikev2 koristi biblioteku Glib2[16] za osiguravanje prenosivosti koda pa su i korišteni tipovi podataka uglavnom iz te biblioteke. Varijable `msg_type` i `message` su nepredznačeni 8-bitni cijeli brojevi dok je pokazivač `mutex` tipa GMutex koji predstavlja strukturu korištenu za međusobno isključivanje. `Msg_type` označava tip poruke (u ovom slučaju to je broj 9, primjerice za RADIUS poruku to je broj 5 itd.),

`message` predstavlja podatak koji se prenosi a pokazivač `mutex` se koristi za sinkronizaciju s automatom stanja.

Posao SNMP modula započinje inicijaliziranjem svih potrebnih parametara:

```
int snmp_module_init(struct ikev2_data* data, GAsyncQueue *upper_queue) {

      int retval = -1;
      char *app = "miljen_ikev2";

      snmp_module_data = g_malloc0(sizeof(struct snmp_module_data));

      snmp_module_data->network_queue = g_async_queue_new();

      g_async_queue_ref(upper_queue);

      snmp_module_data->upper_queue = upper_queue;

      snmp_module_config();

      init_agent(app);

      init_ikemibs(data);

      init_snmp(app);

      snmp_module_data->main_thread = g_thread_create(snmp_module_main_thread,
                                                      snmp_module_data, TRUE, NULL);
      snmp_shutdown(NULL);

      retval = 0;

out:
      return retval;
}
```

Slika 6.5 Inicijalizacija SNMP modula

Većina poziva koji se pojavljuju u ovoj funkciji odnose se na pozive funkcija iz Net-SNMP implementacije (`init_*` pozivi). Bitno je istaknuti registriranje novog reda za primanje poruka od NETWORK podsustava kao i registriranje reda višeg sloja (*upper queue*) kojem se prosljeđuje poruka nakon obrade.

Dinamički pogled upotpunjen je novom dretvom, `snmp_module_main_thread`. Ona u beskonačnoj petlji očekuje zahtjeve koje je glavni agent primio i proslijedio podagentu, odnosno SNMP modulu. Nakon primitka zahtjeva, stvara se nova poruka čija struktura je prikazana na slici 6.4 i koristi se mehanizam međusobnog isključivanja za sinkronizaciju s automatom stanja. Ta poruka se zatim šalje automatu stanja i čeka se na ponovnu mogućnost zaključavanja koja će nastupiti tek kad automat stanja obradi poruku i otključa varijablu koja služi za sinkronizaciju. Programski kod:

```
while (1) {
      LOG_DEBUG("Waiting for SNMP messages");
      snmp_select_info(&numfds, &fdset, tvp, &fakeblock);
      count = select(numfds, &fdset, 0, 0, NULL);
      snmp_mmsg = snmp_module_msg_new(2);
      g_mutex_lock(snmp_mmsg->mutex);
      LOG_DEBUG("Packet ready for sending to the sm_main_thread");
      g_async_queue_push (snmp_module_data->upper_queue, snmp_mmsg);
      LOG_DEBUG("Packet sent to sm_main_thread");
      g_mutex_lock(snmp_mmsg->mutex);
      LOG_DEBUG("Synchronization with sm_main_thread done");
}
```

Slika 6.6 Glavni dio SNMP dretve

Automat stanja nakon primitka `snmp_module` poruke obavlja njeno procesiranje i sinkronizira se sa SNMP modulom pomoću mehanizma za međusobno isključivanje. Procesiranje je prikazano na slici 6.7.

```
/**
 * This is a main IKEv2 thread. All the events are routed to this function
 * which demultiplexes it to appropriate handler. Only one instance of
 * this function is executed ever...
 *
 * TODO: Retransmission handling in this function might slow it down because
 *          of locking. Maybe it should be handled to thread pool...
 */
gpointer sm_main_thread(gpointer data) { ...

#ifdef SNMP_MODULE
           case SNMP_MODULE_MSG:
                LOG_DEBUG("SNMP message arrived..");
                agent_check_and_process(0);
                g_mutex_unlock(msg->snmp_mmsg.mutex);
                LOG_DEBUG("Packet processed..");
                break;
#endif /* SNMP_MODULE */


           } /* switch (msg->msg_type) */
```

Slika 6.7 Procesiranje SNMP poruke u automatu stanja

Treba napomenuti da je cijeli kod koji se odnosi na SNMP podršku ugniježđen između pretprocesorskih direktiva `#ifdef` i `#endif`. To je zato da se omogući pokretanje *daemona* i bez SNMP podrške ukoliko ona nije potrebna. Postavljanjem odgovarajućeg parametra prilikom prevođenja i konfiguriranja uključuje se SNMP i dalje normalno koristi.

6.2 Upravljane varijable

Nakon što je objašnjeno kako je ostvaren i gdje je smješten SNMP podsustav u sklopu cijelog IKEv2 okruženja, potrebno je izdvojiti i popisati one varijable koje bi se mogle koristiti za potrebe nadziranja. Drugim riječima, slijedi lista upravljanih varijabli. One se mogu podijeliti u tri kategorije:

- varijable specifične za samu ikev2 implementaciju
- varijable vezane za pojedinu sjednicu
- varijable vezane za pojedini CHILD SA

Uspostava sigurnosnog udruženja počinje uspostavom sjednice u kojoj se definiraju bitni parametri za komunikaciju kao što su IP adrese, SPI vrijednosti itd. Nakon toga, uspostavljaju se pojedini CHILD SA-ovi koji imaju svoje specifične varijable. Iz tog skupa varijabli odabrane su one koje bi mogle biti bitne upravljaču u sustavu za upravljanje mrežom.

Tablica 6.1 Upravljane varijable vezane za IKEv2

Naziv varijable	Opis varijable
`dos_treshold`	Broj poluotvorenih konekcija nakon kojih se pretpostavlja pojava DoS napada i počinju se slati COOKIE obavijesti
`sm_threads`	Broj radnih dretvi u *thread poolu* automata stanja, određuje se na temelju očekivanog opterećenja
`random_device`	Znakovni niz za inicijalizaciju CRYPTO podsustava
`pidfile`	Naziv i putanja do *pid* datoteke
`mode`	Način u kojem IKEv2 *daemon* funkcionira; može se ponašati kao *initiator, responder* ili kao kombinacija ovih uloga
`ikesa_max`	Najveći dopušteni broj IKE SA-ova dopušten s pojedine IP adrese, ovo je jedan od načina za obranu od DoS napada
`ikesa_max_halfopened`	Najveći dopušteni broj poluotvorenih IKE SA-ova s pojedine IP adrese
`psk_file`	Naziv datoteke s dijeljenom tajnom
`kernel_spd`	Parametar koji određuje kako se IKEv2 *daemon* ponaša s obzirom na SPD
`options`	Opcije specificirane u generalnom odjeljku konfiguracijske datoteke
`session_counter`	Brojač sjednica

Tablica 6.2 Upravljane varijable vezane za pojedinu sjednicu

Naziv varijable	Opis varijable
`state`	Stanje sjednice
`ike_type`	Cijeli broj koji određuje da li ova sjednica pripada *initatoru* ili *responderu*
`create_time`	Vrijeme stvaranja ovog IKE SA
`auth_limit_time`	Maksimalno vrijeme trajanja IKE SA ove sjednice
`rekey_time`	Vrijeme kad je ovaj IKE SA zadnji obavio *rekeying*, odnosno dobavio nove ključeve
`allocations`	Broj alokacija koje ima ovaj IKE SA
`last_seen`	Zadnje vrijeme kada smo nešto primili od *peera*
`msg_id`	ID poslan za *rekeying* IKE SA
`natt`	Zastavica koja označava treba li UDP enkapsulacija biti uključena
`csa_number`	Broj CHILD SA-ova

Tablica 6.3 Upravljane varijable vezane za CHILD SA

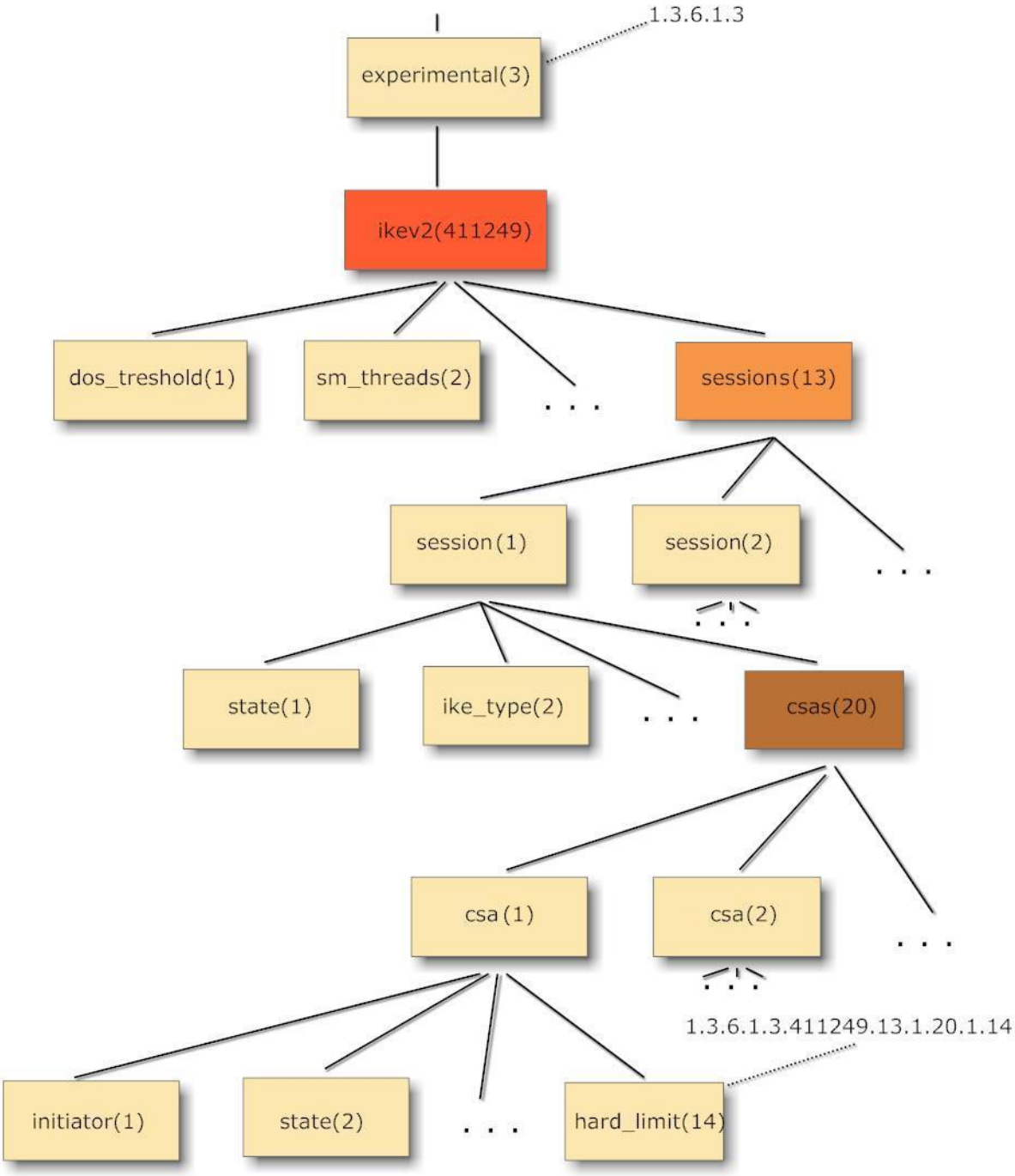
Naziv varijable	Opis varijable
`initiator`	Zastavica koja govori da li je ovaj CSA *initiator*
`state`	Stanje u kojem se CHILD SA nalazi
`msg_id`	ID koji mora sadržavati odgovor *peera*
`spi`	SPI vrijednost za ovaj CHILD SA
`peer_spi`	SPI vrijednost za *peer*
`spi_size`	Veličina SPI-a, 4 ukoliko se provodi stvaranje/*rekeying* CHILD SA-a odnosno 8 ako se provodi *rekeying* IKE SA-a
`ipsec_mode`	Način rada koji se koristi (*tunnel* ili *transport*)
`sk_el`	Dijeljeni ključ *initiatora* za enkripciju CHILD SA
`sk_er`	Dijeljeni ključ *respondera* za enkripciju CHILD SA
`sk_al`	Dijeljeni ključ *initiatora* za autentifikaciju CHILD SA
`sk_ar`	Dijeljeni ključ *respondera* za autentifikaciju CHILD SA
`remove_time`	Vrijeme kada je CHILD SA prešao u stanje REMOVE
`seq`	Ukoliko se pošalje zahtjev jezgri operacijskog sustava, ovo polje čuva broj zahtjeva za koji se očekuje odgovor
`hard_limit`	Zastavica koja označava da li je nastupio *hard limit*

6.3 Smještaj unutar ASN.1 stabla

Upravljane varijable bitne za protokol za razmjenu ključeva definirane su u prethodnom potpoglavlju. Na ovom mjestu biti će prikazan njihov smještaj unutar ASN.1 stabla. Cijelo podstablo s varijablama smješteno je ispod čvora *experimental* ASN.1 stabla, tj. svi identifikatori objekata za ikev2 varijable počinju sa .1.3.6.1.3 (izgled ASN.1 stabla prikazan je na slici 2.3 te u Dodatku B).

U svrhu smještaja ikev2 upravljanih varijabli definiran je novi čvor, 411249, čime je osigurana jedinstvenost identifikatora objekata. Ispod njega nalaze se varijable vezane uz IKEv2 *daemon*, primjerice varijabla *random_device* (treća po redu) ima OID .1.3.6.1.3.411249.3. Nakon tih varijabli slijedi čvor 13 ispod kojeg se nalaze varijable vezane za pojedinu sjednicu. Primjerice, prva sjednica ima OID .1.3.6.1.3.411249.13.1. Zadnji broj ovog OID-a ovisi o rednom broju sjednice – radi se o 32-bitnom brojaču čime je osiguran dovoljan broj vrijednosti za sve moguće sjednice (nerealno je očekivati više od 2^{32} sjednica). U sklopu svake sjednice

varijable su raspoređene na sličan način kao kod varijabli vezane za sam *daemon*; prvo idu varijable specifične za sjednicu a zatim slijedi čvor 20 ispod kojeg počinju CSA-ovi. Prvi CSA ima dakle OID .1.3.6.1.3.411249.13.broj_sjednice.20.1. Slika 6.8 puno jasnije ilustrira smještaj upravljanih varijabli unutar ASN.1 stabla. Na njoj je bitno uočiti moguće postojanje većeg broja sjednica i većeg broja CHILD SA-ova, tj. ovdje se radi o tabličnim objektima za razliku od svih ostalih upravljanih objekata ikev2 implementacije koji su predstavljeni skalarima.



Slika 6.8 Smještaj ikev2 varijabli u ASN.1 stablu

6.4 Registriranje varijabli

Registriranje upravljanih varijabli pomoću programskog sučelja koje pruža Net-SNMP implementacija moguće je na dva načina:

- registriranje pomoću instance
- registriranje pomoću *handlera*

Prvi način je jednostavniji i provodi se na način da se u MIB ubaci referenca na upravljanu varijablu i upravljaču se preko reference lako dostavi vrijednost te varijable prilikom zahtjeva za njom. Drugi način je složeniji. Glavna ideja kod tog načina registriranja je da se nakon primitka zahtjeva upravljača za nekom varijablom poziva odgovarajuća *callback* funkcija koja pritom obavlja sve potrebne radnje i dostavlja upravljaču (odnosno, najprije glavnom agentu, s obzirom na korištenje AgentX *frameworka*) vrijednost tražene varijable.

Registriranje prve grupe varijabli (varijable vezane uz ikev2) obavlja se pomoću instance. Sam poziv zahtijeva specificiranje naziva varijable, njenog OID-a te naravno, pokazivača na samu varijablu.

Registriranje sjednice i njenih varijabli koristi složeniji način, registriranje pomoću *handlera*. Logično pitanje je – zašto? Dva su glavna razloga: takav način registriranja je općenitiji te potreba za zaključavanjem i otključavanjem pojedine sjednice. Ukoliko se koristi prvi način, mora se odmah pozvati funkcija vezana za tip varijable (primjerice, registracija cjelobrojne varijable zahtijeva poziv netsnmp_register_**int**_instance(..) funkcije itd.) što ovdje nije slučaj pa se mogu u samoj *handler* funkciji demultipleksirati zahtjevi ovisno o tipu varijable. Drugi razlog je važniji i vezan je za cijelo ikev2 okruženje. Instancama sjednice nemoguće je pristupati bez prethodnog zaključavanja budući da je moguće da u tom trenutku neki drugi podsustav koristi dijelove sjednice pa bi došlo do kolizije. Sada je jasno da vraćanje varijable upravljaču može uzrokovati čitanje varijable koja se upravo mijenja, i naposljetku vraćanje krive vrijednosti. Zato je prije dohvata varijable potrebno osigurati da nitko drugi ne mijenja njenu vrijednost, i zbog vraćanja ispravne vrijednosti ali i zbog sprečavanja mogućih rušenja sustava uzrokovanih segmentacijskom greškom (*segmentation fault*). Sjednice se zaključavaju mehanizmima koje pruža Glib2 biblioteka. Sam proces registracije sjednice sastoji se od uvećavanja brojača sjednica (koji je, podsjetnika radi, također upravljana varijabla) i registriranja *handlera* za svaku od varijabli sjednice kojima se upravlja. Općenitost je u tome da se prilikom registracije svih tih varijabli daje pokazivač na isti *handler* – funkciju `sm_sess_handle` koja onda na temelju identifikatora objekta određuje o kojoj se varijabli radi i koje predradnje mora učiniti. Potpuno isti koncept koristi se prilikom registriranja CHILD SA-ova i pripadnih varijabli.

Isječak koda za registraciju *handlera* za cjelobrojne varijable sjednice nalazi se na slici 6.9:

```
for(i=1; i<=SESSION_MIB_INT_VARIABLES; i++) {

        session_oid[SESSION_TREE_DEPTH - 1] = i;
        LOG_DEBUG("Registriran %d", i);

        my_test =
                netsnmp_create_handler_registration("session handler",
                                                    sm_sess_handle,
                                                    session_oid,
                                                    OID_LENGTH(session_oid),
                                                    HANDLER_CAN_RONLY);

        netsnmp_register_instance(my_test);
}
```

Slika 6.9 Registracija handlera za cjelobrojne varijable sjednice

6.5 Obrada varijabli

U prethodnom potpoglavlju opisani su načini registracije varijabli i spomenuto je da postoje dva glavna načina. Kod prvog načina (registriranja pomoću instance) nikakva dodatna obrada se ne vrši već se varijabla direktno upakira u odgovarajući PDU i vraća upravljaču preko glavnog agenta. Drugi način, registriranje pomoću *handlera* zahtijeva posebnu funkciju - *handler*, koja će vršiti svu potrebnu obradu i nakon toga vratiti vrijednost varijable. Taj način koristi se prilikom registriranja varijabli vezanih za sjednicu i CHILD SA. Algoritam za obradu varijabli vezanih za sjednicu je sljedeći (slično je za CHILD SA):

1. Iz zahtjeva se izvuče podatak o identifikatoru objekta

2. Poziva se funkcija za dohvat vrijednosti varijable na temelju identifikatora

3. Funkcija za dohvat prvo traži odgovarajuću sjednicu prolaskom kroz listu svih sjednica i usporedbom s identifikatorom iz zahtjeva. Nakon nalaženja sjednice, zaključava ju i iz nje vraća vrijednost tražene varijable. Postupak završava otključavanjem sjednice

4. Ovisno o tipu dohvaćene varijable, ispunjava se odgovarajući PDU pozivom funkcija iz Net-SNMP implementacije i time generira odgovor upravljaču

Programski kod funkcije za nalaženje sjednice na temelju OID-a nalazi se na slici 6.10, a funkcije za dohvat vrijednosti na slici 6.11:

```
/**
 * Search sessions by a given object ID (OID)
 *
 * @param session_object_id  OID to search for
 *
 * Session is searched by OID with a reader lock taken. If
 * session is found, then the lock is taken on session before
 * releasing global reader lock.
 */
struct session *sm_sessions_search_by_oid(int session_object_id)
{
	GSList *slist;
	struct session *session;

	LOG_FUNC_START(1);

	sm_sessions_reader_lock();

	for (slist = ikev2_data->sessions; slist; slist = slist->next) {
		session = slist->data;

		if (session_get_oid(session) != session_object_id)
			continue;
		else {
			session_lock(session);
			goto out;
		}
	}

	session =  NULL;

out:
	sm_sessions_reader_unlock();

	LOG_FUNC_END(1);

	return session;
}
```

Slika 6.10 Nalaženje sjednice preko OID-a

```
void sm_sess_retrieve_value(void **value, int session_index, int variable_index)
{

      struct session* session = NULL;

      LOG_FUNC_START(2);

      session = sm_sessions_search_by_oid(session_index);

      if(session == NULL) {
            LOG_ERROR("Session not found by oid");
            return;
      }
      if(variable_index == SESSION_OID_CSA_NUMBER)
            sm_sess_count_csa(session, value);

      session_get_mib_variable(session, variable_index, value);

      session_unlock(session);

      LOG_FUNC_END(2);
}
```

Slika 6.11 Dohvat vrijednosti upravljane varijable na temelju OID-a

6.6 Deregistriranje varijabli

Nakon što se pojedina sjednica ili CHILD SA uništi, potrebno je osim oslobađanja memorije ukloniti i prethodno registrirane varijable kako ih upravljač zabunom ne bi dohvatio misleći da su još aktivne. Taj posao obavlja funkcija čiji je prototip:

```
unregister_mib(oid *, size_t);
```

Funkcija prima polje u kojem je spremljen OID i veličinu tog polja. Nažalost, ne postoji mogućnost deregistracije cijelog podstabla odjednom, već je potrebno ukloniti jednu po jednu varijablu što je riješeno programskom petljom.

6.7 Dodavanje nove varijable

Jedan od glavnih zahtjeva prilikom ugradnje SNMP podrške bio je da se omogući što lakše dodavanje nove upravljane varijable. Ukoliko se npr. želi dodati nova cjelobrojna varijabla vezana za pojedinu sjednicu, procedura je sljedeća:

1. U datoteci "oids.h" potrebno je pronaći pretprocesorsku direktivu `#define SESSION_MIB_INT_VARIABLES 10` i inkrementirati vrijednost koja piše na kraju retka (u ovom slučaju, 10). U slučaju dodavanja znakovnog niza, traži se makro `SESSION_MIB_STR_VARIABLES` i njegova vrijednost se poveća za jedan. Ovi makroi inače označavaju koliko se trenutno varijabli sjednice registrira za svaki tip upravljane varijable.

2. U istoj datoteci potrebno je pronaći popis makroa za postojeće varijable sjednice koje se registriraju i na kraj tog popisa dodati naziv nove varijable. U trenutku pisanja ovog rada, taj popis završava sa direktivom `#define SESSION_OID_CSA_NUMBER 10` pa se iza nje doda direktiva `#define SESSION_OID_NEW_VARIABLE 11` ako novu varijablu nazovemo primjerice `new_variable`. Analogno za znakovne nizove.

3. U datoteci "session.c" potrebno je potražiti funkciju s prototipom `session_get_mib_variable(struct session* session, int variable_index, void ** value);`

   U toj funkciji u postojeći switch-case konstrukt potrebno je dodati još jednu granu za novu varijablu (slika 6.12) bez obzira o kojem tipu varijable se radi:

```
switch(variable_index) {

        ...

        case SESSION_OID_MSG_ID:
              *value = (int*) & (session->msgid);
              break;
        case SESSION_OID_NATT:
              *value = (int*) & (session->natt);
              break;
        case SESSION_OID_NEW_VARIABLE:
              *value = (int*) & (session->new_variable);
              break;
        default:
              LOG_ERROR("There is no such MIB variable!");
              break;
}
```

Slika 6.12 Dodavanje nove upravljane varijable u switch-case konstrukt

Procedura je doista jednostavna te omogućuje i osobama koje nisu upoznate sa SNMP protokolom da na bezbolan način registriraju nove varijable. Koraci su opisani detaljno uz stalno povlačenje paralele sa registriranjem varijable koja je tipa znakovnog niza. Na potpuno isti način dodaju se varijable vezane za sam ikev2 kao i za pojedini CHILD SA, osim što se u trećem koraku mora potražiti druga datoteka ("sm.c", odnosno "sad.c").

Testiranje nove varijable radi se tako da se pokrene ikev2 implementacija uz uključenu SNMP podršku (opcija `--enable-snmp` prilikom pokretanja `configure` skripte), dodijeli neka vrijednost varijabli i onda ispita naredbom `snmpget`. Ukoliko sve prođe bez greške, na ekranu bi trebala pisati dodijeljena vrijednost.

7. Zaključak

Svaki ozbiljniji suvremeni mrežni protokol definira varijable koje se koriste za potrebe nadziranja na mreži. Tako se rodila i ideja o uvođenju iste funkcionalnosti u implementaciju protokola za razmjenu ključeva, ikev2. Sukladno složenosti same implementacije, SNMP podsustav je uveden pazeći na specifičnost komunikacije s automatima stanja kao i s mrežnim podsustavom.

Sam izbor upravljanih varijabli napravljen je vodeći računa o tome koje varijable bi mogle biti zanimljive mrežnom upravljaču. Ipak, kako se ikev2 razvija i proširuje moguće je da će pojaviti i nove varijable koje treba uključiti u bazu upravljanih informacija pa je podrška razvijena u smjeru što lakšeg proširenja tog skupa varijabli. Odabrani identifikatori objekata osiguravaju prije svega jedinstvenost, a odmah potom i proširivost.

Iako u svom nazivu ima pridjev "jednostavni“, SNMP nipošto nije jednostavan protokol i za njegovo bolje razumijevanje potrebno je dosta vremena. Ugradnja u ikev2 bila je olakšana postojanjem Net-SNMP implementacije i programskih sučelja koje ona nudi, ali i tu je bilo problema. Naime, višedretvenost koja je važno svojstvo ikev2 je slabo podržana u Net-SNMP paketu a i starija verzija tog paketa je imala problema s AgentX protokolom pa je na kraju korištena trenutno najnovija verzija (5.4.1).

Ugradnjom SNMP podrške u ikev2 napravljen je dodatni iskorak kojim je osiguran još veći broj mogućih korisnika ove implementacije protokola za razmjenu ključeva, a time i IPsec arhitekture koja se sve više nameće kao jedno od glavnih rješenja sigurnosnih problema Interneta.

8. Literatura

[1] C. Kaufman, Ed.: *Internet Key Exchange Key (IKEv2) Protocol*, URL: http://www.ietf.org/rfc/rfc4306.txt, 2005.

[2] S.Kent, K.Seo: *Security Architecture for Internet Protocol*, RFC 4301, December 2005.

[3] J.Case, M.Fedor, M.Schoffstall, J.Davin: *A Simple Network Management Protocol*, RFC 1157, May 1990.

[4] A.Kukec: *Introduction to IKEv2 and IPsec,*, URL: http://os2.zemris.fer.hr/ns/2006_kukec/, March 2006.

[5] M.Rose, K.McCloghrie: *Structure and Identification of Management Information for TCP/IP based Internets*, RFC 1155, May 1990.

[6] *Net-SNMP implementation*, URL:http://www.net-snmp.org,

[7] ITU-T: *Information Technology - Abstract Syntax Notation One (ASN.1) : Specification of basic notation*, X.680, July 2002.

[8] S.Kent: *IP Encapsulating Security Payload (ESP)*, RFC 4303, December 2005.

[9] S. Kent: *IP Authentication Header*, RFC 4302, December 2005.

[10] ITU-T: *Information Technology – ASN.1 encoding rules*, X.690, July 2002.

[11] K. McCloghrie, D. Perkins, J. Schoenwaelder: *Structure of Management Information Version 2 (SMIv2)*, RFC 2578, April 1999.

[12] S.Groš, V.Glavinić: *Architecture of an IKEv2 protocol implementation*, May 2007.

[13] A.Bažant: *Protokoli upravljanja mrežom*, December 2003.

[14] D.Bruey: *SNMP: Simple? Network Management Protocol*, 2005.

[15] M.Daniels, B.Wijnen, M.Ellison, D.Francisco: *Agent Extensibility (AgentX) Protocol Version 1*, RFC 2741, January 2000.

[16] *Glib2 API documentation*, URL: http://developer.gnome.org/doc/API/2.0/glib/index.html, September 2008.

[17] J.Davin, J.Case, M.Fedor, M.Schoffstall: *A Simple Gateway Monitoring Protocol*, RFC 1028, November 1987.

Dodatak A – MIB datoteka

```
IKEV2-MIB DEFINITIONS ::= BEGIN

IMPORTS
       OBJECT-TYPE,
       Integer32,
       MODULE-IDENTITY,
       experimental        FROM SNMPv2-SMI;

ikev2MIB MODULE-IDENTITY
       LAST-UPDATED "200809010000Z"     -- 01 September 2008, midnight
       ORGANIZATION "FER, ZEMRIS"
       CONTACT-INFO "email: ikev2-devel@zemris.fer.hr"
       DESCRIPTION "Mib for IKEv2 implementation."

       ::= {experimental 411249 }

ikev2dosTreshold OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Number of half opened connections after which we assume
        DOS attack and start to send COOKIE notifications."

    ::= { ikev2MIB 1 }

ikev2smThreads OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Number of worker threads in sm pool. This parameter has
        to be determined based on expected load."

    ::= { ikev2MIB 2 }

ikev2randomDevice OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Random device to initialize crypto system."

    ::= { ikev2MIB 3 }

ikev2pidFile OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Name and path of a pid file."

    ::= { ikev2MIB 4 }

ikev2mode OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Mode in which IKEv2 daemon functions. It can strictly
        behave as initiator, responder, or it can take both
```

```
        roles."

    ::= { ikev2MIB 5 }

ikev2ikesaMax OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Maximum number of IKE SAs allowed from a single IP address.
        This is also form of a DOS attack defence, but NATs create
        problems here."

    ::= { ikev2MIB 6 }

ikev2ikesaMaxHalfopened OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Maximum number of half opened IKE SAs from the single IP address."

    ::= { ikev2MIB 7 }

ikev2pskFile OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "File with a shared keys."

    ::= { ikev2MIB 8 }

ikev2kernelSPD OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Parameter that determines how IKEv2 daemon behaves with respect
        to in-kernel SPD."

    ::= { ikev2MIB 9 }

ikev2options OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Options specified in the general section."

    ::= { ikev2MIB 10 }

ikev2sessionCounter OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "Counter of active sessions."

    ::= { ikev2MIB 11 }

ikev2sessions     OBJECT-TYPE
    SYNTAX       SEQUENCE OF Session
    MAX-ACCESS   not-accessible
    STATUS       current
```

```
    DESCRIPTION "IKEv2 sessions.."

    ::= { ikev2MIB 13 }


session     OBJECT-TYPE
    SYNTAX       Session
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "Session that is established between peers."

--  INDEX       { sessionID }

    ::= { ikev2sessions 1 }

Session ::= SEQUENCE
    {
       sessionState              Integer32,
       sessionIkeType            Integer32,
       sessionCreateTime         Integer32,
       sessionAuthLimitTime      Integer32,
       sessionRekeyTime          Integer32,
       sessionAllocations        Integer32,
       sessionLastSeen           Integer32,
       sessionMsgID              Integer32,
       sessionNatt               Integer32,
       sessionCsaNumber          Integer32,
       sessionCsas               CSA
    }

sessionState  OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "State of the session."
    ::= { session 1 }

sessionIkeType   OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION  "Ike_type defines if the particular structure belongs to initiator
                (IKEV2_INITIATOR) or to responder (IKEV2_RESPONDER)."

    ::= { session 2 }

sessionCreateTime OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION  "Timestamp when this IKE SA has been created."

    ::= { session 3 }

sessionAuthLimitTime OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION  "Maximum time that IKE SA of this session can be extended
                (in seconds relative to 'create_time')."

    ::= { session 4 }

sessionRekeyTime  OBJECT-TYPE
    SYNTAX        Integer32
```

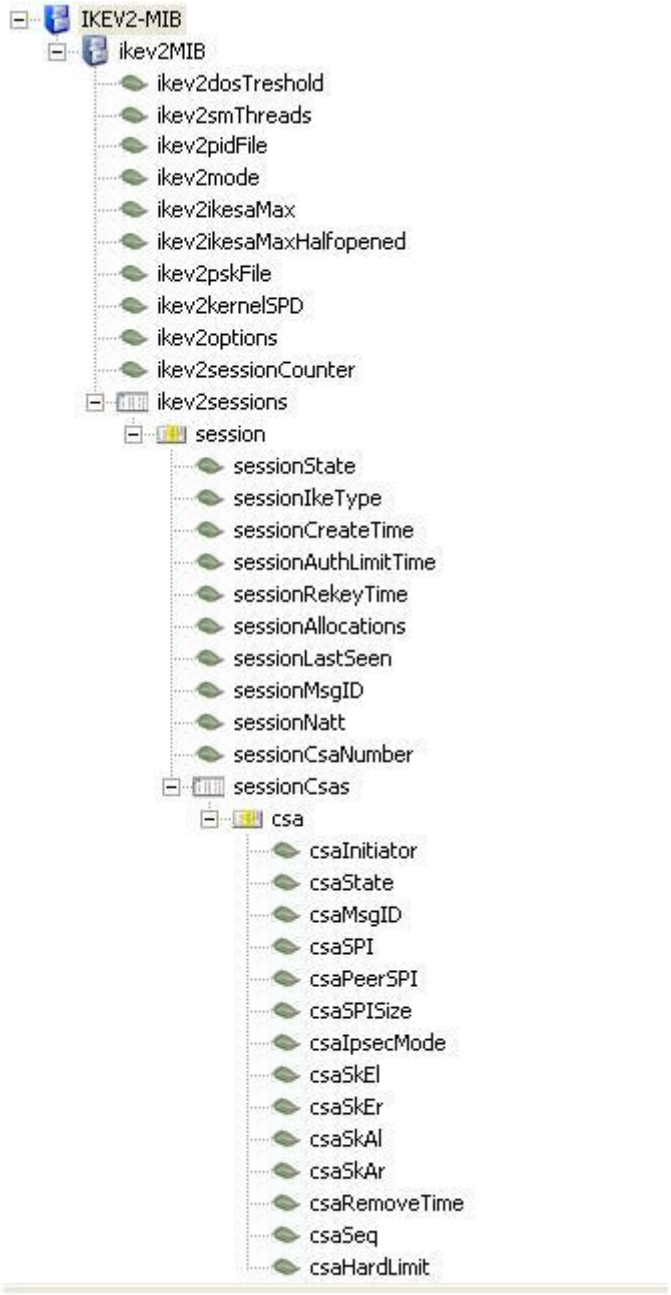
```
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION  "Timestamp when this IKE SA has last been rekeyed."

    ::= { session 5 }

sessionAllocations OBJECT-TYPE
    SYNTAX         Integer32
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION   "Number of allocations this IKE SA had."

    ::= { session 6 }

sessionLastSeen   OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION  "Last time (in Unix time) we received something from peer."

    ::= { session 7 }

sessionMsgID     OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "ID with request sent to rekey IKE SA."

    ::= { session 8 }

sessionNatt      OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "If UDP encapsulation should be enabled."

    ::= { session 9 }

sessionCsaNumber OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "NUmber of active CHILD SAs."

    ::= { session 10 }

sessionCsas     OBJECT-TYPE
    SYNTAX       SEQUENCE OF CSA
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "CHILD SAs"

    ::= { session 20 }


csa     OBJECT-TYPE
       SYNTAX       CSA
       MAX-ACCESS   not-accessible
       STATUS       current
       DESCRIPTION "Child SA that is established between peers."

--  INDEX       { csaID }

    ::= { sessionCsas 1 }
```

```
CSA ::= SEQUENCE
    {
       csaInitiator        Integer32,
       csaState            Integer32,
       csaMsgID            Integer32,
       csaSPI              Integer32,
       csaPeerSPI          Integer32,
       csaSPISize          Integer32,
       csaIpsecMode        Integer32,
       csaSkEl             OCTET STRING,
       csaSkEr             OCTET STRING,
       csaSkAl             OCTET STRING,
       csaSkAr             OCTET STRING,
       csaRemoveTime       Integer32,
       csaSeq              Integer32,
       csaHardLimit        Integer32
    }

csaInitiator  OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "TRUE if this CSA is initiator. This variable changes the way
                 sk_e? and sk_a? fields are treated."
    ::= { csa 1 }

csaState      OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "State in which this CHILD SA is."
    ::= { csa 2 }

csaMsgID      OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "If we sent request message to a peer, then the peer has to respond
                 with a message that has message_id as the one placed in the
                 following field..."
    ::= { csa 3 }

csaSPI        OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "SPI value choosen by local node."
    ::= { csa 4 }

csaPeerSPI    OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "SPI value choosen by peer."
    ::= { csa 5 }

csaSPISize    OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "This depends, if we are creating/rekeying
                 CHILD SA in which case it's 4, or rekeying
                 IKE SA, then it's 8."
    ::= { csa 6 }
```

```
csaIpsecMode  OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "SA mode that is used, either tunnel (IPSEC_MODE_TUNNEL),
                or transport (IPSEC_MODE_TRANSPORT). If this field is
                zero, than mode is inherited from the associated policy."
    ::= { csa 7 }

csaSkEl             OBJECT-TYPE
    SYNTAX       OCTET STRING
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "Shared key for CHILD SA
                Unlike the specification that calls them i for initiator
                and r for remote, we call them l for local and r for
                remote.."
    ::= { csa 8 }

csaSkEr             OBJECT-TYPE
    SYNTAX       OCTET STRING
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "Shared key for CHILD SA
                Unlike the specification that calls them i for initiator
                and r for remote, we call them l for local and r for
                remote.."
    ::= { csa 9 }

csaSkAl             OBJECT-TYPE
    SYNTAX       OCTET STRING
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "Shared key for CHILD SA
                Unlike the specification that calls them i for initiator
                and r for remote, we call them l for local and r for
                remote.."
    ::= { csa 10 }

csaSkAr             OBJECT-TYPE
    SYNTAX       OCTET STRING
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "Shared key for CHILD SA
                Unlike the specification that calls them i for initiator
                and r for remote, we call them l for local and r for
                remote.."
    ::= { csa 11 }

csaRemoveTime OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "When this CHILD SA has been transfered to REMOVE state."
    ::= { csa 12 }

csaSeq        OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "If we sent some request to a kernel then this field is different
                from zero and holds a number of a request to which we expect
                response."
    ::= { csa 13 }
```

```
csaHardLimit  OBJECT-TYPE
    SYNTAX       Integer32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "Have we reached hard limit of SA?"
    ::= { csa 14 }

END
```

Dodatak B - ASN.1 stablo s IKEv2 varijablama



Slika B.1 ASN.1 stablo s ikev2 varijablama