UNIVERZITET SINGIDUNUM

Departman za poslediplomske studije

Savremene informacione tehnologije

Upotreba *Oracle Wallet* rešenja za sigurno pozivanje Web servisa iz *PL/SQL-a*

Master rad

Mentor: Prof. dr. Mladen Veinović

Student: Zurnađi Anđela

Broj indeksa: 2014/410150

Beograd, 2017.

Upotreba *Oracle Wallet* rešenja za sigurno pozivanje Web servisa iz *PL/SQL-a*

Sažetak: Integracija postojećih informacionih sistema je do pojavljivanja *Web* servis (engl.*web service*) standarda predstavljala veliki izazov za brojne kompanije. Usled potrebe razmene informacija, njihovog masovnog korišćenja i povezivanja distribuiranih komponenti napisanih u različitim tehnologijama (*java, .net, VB, COBOL, C*...), uspostavljen je standard koji ne zavisi od korišćenog programskog jezika, niti platforme na kojoj je realizovan i koristi standardni *http(s)* protokol putem koga se vrši komunikacija.

U ovom radu je opisan jedan od mogućih načina pozivanja *Web* servisa iz *Oracle PL/SQL* procedura. Oslanjajući se na primer upotrebe u realnom okruženju, od *Oracle* verzije *9i Release 2,* paket *UTL_HTTP* omogućava pozivanje servisa preko *https* protokola (engl. *Hypertext Transfer Protocol Secure*) konfiguracijom *Oracle Wallet* rešenja na serveru, kako bi se omogućio pouzdan prenos podataka uključujući sigurnu identifikaciju servera.

Ključne reči: *SOAP, XML, PL/SQL* procedure, Web servis, *UTL_HTTP(s), Oracle Wallet*

Sadržaj:

1. Uvod

Pojava globalne ekonomske krize i istovremeni eksponencijalni razvoj informacionih tehnologija u današnjim uslovima rezultirali su sve intenzivnijom potražnjom za adekvatnim, ali istovremeno univerzalnim sistemskim rešenjem među kompanijama, kome je dugi niz godina prethodilo međusobno, takmičenje za primat na tržištu. Osnova svih integracija dobila je jedno novo značenje, a podrazumeva prvenstveno integraciju informacija čime se postiže podrška svim poslovnim funkcijama, procesima, aktivnostima i operacijama sa jedne, ali i podržava odlučivanje svih nivoa menadžmenta kompanije koji teže ka integrisanim informacionim resursima zajedničkim za sve aplikacije i korisnike, sa druge strane.

Centar diskusija poslednjih decenija u mnogim informacionim oblastima jeste na koji način izabrati rešenje koje će omogućiti da se najoptimalnije iskoriste informacioni resursi. Većina kompanija danas u svom posedu ima računarske sisteme različitih proizvođača, tehnologija, starosti, koji ujedno koriste različite aplikacije. Jedan od pokazatelja koji direktno potvrđuje i perspektivu jedne potpuno nove tehnologije je taj da brojne (softverske) kompanije poput *Oracle, Novell, Microsoft, IBM, HP* gotovo svakodnevno podržavaju razvoj i nude određena rešenja na tom području. Sve veće kompanije, sa sve kompleksnijim rešenjima koja nude postepeno su uvela složeniju komunikaciju klijenta i servera. Prilagođavanje svojih proizvoda nije samo na internom nivou; od ključnog značaja su promene podložne eksternim faktorima poput zadovoljenja zahteva sve većeg broja kupaca i korisnika. Sve to, gledajući globalno, rezultovalo je sve bržim razvojem distribuiranih aplikacija baziranim na softverskim komponentama. Paralelno sa razvojem web tehnologija i sve većim brzinama prenosa, razvijaju se novi tipovi usluga na Internetu. Budućnost donosi potpuno objedinjavanje kako *web* aplikacija, tako i *desktop*/mobilnih platformi. Zajedničkim naporom velikog broja vodećih tehnoloških kompanija nastala je strogo definisana spcifikacija *Web* servisa kao novog metoda za adresiranje problema integracije raznorodnih sistema, na potpuno standardan način. Budući da se komunikacija na ovakav način obavlja putem *XML*-a, time je otvoren put za heterogene sisteme koji prevazilaze nekompatibilnosti između postojećih razvojnih okruženja upotrebom integracionih tehnologija. Opšte poznata priča iz svakodnevice je ta da se današnje tehnologije mogu opisati sa samo dva pojma- **heterogeno** i **promenljivo**. Kako bi što duže opstali na tržištu,

fundamentalo načelo kompanija jeste prilagođavanje proizvoda pod uticajem svih faktora. Sve to, proporacionalno povećava zahteve za transferom kako informacija, tako elektronskih dokumenata i znanja, što rezultuje razvojem distribuiranih aplikacija baziranim na softverskim komponentama.

2. Metodologija naučnog istraživanja

2.1. Predmet istraživanja

Kroz jednostavan teorijski prikaz osnovnih funkcionalnosti *Web* servisa, i upoznavanjem sa osnovnim konceptima rada istih, predmet istraživanja ovog rada utemeljen je na pitanjima sta su uopšte *Web* servisi, kakva je njihova uloga u realnom svetu. Kako je sigurnost jedna od važnijih komponenti u današnjim *IT* okruženjima, u ovom radu kroz teorijska objašnjenja, praktične prikaze, definicije i prednosti korišćenja ovakvih usluga, ideja o *Web* servisima nije ostala samo na teoriji bez prakse, a dodatni doprinos smeštanja servisa u realni svet samo je jedna je od glavnih tema koja se obrađuje u ovom radu.

2.2. Ciljevi i zadaci

Cilj ove tehnologije, koja je i dalje u razvoju, jeste omogućiti povezano (*B2B, business-to-business*) poslovanje, odnosno distribuciju softverskih komponenti bez obzira na platformu na kojoj su realizovani i na kojoj se izvršavaju, koji je programski jezik korišćen. Time se postiže spajanje milionskih komponenti dostupnih preko Interneta čija je osnovna karakteristika upotrebljivost na širokom novou. U ovom radu, prikazana je implementacija u realnom sistemu, sa glavnim ciljem osigurati komunikaciju između učesnika uz korišćenje *Oracle wallet* rešenja.

3. Pojam Web servisa

U vreme kada *Web* servisi nisu raspolagali velikom brzinom i masovnim korišćenjem, problem labavo povezanih struktura na Internetu (engl. *loosely connected*) bio je nezaobilazan. Zahvaljujući razvoju i napretku informacionih tehnologija, samim tim i širokopojasnim vezama (engl. *broadband Internet access*) sve veće platforme mogu pristupati Internetu putem pretraživača čime se komunikacija podiže na viši nivo - nivo međusobne komunikacije i interakcije, čime su stvorene *web* aplikacije. Računarstvo koje je zasnovano na pružanju usluga, jedan je novi pokret koji se zasniva na izradi (javno) dostupnih servisa čime se postiže određena funkcionalnost. Obavljanje poslova putem, do sada uobičajenih programskih alata, više nije na snazi, a primat zauzimaju manji gradivni blokovi - **servisi**. U zavisnosti od svoje primene, jednostavni ili složeni pa čak i hibridni u nekim okruženjima, zastupaju zajednički stav- pružanje što potpunije usluge i dostupnosti servisa putem jedinstvene globalne mreže. Rastavljanjem složenijih kompanijskih zahteva na niz manjih i prostijih, na prvi pogled pružaju smanjenje broja potencijalnih grešaka u kodu aplikacija i ubrzavaju njihovo optimalno vreme razvoja. Generalno, servisi su realizacija softverskih entiteta koji vrše interakciju sa drugim servisima nezavisno od broja komponenti sa krajnjim ciljem ostvarivanja funkcije koju reprezentuju.

Kako govorimo o pojmu servisa, važno je napomenuti da se takva arhitektura softvera- servisno orijentisana arhitektura (engl. *service-oriented architecture- SOA*) fokusira na realizaciju zadataka kroz logičko rešavanje problematike višeslojnih aplikacija, pa je zbog toga *SOA* moguće posmatrati kroz prizmu više funkcionalne a ne tehnološki orijentisane arhitekture. Integracijom se uvodi dodatni nivo koji zapravo predstavlja model koji će biti primenljiv ne samo na pojedinačne aplikacije već uopštenu platformu za interoperabilnost u heterogenim okruženjima. Opstanak na sve dinamičnijem tržištu i odgovor zahtevima tržišta zavisi od:

- Očuvanja postojećih, primarnih resursa kompanije i pored dodavanja novih funkcionalnosti postojećim komponentama poslovnih servisa;
- Transparentnosti intrene strukture kompanije čime se postiže integracija u heterogenim okruženjima ;
- Brze reakcije na zahteve korisnika, čime *SOA* smanjuje efektivno vreme razvoja softvera– nema potrebe za ponovnim razvojem aplikativnih rešenja iz korena;

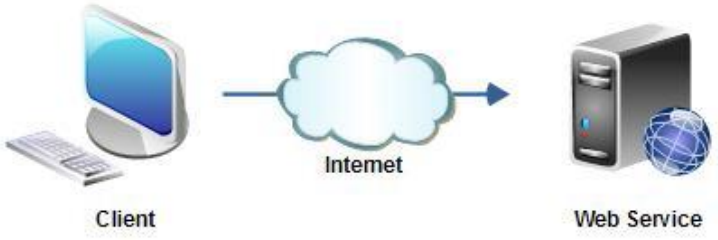
- Niže cene i re-upotreba kroz razvoj autonomnih komponenti

Kao logičan sled događaja, nastavak razvoja Interneta krenuo je u pravcu sve većeg investiranja u razvoj modularnih, samoopisujućih aplikacija, takozvanih *Web* servisa u čijoj je osnovi mogućnost 'pozivanja' sa bilo koje tačke na *web*-u. Drugačije rečeno, pod pojmom *Web* servis, podrazumeva se bilo koja (softverska) aplikacija dostupna preko mreže, čiji je cilj da, bez obzira na platformu i jezik na kojoj je bazirana, podrži interakciju celokupnog sistema.

- *W3C,* konzorcijum za standardizaciju tehnologija korišćenih na Internetu, definiše *Web* servis generalno kao: *softverski sistem dizajniran da podrži interoperabilnu (mašina sa mašinom) interakciju preko mreže*. [1]

Kao softver, čija je prvenstveno namena da podrži interakciju između mašina, servisi imaju za cilj i da na sttandardna način obezbede u svakom trenutku obavljanja poslovih procesa, interoperabilnost različith aplikacija na različitim platformama i pre svega različitim radnim okruženjima. Vremenom je razvijen prepoznatljiv standard što je vrlo brzo globalno prihvaćeno kao preporuka za dalji digitalni zapis podataka u informatičkoj industriji. *SOAP (*engl. *Simple Object Access Protocol)* je tada postao prihvaćen protokol od strane velikih kompanija koje su se aktivno uključile u dalji razvoj sa ciljem da postane protokol za preporuku prilikom razvoja klijentskih aplikacija.

Iako najčešće egzistiraju u distribuiranim sistemima, njihova primena moguća je i u sklopu privatnih *LAN*-ova (interna mreža po modelu klijent-server) (Slika 1-Pozivanje *Web* servisa putem Interneta), gde im se takođe pristupa kao *Web* stranama, koristeći određen protokol za transport poruka. Osnovni princip *Web* servisa prikazan je na sledeći način:



Slika 1-Pozivanje *Web* servisa putem Interneta

Varijacije u funkcijama, od najjednostavnijih do onih složenih, poput onih za konverziju valuta u okruženjima poslovnih banaka, ili složenijih poslovnih sistema, samo su jedan od proizvoljnih faktora za sam razvoj i nastanak servisa. Kada se on stavi u funkciju, druge aplikacije i servisi mogu ga slobodno otkriti i pozvati, koristeći najrazličitije izvore, bez obzira gde se nalaze i kako su implementirani.

Postavlja se pitanje, da li uopšte postoji razlika između *Web* servisa i *Web* aplikacija? Njihova različitost ogleda se u samoj funkcionalnosti iako se na prvi pogled čini kao da su isti. Naime, *web* aplikacije imaju prezentacionu ulogu te pružaju informacije putem internet pretraživača (*web* strana). U ovom slučaju govorimo isključivo o statičkim informacijama, jer bez obzira na to koliko će korisnika zahtevati pregled određenog sadržaja, taj sadržaj će svima biti prikazan na identičan način. Dakle, ukupan korisnički unos zavisi od prezentacionog sloja. Drugo, kod *web* aplikacija obično se celokupna integracija radi na jednoj platformi. Tada korsnik na Internetu ne mora biti upućen u instalaciju i održavanje same aplikacije. Njihova popularnost upravo zbog ove funkcionalnosti je porasla, a vremenom su iz *single tier* modela (jedan rang) prešle u *three tier* model (svaki rang odgovoran za jedan nivo usluge- prezentacija, aplikacija i skladištenje). Za razliku od toga, *Web* servisi podržavaju potpuno drugačiji koncept. Prvenstveno, *Web* servisi nisu inicijalno namenjeni za upotrebu od strane korisnika direktno. Njih koriste aplikacije (programi), a podaci koje korisnik dobija od servisa u potpunosti zavise od parametara koje on prosleđuje servisu. Ne zavise od prezentacionog sloja, poput svog prethodnika. Pojednostavljeno, aplikacije šalju *Web* servisima ulazne parametre a kao rezultat toga dobijaju funkcionalno zavisne izlazne podatke. Zavisnost ulazno/izlaznih parametara ogleda se u samoj nameni aplikacije i njoj određenog servisa. Pošto šalju samo odgovor (parametre komunikacije) time direktno utiču na opterećenost mreže resursima. To bi značilo da su štedljivi po pitanju opterećenosti mreže i svih

mrežnih resursa, jer prilikom komunikacije oni šalju samo odgovor, dok aplikacije šalju pored odgovora i kompletan interfejs. Sama ideja računarsva zasnovanog na servisima je da osigurava upotrebu najrazličitijih platformi prilikom ostvarivanja različith funkcionalnosti. Fleksibliniji su i prilagodljiviji po mnogim parametrima, a jedan od njh je upravo fleksibilnost u pogledu samog izgleda aplikacije, dok korisnik *web* strane nema uticaj na sam prikaz informacija na svom ekranu. To ne znači da njima nismo u mogućnosti da pristupamo putem internet pretraživača, suprotno. Neki servisi su dostupni samo putem intraneta, čime su orijentisani samo na pojedine segmente poslovanja. U poređenju sa tradicionalnim klijent-server arhitekturama, odnosno modelu kao najzastupljenijem modelu na Internetu, napravljen je veliki pomak u razmeni podataka i samom obavljanju poslova putem Interneta. Subjekti koji iniciraju ali istovremeno i završavaju bilo kakvu konekciju sa serverom u modelu klijent-server su *web* klijenti. Njihova zavisnost je usko povezana. Koriste razne protokole i otvorene standarde za izvršenje svojih poslova. Međutim, primenom *Web* servisa, dolazimo do modela koji ne isključuje krajnjeg korisnika aplikacije (u osnovi je baziran na ovakvom modelu već suprotno, omogućava ravnopravnu komunikaciju između višerodnih aplikacija na identičan način kao prethodni model. U ovom modelu, klijent ne mora da ima prethodno znanje o servisu pre nego što napravi poziv ka njemu.

Naredno pitanje koje se postavlja jeste šta zapravo razlikuje koncept *Web* servisa u odnosu na druge distribuirane softverske sisteme? Odgovor leži u sledećem [2]:

- Transparentnost (engl. *transparency*). Pod ovim pojmom, podrazumeva se kako jezička, tako i platformaska transparentnost. Njihovo objavljivanje i upotreba ne sme da zavisi od platforme na kojoj egzisitraju i programskog jezika u kome su napisani.
- Modularnost (engl. *modular design*), modularan dizaj. Nadogradnja novih na stare servise. Zvezda vodilja prilikom kreiranja servisa mora biti: početi sa jednostavnijim operacijama, a onda njihovo grupisanje u servise koji mogu biti organizovani da rade sa drugim servisima. U osnovi, *web* servisi su mali delovi softvera (moduli) iz kojih se mogu dizajnirati proizvoljno veliki sistemi.
- Otvorena infrastruktura (engl. *open infrastructure*). Protokoli i jezici poput *XML*, *HTTP*, *JSON* i drugih su po svojoj prirodi standardizovani (nezavisni od proizvođača) i sve prisutni čime promovišu interoperabilnost između servisa.

U pozadini svakog servisa nalazi se interfejs opisan na jeziku koji ‘trpi’ mašinsku obradu. Taj jezik, poznatiji kao *WSDl* (engl. *Web Service Description Language*) nudi opis usluga koji se koriste putem servisa, dok drugi sistemi komunikaciju sa servisom ostvaruju preko unapred definisanog načina i korišćenjem *SOAP* (engl. *Simple Object Access Protocol*) poruka koje se prenose standardnim otvorenim *HTTP* protokolom. U osnovi rada servisa nalazi se univerzalni jezik za razmenu podataka na Internetu, *XML* (engl. *Extensible Markup Language*) koji je pogodan i razumljiv korisnicima ali i računarima. Drugačije rečeno, *WSDL* je gramatika za opisivanje *Web* servisa, zasnovana na *XML*-u; njime se definišu i daju opisna svojstva servisima, na isti način na koji *header*-i definišu i opisuju standardne binarne fajlove. Pod pojmom opisa *web* servisa, podrazumeva se sve ono što je od važnosti za ostvarivanje komunikacije sa servisom, uključujući format poruka, protokole za transport podataka i njihovu lokaciju.

U nastavku teksta, biće ukratko opisan rad osnovih *Web* servisa.

3.1. Osnove rada Web servisa

Kao što je pomenuto, *Web* servis standard pruža neutralni model komunikacije i čitljivost informacija nezavisno od tipa računara i operativnog sistema koji se na njemu nalazi. U zavisnosti od protokola koje koriste za osnovnu komunikaciju (*FTP, HTTP, SMTP, JMS*), postoji nekoliko vrsta *Web* servisa. Iako se ne smatra nužnim, protokol koji je najzastupljeniji u komunikaciji između klijenta i servera je *HTTP* protokol, a u prilog tome ide činjenica da uređaji koji rade na mrežnom *OSI* sloju (primera radi *firewall* uređaji) dozvoljavaju protok informacija po portu 80, čime komunikaciju među aplikacijama čine jednostavnom i dinamičnom. Upravo ovaj protokol i njegova kombinacija sa *SSL/TLS* (engl. *Transport Layer Security*, i prethodnik, engl. *Secure Sockets Layer*) protokolom biće predmet ovog rada, koristeći pri tome proceduralno proširenje *SQL* jezika (engl. *PL/SQL, Procedural Language/Structured Query Language*) u *Oracle* okruženju. Kako bi se što bolje razumeo način korišćenja *Web* servisa preko *https* protokola uz pomoć *PL/SQL*-a, važno je i razumeti osnovu rada svakog *web*-orijentisanog servisa. Zajedničko za sve jeste to da sama aplikacija brine o svim detaljima, dok je prava funkcionalnost servisa na samom serveru. Drugačije rečeno, *Web* servise možemo posmatrati kao ‘*developer-friendly*’ servisne sisteme. Povezujući minimum dve softverske aplikacije, servisi

postaju međunivo (engl. *middleware*) za interakciju i razmenu podataka u distribuiranim, heterogenim okruženjima.

Osnova funkcionisanja *Web* servisa je po modelu *request-response (request-response* paradigma*),* a u nastavku teksta biće opisana kroz servisno-orijentisanu arhitekturu *SOA* (engl. *Service-Oriented Architecture*), nakon što se objasne osnovni pojmovi vezani za samu razmenu između entiteta.

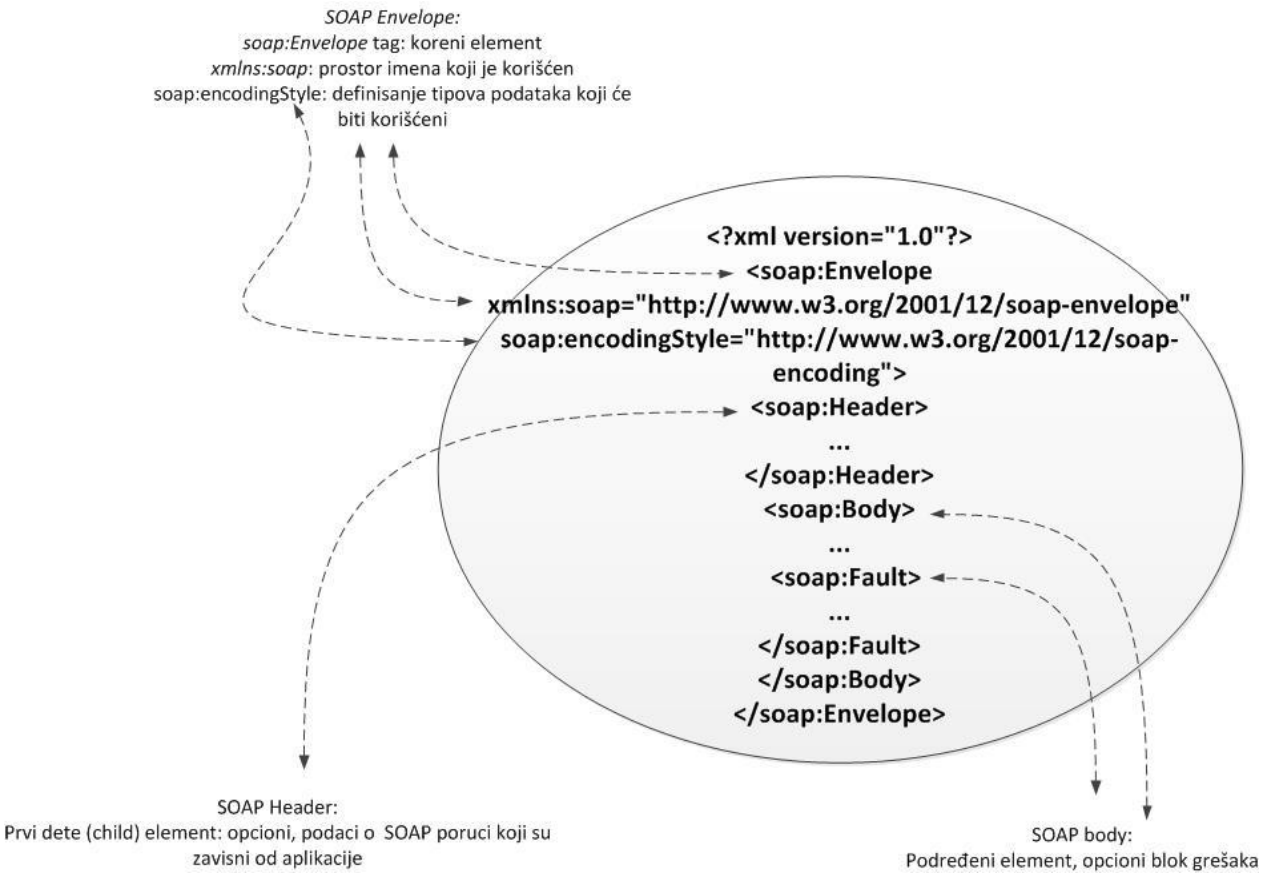
Komunikacija izmedju entiteta (u daljem tekstu *provider* i *requester*) odvija se putem Interneta, što znači da se sva komunikacija odvija putem mrežnih protokola, dok se na aplikativnom nivou koristi *HTTP* protokol. Razmena podataka, poruka između ovih entiteta definisana je *SOAP* specifikacijom (engl. *Simple Object Access Protocol*), koja jasno daje odgovore na pitanja koji je format zapisa podataka u poruci, koji je format zapisa same poruke, koji je standardni način njenog slanja komunikacionim kanalom, i druge konkretne parametre. Svaki *SOAP* servis mora da pruži javni interfejs dostupan preko specifičnog *URL*-a. Ova 'ulazna tačka' drugačije se naziva '*endpoint'*. To bi značilo da u trenutku kada korisnik pokušava da koristi usluge nekog *SOAP Web* servisa, on mora da zna krajnju, finalnu *endpoint* tačku pristupa ali i samu definiciju interfejsa. Protokol za razmenu podataka u *Web* okruženjima, utemeljen je na *XML*-u, pa ima dvosmernu ulogu- služi za prenos poziva metoda od entiteta *requester* do entiteta *provider* (u daljem tekstu samo *requester*, *provider*) kao i za prenos u suprotnom smeru. Slanju prethodi kreiranje nove *SOAP* poruke u čijem se zaglavlju (engl. *header*) i telu (engl. *body*) poruke smeštaju podaci, dok svi zajedno (Slika 2-Struktura *SOAP* poruke) [3] čine *SOAP* omotač (engl. *envelope*). Inicijalna verzija protokola, 1998.godine nije podržavala *XML* način zapisa podataka, ali je za razliku od toga podržavala primitivnije tipove podataka pisane u strukturama, dok je slanje i prijem podataka vršen posebnim operacijama i metodama. Ovakav jezik je nezavisan od operativnog istema i drugih programskih jezika, pa stoga u potpunosti podrzava *Unicode* za internacionalnu upotrebu jezika za definisanje novih formata u Web okruženjima. Činjenica koja ide u privolg priči o *XML*-u je to da je to klasičan *.txt* fajl za generisanje i editovanje *XML* dokumenata, što bi značilo da je zapravo kompletna tehnologija za definisanje same strukture podataka, sadržaja dokumenata sa elementima prikaza podataka (u poređenju sa *HTML*-om koji je kombinacija elemenata i strukture). Njihova mogućnost automatske obrade – ***parsiranje***, osim što je inverzan formatiranju *XML* poruka, omogućava proveru validnosti dokumenta (sintaksa i

semantika) kao i prenos dokumenata do aplikacije (ili prenos dela dokumenta do aplikacije) koja dalje obrađuje *XML* dokument.

Slika 2-Struktura *SOAP* poruke

Sintaksna pravila svake *SOAP* poruke, govore nam da ona mora da bude ispravno formatiran *XML* dokument, a u tome joj pomažu *SOAP* envelopa kao i *SOAP XML* prostor imena. Sa druge strane, takva poruka sme da sadrži reference na *DTD* (engl. *Document Type Definition*), ali i bilo kakav drugi vid naredbi za procesiranje *XML* dokumenta (direktiva za procesiranje *XML*-a). semantička pravila koja utiču na dobro formiranu poruku, dokument predstavljaju se *DTD* fajlovima pomoću kojih se definiše struktura podataka, elementi i njihovedeklaracije, spisak atributa i entiteta i njihove međusobne relacije.

Sastavni delovi *SOAP envelope* opisani su u nastavku, a upoznavanjem sa glavnim karakteristikama ovih elemenata jasno raspoznajemo gde se smeštaju podaci, kako se proširuje poruka i način na koji se predstavljaju greške (Slika 3- *SOAP Envelope*).



Slika 3- ***SOAP Envelope***

- *SOAP Envelope* sa svojim *encodingStyle* atributom predstavlja identifikartor (koverta) da je *XML* dokument zapravo *SOAP* poruka, i ovaj atribut nasleđuju rekurzivno svi podređeni elementi *SOAP* poruke (pod uslovom da oni ne sadrže sopstvene atribute). Definiše sva moguća pravila prema kojima će se podaci enkapsulirati (zaštititi, sakriti i kontrolisati pristup komponentama nekog objekta; karakteristika objektno-orijentisanog programiranja da olakša grupisanje podacima i metoda koje operišu nad tim podacima) prilikom razmene. Dodatne informacije od značaja mogu biti adresa kome je poruka namenjena, imena metoda, povratne vrednosti parametara. *SOAP* poruka nema podrazumevano enkodovanje, pa joj se prosleđuje sintaksa *soap:encodingStyle="URI.* Ovaj element nije podrazumevani element poruke, pa se mora definisati svaki put.

Sadrži dva podelementa:

- *SOAP Header* smatra se podređenim elementom *SOAP envelope*; sadrži informacije o pošiljaocu, primaocu i nameni poruke. Nije obavezan element, a njegovi atributi definišu kako primalac zahteva treba da obrađuje *SOAP* poruku.
- *SOAP Body* takođe podređeni element *envelope*; predstavlja sadržaj same poruke kao i opcioni blok grešaka koje se javljaju u cilju obrade podataka (prilikom razmene *SOAP* poruke, mogu se pojaviti greške- *detail, faultcode, faultstring, faultfactor*). Smatra se obaveznim elementom. Podaci koji se nalaze unutar ovog elementa i njihovo predstavljanje naziva se *SOAP* kodiranje. Pojam kodiranje veoma je bitno razumeti, jer upravo su poruke sredstvo komunikacije pojedinih aplikacija, a neophodno je da one budu formatirane na način koji im je razumljiv. Smatra se da ima sopstvenu strukturu koja nije vezana za sam *envelope*. Ovaj element sadrži set pravila kodovanja tipova podataka u poruci.

Primer povezivanja *HTML* i *SOAP* vrši se sledećim načinom:

```
POST /item HTTP/1.1

Content-Type: application/soap+xml; charset=utf-8
```

Osnovne faze komunikacije koristeći *SOAP* protokol mogli bi podeliti u nekoliko koraka:

- Kreiranje zahteva, *XML* dokumenata od strane *SOAP* klijenata
- Slanje zahteva serveru pomoću protokola *HTTP*
- Kreiranje odgovora, takođe *XML* dokumenata na osnovu specifikacije zahteva od strane klijenata
- Slanje odgovra klijentu, takođe pomoću protokola *HTTP*

Naslednik je *XML-RPC* protokola koji služi za ramenu podataka i pozivanje procedura na udaljenim računarima (engl. *Remote Procedure Call*), od koga je preuzeo identična pravila transporta, ali sa bitnom razlikom u načinu formatiranja poruka [4]. Ovaj protokol omogućava pozivanje procedura i njihovo izvršenje u drugom adresnom prostoru (drugi računar ili sasvim

druga mreža) bez dodatnih programerskih podešavanja za udaljenu interakciju. Predstavlja jedan vid klijent-server konfiguracije, implementiranu kroz *request-response* paradigmu prenosa poruka kroz sistem.

Kod i sastavni delovi *SOAP envelope* na osnovu koje je rađena praktična upotreba *web* servisa objašnjena u ovom radu, data je u nastavku teksta:

```
POST http:// <ip address>:2003/MUP/ZinNbsPrijem HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"xmlns:ws="http://ws/">
        <soapenv:Header/>
        <soapenv:Body>
        <ws:eZinNbsOtpremnica>    <!--Optional:-->
                                <arg0>?</arg0>
                                <!--Optional:-->
                                <arg1>?</arg1>
                                <!--Optional:-->
                                <arg2>?</arg2>
        </ws:eZinNbsOtpremnica>
        </soapenv:Body>
        </soapenv:Envelope>
```

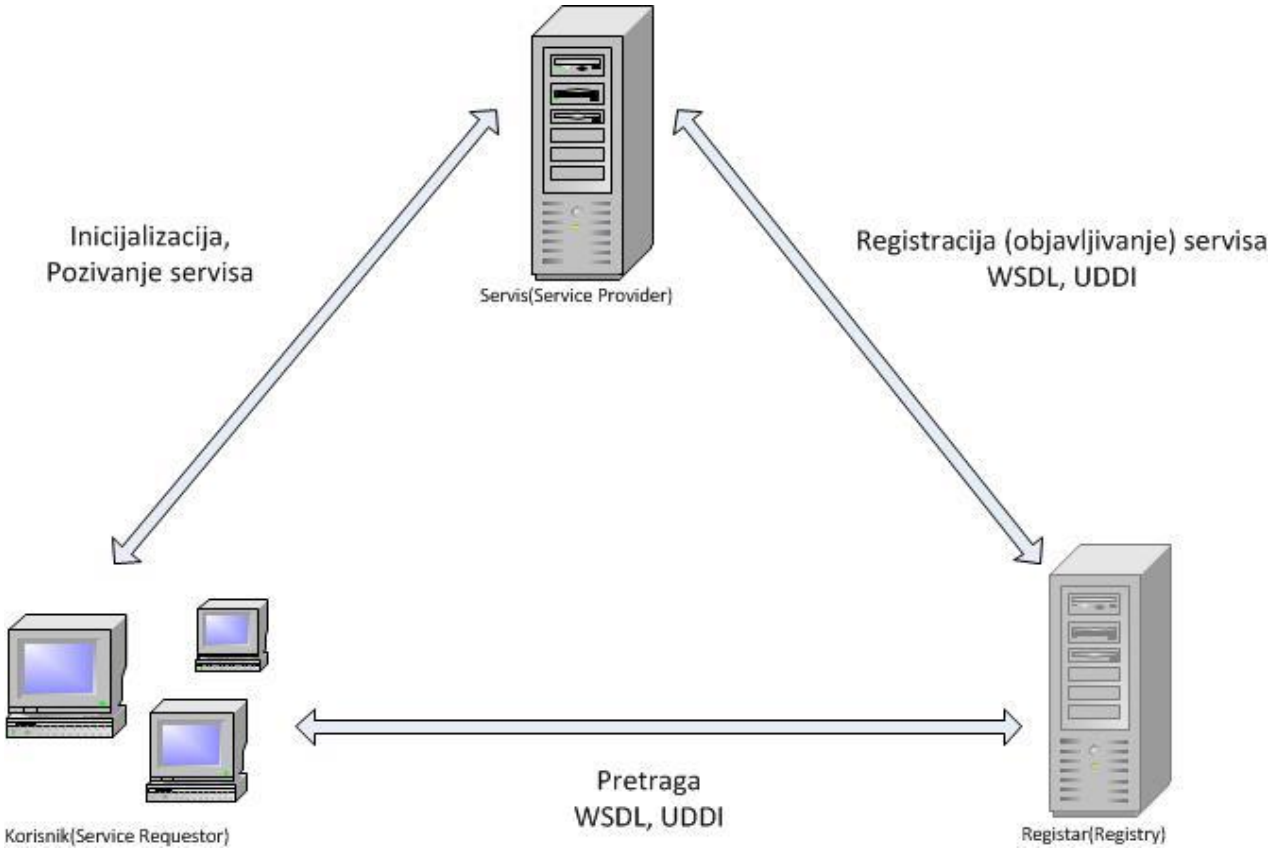
Svoju funkcionalnost, servisi postižu time što su razmenjeni podaci zapisani u *XML* obliku. To bi značilo da kada *Web* aplikacija (*requester*) želi da pruži *proivider*-u podatke on će te iste podatke zapisati u gorenavedenom formatu, a zatim ih adresirati suprotnoj strani. Aplikacija te ulazne podatke najpre slaže u *SOAP* poruku koja sadrži dodatne parametre, takođe zapisanim u *XML* obliku, čiji skup tagova prethodno nije definisan kako bi se zadovoljila osnovna namena ovog jezika- čitljivost na bilo kom računaru. Važno je napomenuti da kao ovakav, *SOAP* protokol nije

isključivo transportni protokol, već je jednostavan protokol za razmenu informacija između uređaja koji imaju ograničenja u vidu ravnopravnih komunikacionih mogućnosti u distribuiranim okruženjima, kao i decentralizovanim okruženjima poput Interneta.

Ovakav oblik višeslojne organizacije računarskih sistema, deljenje logike poslovanja obezbedjuje kroz tri ključne entitet-uloge (role) o kojima će više reči biti u nastavku. To su:

- Entitet *provider* rola
- Entitet *requester* rola
- Entitet *servise registry* rola

Pored osnovnih uloga entiteta, od ključnog značaja su operacije koje se izvršavaju između njih. Sa slike (Slika 4- *Web* servis, role i operacije) se vidi osnovni princip rada *Web* servisa.



Slika 4- *Web* servis, role i operacije

Entitet *provider* **definiše** (opisuje) svoj servis korišćenjem *WSDL* jezika i objavljuje ga entitetu registar (engl. *registry*) što zapravo predstavlja katalog servisa (definicja servisa korišćenjem *WSDL*-a objavljuje se u katalogu servisa). Trenutno najkorišćeniji registar je *UDDI* (engl. *Universal Description, Discovery, and Integration*) koji kao svoju osnovnu funkciju ima da centralizuje servise u jedan zajednički registar koji omogućava učesnicima u komunikaciji da jednostavno objavljuju i pronalaze traženi servis. Drugačije rečeno, predstavlja centralizovanu bazu podataka o *Web* servisima koji čuva zapise o kompanijama ali i servisima koje te kompanije nude, u formatu *XML*-a. Sadrži tri osnovne vrste informacija:

- **Bele strane** (engl. *white pages*): deo u kome su definisani osnovni podaci o kompaniji- poslovno ime, adresa, telefonski broj, *web* sajt.
- **Žute strane** (engl. *yellow pages*): pružaju dodatni nivo informacija o kompaniji- identifikacija posla, kategorizacije, tačnije sve ono što bi omogućilo lakšu pretragu informacija o kompaniji.
- **Zelene strane** (engl. *green pages*): pružaju tehničke informacije o samim servisima. U ove elemente spadaju: *URL* adrese, *Discovery* informacije, način ostvarivanja interakcije sa servisom.

Međutim u praksi je veoma čest slučaj korišćenja drugih registara, ali se svi zapisi realizuju pomoću *XML* elemenata, kao u datom primeru ispod.

<businessEntity>, <businessService>, <bindingTemplate>, <tModel> i <publisherAssertion>

Bitna napomena je da *UDDI* nije ograničen samo na opisivanje web usluga baziranim na *SOAP* protokolu- može se korisititi za opisivanje bilo koje druge usluge, poput *CORBA*, *Java RMI* i drugih usluga.

Koristeći operaciju **pretrage** direktorijuma registra, korisnik, ili aplikacija koja ima ulogu *requestor*-a, mapira (locira) željeni servis i saznaje način kako da ostvari komunikaciju sa pružaocem usluga koji entitetu na suprotnoj strani dostavlja deo *WSDL* opisa servisa radi ostvarivanja komunikacije. U skladu sa svojom funkcijom, servis obezbeđuje konkretan odgovor (na prethodno postavljen zahtev od strane korisnika) i dostavlja ga korisniku određenog servisa, čime je zadovoljen poslednji korak- operacija **inicijalizacije** (pozivanje, engl. *invocation*) *Web*

servisa. Korisnik servisa koristi *WSDL* opis i šalje zahtev servisu, koji u skladu sa svojom funkcionalnošću obezbeđuje odgovor i dostavlja ga krajnjem korisniku.

Servis može imati više klijenata koji mu se obraćaju, pri čemu se vodi računa da komunikacija bude potpuno transparentna - servis i klijenti ne moraju biti napisani u istom jeziku. Ukoliko je slučaj da ne postoji identičnost u progrmskom jeziku, uvodi se jedan novi, središnji sloj koji će imati ulogu da manipuliše razlikama u tipovima između klijentskog i jezika servisa. Generalno govoreći, nastoje da u svakom trenutku imaju što jednostavniju strukturu, ali i sama struktura servisa može biti definisana tako da može da bude sačinjen i od drugih servisa, podservisa. Sve poruke koje se razmenjuju između servisa i njegovog krajnjeg korisnika realizovane su *SOAP* protokolom, koji su zajedno sa *WSDL* opisom servisa bazirani na *XML* jeziku. Najaktuelnija *SOAP* specifikacija može se pronaći na mnogobrojnim web stranama [5].

3.1.1. Prednosti i nedostaci *SOAP* poruka

U nastavku teksta, ukratko će biti predstavljene prednosti i nedostaci *SOAP* protokola.

- Prednosti:

Neutralna karakteristika *SOAP* protokola (nezavisnost od platforme i programskog jezika) čini ga veoma pogodnim za korišćenje u kombinaciji sa bilo kojim drugim transportnim protokolom. Implementacije veoma često koriste *HTTP* kao osnovu za transport poruka, ali se u praksi sve češće susrećemo i sa drugim popularnim protokolma, kao što su *SMTP*, *JMS* i drugi. Poruke razmenjene ovim protokolom su razumljive, čitljive čoveku i jednostavne u *XML* formatu.

U kombinaciji sa *HTTP* protokolom po portu 80, saobraćaj je potpuno transparentan i otvoren prema *firewall*-ovima, pa je time smanjena mogućnost pojave problema u funkcionisanju *Web* servisa kao u slučaju brojnih mrežnih, infrastrukturnih ograničenja, te nema potrebe za njihovom modifikcijom ili uz čak uz minimalni razvoj tehnologije. Počevši od verzije *1.1 SOAP* specifikacije, postiže se jednostavan način udaljene ***objekat-komponenta-usluga*** načina komunikacije, ali se vodi računa o tome da *SOAP* i *HTTP* zaglavlja nemaju istu ulogu:

SOAP zaglavlje definiše ko obradjuje podatke i na koji način, dok *HTTP* zaglavlje vrši prenos informacija do odredišta.

Još jedna od prednosti ovog protokola je to što se od inicijalne verzije uvodi pojam posrednika u komunikacij - zaustavljanja (engl. *stops*) izmedju poziva i *endpoint* tačke koji ne utiču na kvalitet komunikacije, a omogućavaju da ne postoji strogo definisan broj posrednika, već suprotno- lanac posrednika (skalabilne *SOAP* poruke).

- Nedostaci:

Generalno gledano, *SOAP* je malo sporiji u poređenju sa drugim *middleware* standardima, uključujući *CORBA* (engl. *Common Object Request Broker Architecture)* arhitekturu, jer korišćenje *HTTP* protokola nije namenjeno prenosu binarnih podataka, a takav način prenosa binarnih elemenata uzrokuje brojne problema kompatibilnosti. U takvim situacijama, *proxy* server blokira takvu vrstu saobraćaja, pa je bolji način komunikacije prenos tekstualnih podataka.

Interoperabilnost nije pojam koji uvek važi i koji je uvek moguće izvesti. Postojeće implementacije neće uvek podrazumevati pouzdanost servisa. Sa druge strane, mogućnost zaobilaženja *firewall*-a bez velikih poteškoća, ne omogućava sigurnost poziva na određenim portovima. Kretanje informacija i praćenje njihovog puta je složen proces, jer pored pronalaska formata poruke, potrebno je ustanoviti i sam sadržaj istih. Da bi se to omogućilo, dodaju se nove naredbe (tagovi, *verbose*) kao proširenja samog protokola, pa je potrebno dobro razumeti ograničenja performantnosti pre same izrade aplikativnog rešenja oko samog *SOAP* protokola.

4. Sigurnost Web servisa

Ispitivanje sigurnosti jedan je od načina da se utvrdi koliko je određeni segment informacionog sistema neke kompanije ispravno implementiran (u skladu sa svim poslovnim i sigurnosnim zahtevima) ili otporan na brojne neovlašćene aktivnosti. Kako je veličina i kompleksnost informacionih sistema očigledna, veoma je teško na kontinuiran način nadlgedati a samim tim održavati visok nivo sigurnosti u svim njegovim delovima, pa sve veći broj uređaja sa različitim operativnim sistemima i servisima doprinose otežavanju tog zadatka.

Sve dinamičniji razvoj *weba*, i svakodnevni rad korisnika u ovakvim tehnologijama ponudili su dobru osnovu Internet bankarstvu za osnovne funkcionalnosti- plaćanje transakcija putem Interneta, mobilnih aplikacija. Zbog uptrebe uvek istih servisa, postali su meta zlonamernih napada na informacione sastave. Izvori napada postaju neograničeni i time otvaraju vrata mnogim ranjivostima kojima se postiže određena korist za napadače. Izazov za svako poslovanje je da se održi sigurna, otvorena, povezana ali istovremeno mreža jednostavna za razmenu informacija sa poslovnim partnerima, korisnicima. Obzirom na brz i eksponencijalni razvoj informatičke industrije i visokih tehnologija modrenog doba, za novo otkrivene ranjivosti potrebno je iznova osmišljavati načine za zaštitu informacija i sistema. Kao dodatak ovome, moramo napomenuti da ne možemo biti imuni na činjenicu da maliciozni softveri sa kojima se susrećemo imaju konstantu težnju da remete poslovne operacije, čime je zadatak da se osigura dostupnost i sigurnost informacija veoma težak. Računarska mreža i njeni entiteti, ukoliko se pokreću bez postojanja važećih 'zakrpi' (engl. *patch*) koji se koriste za međusobnu komunikaciju, nužno uključuju određeni stepen ranjivosti pa je svako poslovanje vezano za Internet u potencijalnom riziku upravo zbog postojanja tih (mrežnih) ranjivosti. Uklanjanjem njihovih izvora, brinemo o sigurnosti *IT* infrastrukture, sa ciljem unapređenja sigurnosti mreže i mrežom dostupnih servisa.

Nove ranjivosti stupaju na snagu gotovo svakodnevno, kao posledica nedostataka u softveru i programskom kodu generalno, neispravno konfigurisanim aplikacijama, ili čak usled grešaka *IT* inženjera. Računarski znanstvenici procenjuju da je oko 5 do 20 propusta prisutno na svakih hiljadu linija softverskog koda, a rizik za ranjivostima raste sa povećanjem upotrebe *General Public License* softvera naročito zbog implementacije neproverenih i netestiranih modula aplikativnog koda. Kao takvi, moduli često uključuju greške u implementaciji softvera, praveći ih time lakim metama za napade u realnom okruženju.

Obzirom na činjenicu da *SOAP* standard za razmenu poruka (pozivanje samog *Web* servisa) nema implementiran bezbednosni mehanizam, logično pitanje koje se nameće jeste kako i na koji način je moguće osigurati komunikaciju između učesnika, a da se pri istom ne poremete sigurnosni zahtevi. Kao posledica sve većeg značaja ove tehnologije, ali i sve bržoj ekspanziji, servisi postaju aktivni ciljevi napadačima, a poput bilo kog drugog segmenta *IT* sistema, rezulati narušavanja sigurnosti mogu biti katastrofalni za dalje poslovanje.

Do sada tradicionalna sigurnosna rešenja koja su pružala visok nivo zaštite, poput *firewall* uređaja ili mnogobrojne kriptografske metode zaštite, u današnje vreme nisu dovoljna. Sigurnost nije bila zvezda vodilja a ni osnovni zahtev o kome se vodilo računa u najranijim začecima ove tehnologije. Vodeće informacione kompanije shvatile su značaj ovog problema, te je osnovana *WS-I* organizacija (engl. *Web Services Interoperability Organization*) [6] radi promovisanja interoperabilnosti između gomile specifikacija *Web* servisa. Donet je niz standarda koji u svom nazivu sadrže *WS-** oznaku, a neki od njih su: *WS-Policy, WS-Security, WS-Trust, WS-SecureConversation, WS-ReliableMessaging, WS-AtomicTransactions*. Ovakva specifikacija predstavlja klasično proširenje *SOAP* protokola kojim se dodaju sigurnosna obeležja u komunikaciji sa servisom. Primera radi, podržani su *X.509* sertifikati za prenos javnih ključeva, *Kerberos* mrežni protokol za autentifikaciju učesnika komunikacije i mnogi drugi mehanizmi.

Sa druge strane, *Web* servisi, zahtevaju određeni vid zaštite podataka na višestrukom nivou:

- *SOAP* poruke koje se prenose putem Interneta, moraju biti dostavljene na poverljiv način.
- Server mora da bude konfigurisan tako da pruža maksimalnu poverljivost u komunikaciji sa klijentima.
- Klijenti moraju da budu sigurni da u komunikaciji učestvuje relevantan server, kao i da u pitanju nije vrsta krađe identiteta na mrežnom nivou (engl. *phishing*).
- Sistemski logovi poruka trebalo bi da sadrže dovoljno informacija u cilju pouzdanog rekonstruisanja potencijalnih događaja do određenog trenutka.

Sa stanovišta informacione sigurnosti unutar implementacije *Web* servisa, postoji nekoliko očitih tačaka izvora ranjivosti. Komunikacioni kanal i pravo obavljanja određenih akcija- pružanje usluga. Zahvaljujući razvoju tehnologija prvi problem rešen je upotrebom kriptografskih metoda zaštite, tajnosti i autentifikacije korisnika [7]. Drugi problem usko je vezan za pojam autentifikacije, a podrazumeva korišćenje listi za proveru pristupa *ACL* (engl. *Access Control List*) kojima se uparuju korisnici sa određenim dozvoljenim akcijama i korišćenim resursima. Ovim listama preciznije definišemo i ograničavamo pristup resursima na određenu *IP* adresu. Ukoliko govorimo o kritičnijim servisima, primera radi novčanim transkacijama, monitoriše se i prati svaki korak korisnika. Time se lakše rekonstruišu nizovi sprovedenih akcija unutar sistema i postavljaju se mogući sankcioni faktori za buduće, moguće pokušaje napada.

Svaka organizacija, u cilju svog uspešnog poslovanja poseduje poverljive podatke koje želi da zadrži sigurnima. Budući da takvi podaci ne smeju biti javno dostupni (moraju biti tajni), ne smeju biti podložni promenama bez prethodnog odobrenja i ne smeju biti nedostupni korisnicima. Važno je sprovesti određene mere i u svakom trenutku obezbediti tri sigurnosna zahteva- **poverljivost (tajnost), dostupnost** i **integritet**.

Svi faktori u nastavku teksta naročito su fundamentalni i od velikog značaja za pravilno funkcionisanje svih delova jednog složenog informacionog sitema, a ujedno i samih *Web* servisa.

4.1. Poverljivost (tajnost)

Poverljivost (tajnost, engl. *confidentiality*) odnosi se na ograničenje u pristupu i otkrivanju informacija samo ovlašćenim korisnicima. Iako vlada mišljenje da je ovaj tip sigurnosnog zahteva od najveće važnosti za državne institucije i vojsku, on može biti značajan i za kompanije koje imaju potrebu za zaštite informacije od strane konkurencije ili neovlašćenog pristupa. Kako bi bio ostvaren, ključni aspekti poverljivosti su identifikacija korisnika i provera autentičnosti. Poverljivost se može izgubiti na mnogo načina- pogrešnim definisanjem i sprovođenjem prava pristupa mreži.

4.2. Integritet

Integritet podataka (engl. *integrity*) odnosi se na zaštitu podataka od namernog (ili slučajnog) menjanja sadržaja istih. Glavni zahtev komercijalnih i državnih institucija jeste osigurati integritet podataka kako bi se izbegle greške i zloupotrebe. Kao i poverljivost, integritet može biti ugrožen, naročito od strane hakera, lažnog predstavljanja, neovlašćenih aktivnosti i nedozvoljenih programa (virusi, trojanski konji) jer sve navedene aktivnosti mogu dovesti do neovlašćenog menjanja podataka. Podaci moraju biti konzistentni interno i eksterno, što bi značilo da su upravo ti interni podaci konzistentni u svim svojim delovima sa spoljnim okruženjem. Ukoliko govorimo o integritetu komunikacije, mislimo na proces održavanja integriteta protoka informacija u svakom trenutku komunikacije.

4.3. Dostupnost

Dostupnost (engl. *availability*) odnosi se na dostupnost informacionih resursa (pouzdano i pravovremeno pristupanje podacima ili računarskim resursima). Ukoliko *IS* nije dostupan u trenutku kada su informacije potrebne, jednako je loš u istoj meri kao i da ne postoji. Dostupnost sistema (podataka) predstavlja garanciju da je ovlašćenim korisnicima on raspoloživ u svakom trenutku kada za to postoji potreba odnosno obezbeđuje dostupnost sistema koji pruža neke usluge. Primer takve usluge je sprečavanje *DoS* napada ili širenje virusa koji brišu ili imaju mogućnost da oštete datoteke. Dostupnost, zajedno sa pojmom poverljivosti moraju biti očuvani čak uz moguće prisustvo posrednika u komunikaciji.

Poslednjih godina, pored ova tri bitna faktora, sve češće susrećemo se i sa pojmom neporecivosti (engl. *Non-repudiation*) odnosno takozvanim izbegavanjem odgovornosti. Ovaj faktor postaje sve bitniji u vreme kada se veliki deo transakcija (prvenstveno novčanih) obavlja putem nesigurnih komunikacionih kanala. Cilj je ostvariti takvu komunikaciju da bilo koji entitet ne može da poriče svoju akciju ili obavezu. Drugim rečima, to znači da se u svakom trenutku može potvrditi da su entiteti u komunikaciji zaista ti entiteti koji su poslali, ili primili poruke.

Takođe, poslednjih godina, zabeležen je sve veći procenat napada koji se izvršavaju na aplikativnom nivou. Pored dobro poznate kategorije *DOS* napada, primat na ovom polju preuzimaju i propusti u samom dizajnu i implementaciji servisa, poput nepravilne validacije prenosnih parametara (ulazno-izlazni parametri). Manipulacija ovako okarakterisanim parametrima postaje sve lakša, a upravo na taj način dalje nastaju napadi poput *SQLI* i *XPathI* (engl. *SQL Injection*, *XPath Injection*). Nepravilno filtriranje parametara, i to onih koji su upućeni od strane web servis klijenta, doprinose nepravilnoj intrepretaciji informacija iz baze podataka. Ukoliko napadač uspe da ubaci dodatnu *SQL* naredbu u *web* servis, svojim izmenjenim upitom narušio je sigurnost sistema. Indirektno, to bi značilo da je u pitanju manipulacija parametara unutar *SOAP* poruke. Činjenica da je tehnologija *Web* servisa i dalje relativno nov koncept integracije aplikacija na internetu, ne olakšava sprečavanje brojnih napada na same servise, već suprotno. Ova tehnologija se sastoji od skupa *XML* baziranih protokola pri čemu svaki od njih ima tačno određenu ulogu u celom sistemu. Narušavanje samo jednog od njih, može izazvati potpuni zastoj u radu celog sistema. Primera radi, poput standardnih *e-mail* poruka, *SOAP* poruke

mogu sadržati i sopstvene priloge (engl. *attachments*). Ukoliko ni na koji način nije uspostavljena kontrola sadržaja poruka, napadač uz *SOAP* može dodatno da priloži velike dokumente koji kao primarnu svrhu imaju otežavanje komunikacije i samu obradu poruka. Ti dokumenti mogu sadržati i maliciozni kod, čime dovode do zagušenja komunikacionih kanala ili samog servisa. Kasno, njihovo neblagovremeno tkrivanje može naneti milionske štete kompanijama. Jedan od načina zaštite servisa jeste striktna provera podataka koje će aplikacija preuzeti od korisnika kao i u obrnutom smeru. Problematika koja je očigledna nameće brigu od samog početka razvoja aplikativnog rešenja, a ne kroz upotrebu i testiranje gotovog programerskog proizvoda.

Sa druge strane, nesigurnost kanala putem kojih se odvija komunikacija oduvek je predstavljala problem naročito kod distribuiranih sistema. Servisno orijentisana arhitektura podrazumeva ostvarivanje komunikacije preko otvorene, javne mreže čime postaje osetljiva na gore opisanu problematiku. Ukoliko pored toga sama komunikakcija nije dodatno zaštićena, presretanje poruke, njena modifikacija predstavljaju pretnju po tajnost komunikacije. Još jedna od pretnji kod nesigurnih komunikacionih kanala je mogućnost da se u nekoj od pristupnih tački rutiranja, izvrši takozvano zlonamerno rutiranje na pogrešno konfigurisanu adresu. Takva adresa omogućava napadačima da preusmerene poruke mogu da obrađuju a zatim ih vrate, preusmere na inicijalnu adresu. Kako standardni mrežni *firewall* uređaji proveravaju samo *IP*, *TCP*, *HTTP* zaglavlja, ne samu *SOAP* poruku, na žalost ne mogu da pruže odgovarajući nivo zaštite. Zbog toga, često se surećemo sa upotrebom *XML firewall* uređaja koji vrše direktno proveru sadržaja komunikacije. Kao takvi, pružaju kontrolu pregleda i analiziraju dolazne zahteve, a zatim sprovode pojedine akcije u skladu sa utvrđenim sadržajem poruka. I ovakva zaštita ima svoja ograničenja prijema kriptovanih poruka, pa je jasno da je potpuno rešenje samo sigurno projektovan i implementiran sistem.

5. Korišćenje *http* servisa iz *PL/SQL-a*

Poslednjih godina, *web* bazirane usluge dostigle su svoju popularnost do maksimalne tačke kada gotovo sve aplikacije pokušavaju da ih inkorporiraju do određenog nivoa i u izvesnoj meri. Kako je *HTTP* osnovni protokol na kome počiva komunikacija sa *web* servisom, za implementaciju komunikacije, u realnom, poslovnom *Oracle* okruženju, potrebno je implementirati paket kojim je moguće izvršavati *HTTP* pozive iz *PL/SQL* procedura. Verzija *Oracle 9i* omogućava direktan

pristup *web* servisima koristeći *UTL_HTTP* paket, a kako je *SOAP* protokol *XML* baziran protokol, omogućava aplikacijama razmenu podataka preko *HTTP*-a. Naprednija verzija, *Oracle 10g* nudi mogućnost objavljivanja *PL/SQL* koda iz same baze. Njegova (*SOAP_API_pkg*) incijalna verzija prikazana je u nastavku teksta kao i opis varijabli i tipova unutar samog paketa [8] .

```
/* Formatted on 4/29/2016 11:44:08 AM (QP5 v5.256.13226.35538, andjela.zurnadji@nbs.rs) */
CREATE OR REPLACE PACKAGE soap_api
AS
-- ----------------------------------------------------------------------
-- Name        : https://oracle-base.com/dba/miscellaneous/soap_api.sql
-- Author      : Tim Hall
-- Description  : SOAP related functions for consuming web services.
-- License     : Free for personal and commercial use.
--               You can amend the code, but leave existing the headers, current
--               amendments history and links intact.
--               Copyright and disclaimer available here:
--               https://oracle-base.com/misc/site-info.php#copyright
-- Ammedments   :
--   When        Who       What
-- ==========================================================
--   04-OCT-2003  Tim Hall  Initial Creation
--   23-FEB-2006  Tim Hall  Parameterized the "soap" envelope tags.
--   25-MAY-2012  Tim Hall  Added debug switch.
--   29-MAY-2012  Tim Hall  Allow parameters to have no type definition.
--                          Change the default envelope tag to "soap".
--                          add_complex_parameter: Include parameter XML manually.
--   24-MAY-2014  Tim Hall  Added license information.
-- ----------------------------------------------------------------------
TYPE t_request IS RECORD
(
method        VARCHAR2 (256),
namespace     VARCHAR2 (256),
body          VARCHAR2 (32767),
envelope_tag  VARCHAR2 (30)
);

TYPE t_response IS RECORD
(
doc           XMLTYPE,
envelope_tag  VARCHAR2 (30)
);

FUNCTION new_request (p_method        IN VARCHAR2,
```

```
p_namespace      IN VARCHAR2,
p_envelope_tag   IN VARCHAR2 DEFAULT 'soap')
RETURN t_request;


PROCEDURE ADD_PARAMETER (p_request   IN OUT NOCOPY t_request,
p_name      IN          VARCHAR2,
p_value     IN          VARCHAR2,
p_type      IN          VARCHAR2 := NULL);

PROCEDURE add_complex_parameter (p_request   IN OUT NOCOPY t_request,
p_xml       IN          VARCHAR2);

FUNCTION invoke (p_request   IN OUT NOCOPY t_request,
p_url       IN          VARCHAR2,
p_action    IN          VARCHAR2)
RETURN t_response;

FUNCTION get_return_value (p_response    IN OUT NOCOPY t_response,
p_name       IN          VARCHAR2,
p_namespace  IN          VARCHAR2)
RETURN VARCHAR2;

PROCEDURE debug_on;

PROCEDURE debug_off;
END exchangeMUPNBS.soap_api;
/

SHOW ERRORS

**********************************************
CREATE OR REPLACE PACKAGE BODY soap_api
AS
-- ----------------------------------------------------------------------
-- Name        : https://oracle-base.com/dba/miscellaneous/soap_api.sql
-- Author      : Tim Hall
-- Description : SOAP related functions for consuming web services.
-- License     : Free for personal and commercial use.
--               You can amend the code, but leave existing the headers, current
--               amendments history and links intact.
--               Copyright and disclaimer available here:
--               https://oracle-base.com/misc/site-info.php#copyright
-- Ammedments  :
--   When        Who       What
--   ===========  ========
================================================
```

```
--   04-OCT-2003  Tim Hall  Initial Creation
--   23-FEB-2006  Tim Hall  Parameterized the "soap" envelope tags.
--   25-MAY-2012  Tim Hall  Added debug switch.
--   29-MAY-2012  Tim Hall  Allow parameters to have no type definition.
--                          Change the default envelope tag to "soap".
--                          add_complex_parameter: Include parameter XML manually.
--   24-MAY-2014  Tim Hall  Added license information.
-- ------------------------------------------------------------------------
g_debug   BOOLEAN := FALSE;

PROCEDURE show_envelope (p_env       IN VARCHAR2,
p_heading   IN VARCHAR2 DEFAULT NULL);

-- ------------------------------------------------------------------------
FUNCTION new_request (p_method        IN VARCHAR2,
p_namespace      IN VARCHAR2,
p_envelope_tag   IN VARCHAR2 DEFAULT 'soap')
RETURN t_request
AS
-- ------------------------------------------------------------------------
l_request   t_request;
BEGIN
l_request.method := p_method;
l_request.namespace := p_namespace;
l_request.envelope_tag := p_envelope_tag;
RETURN l_request;
END;
-- ------------------------------------------------------------------------
PROCEDURE ADD_PARAMETER (p_request   IN OUT NOCOPY t_request,
p_name      IN          VARCHAR2,
p_value     IN          VARCHAR2,
p_type      IN          VARCHAR2 := NULL)
AS
-- ------------------------------------------------------------------------
BEGIN
IF p_type IS NULL
THEN
p_request.body :=
p_request.body
|| '<'
|| p_name
|| '>'
|| p_value
|| '</'
|| p_name
|| '>';
ELSE
```

```
p_request.body :=
p_request.body
|| '<'
|| p_name
|| ' xsi:type="'
|| p_type
|| '">'
|| p_value
|| '</'
|| p_name
|| '>';
END IF;
END;
-- ---------------------------------------------------------------
PROCEDURE add_complex_parameter (p_request   IN OUT NOCOPY t_request,
p_xml       IN          VARCHAR2)
AS
-- ---------------------------------------------------------------
BEGIN
p_request.body := p_request.body || p_xml;
END;
-- ---------------------------------------------------------------
PROCEDURE generate_envelope (p_request   IN OUT NOCOPY t_request,
p_env       IN OUT NOCOPY VARCHAR2)
AS
-- ---------------------------------------------------------------
BEGIN
p_env :=
'<'
|| p_request.envelope_tag
|| ':Envelope xmlns:'
|| p_request.envelope_tag
|| '="http://schemas.xmlsoap.org/soap/envelope/" '

|| 'xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">'
|| '<'

|| p_request.envelope_tag
|| ':Body>'
|| '<'
|| p_request.method
|| ' '
|| p_request.namespace
|| ' '
|| p_request.envelope_tag
|| ':encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">'
```

```
|| p_request.body
|| '</'
|| p_request.method
|| '>'
|| '</'
|| p_request.envelope_tag
|| ':Body>'
|| '</'
|| p_request.envelope_tag
|| ':Envelope>';
END;


-- ---------------------------------------------------------------
PROCEDURE show_envelope (p_env      IN VARCHAR2,
p_heading   IN VARCHAR2 DEFAULT NULL)
AS
-- ---------------------------------------------------------------
i      PLS_INTEGER;
l_len   PLS_INTEGER;
BEGIN
IF g_debug
THEN
IF p_heading IS NOT NULL
THEN
DBMS_OUTPUT.put_line ('*****' || p_heading || '*****');
END IF;

i := 1;
l_len := LENGTH (p_env);

WHILE (i <= l_len)
LOOP
DBMS_OUTPUT.put_line (SUBSTR (p_env, i, 60));
i := i + 60;
END LOOP;
END IF;
END;

-- ---------------------------------------------------------------
PROCEDURE check_fault (p_response IN OUT NOCOPY t_response)
AS
-- ---------------------------------------------------------------
l_fault_node     XMLTYPE;
l_fault_code     VARCHAR2 (256);
l_fault_string   VARCHAR2 (32767);
```

```
BEGIN
l_fault_node :=
p_response.doc.EXTRACT (
'/' || p_response.envelope_tag || ':Fault',
'xmlns:'
|| p_response.envelope_tag
|| '="http://schemas.xmlsoap.org/soap/envelope/');

IF (l_fault_node IS NOT NULL)
THEN
l_fault_code :=
l_fault_node.EXTRACT (
'/'
|| p_response.envelope_tag
|| ':Fault/faultcode/child::text()',
'xmlns:'
|| p_response.envelope_tag
|| '="http://schemas.xmlsoap.org/soap/envelope/').getstringval ();
l_fault_string :=
l_fault_node.EXTRACT (
'/'
|| p_response.envelope_tag
|| ':Fault/faultstring/child::text()',
'xmlns:'
|| p_response.envelope_tag
|| '="http://schemas.xmlsoap.org/soap/envelope/').getstringval ();
RAISE_APPLICATION_ERROR (-20000,
l_fault_code || ' - ' || l_fault_string);
END IF;
END;
-- ---------------------------------------------------------------------
FUNCTION invoke (p_request   IN OUT NOCOPY t_request,
p_url      IN          VARCHAR2,
p_action    IN          VARCHAR2)
RETURN t_response
AS
-- ---------------------------------------------------------------------
l_envelope       VARCHAR2 (32767);
l_http_request    UTL_HTTP.req;
l_http_response   UTL_HTTP.resp;
l_response       t_response;
BEGIN
generate_envelope (p_request, l_envelope);
show_envelope (l_envelope, 'Request');
l_http_request := UTL_HTTP.begin_request (p_url, 'POST', 'HTTP/1.1');
UTL_HTTP.set_header (l_http_request, 'Content-Type', 'text/xml');
UTL_HTTP.set_header (l_http_request,
```

```
'Content-Length',
LENGTH (l_envelope));
UTL_HTTP.set_header (l_http_request, 'SOAPAction', p_action);
UTL_HTTP.write_text (l_http_request, l_envelope);
l_http_response := UTL_HTTP.get_response (l_http_request);
UTL_HTTP.read_text (l_http_response, l_envelope);
UTL_HTTP.end_response (l_http_response);
show_envelope (l_envelope, 'Response');
l_response.doc := XMLTYPE.createxml (l_envelope);
l_response.envelope_tag := p_request.envelope_tag;
l_response.doc :=
l_response.doc.EXTRACT
(

'/'
|| l_response.envelope_tag
|| ':Envelope/'
|| l_response.envelope_tag
|| ':Body/child::node()',
'xmlns:'
|| l_response.envelope_tag
|| '="http://schemas.xmlsoap.org/soap/envelope/"');
check_fault (l_response);
RETURN l_response;
END;

-- ---------------------------------------------------------------------
FUNCTION get_return_value (p_response    IN OUT NOCOPY t_response,
p_name       IN          VARCHAR2,
p_namespace   IN          VARCHAR2)
RETURN VARCHAR2
AS

-- ---------------------------------------------------------------------
BEGIN
RETURN p_response.doc.EXTRACT ('//' || p_name || '/child::text()',
p_namespace).getstringval ();

END;
-- ---------------------------------------------------------------------
PROCEDURE debug_on
AS
-- ---------------------------------------------------------------------
BEGIN
g_debug := TRUE;
END;
-- ---------------------------------------------------------------------
```

```
PROCEDURE debug_off
AS
-- ------------------------------------------------------------
BEGIN
g_debug := FALSE;
END;
-- ------------------------------------------------------------
END exchangeMUPNBS.soap_api;
/
SHOW ERRORS
show_envelope(l_envelope, 'Response');
l_response.doc := XMLTYPE.createxml(l_envelope);
l_response.envelope_tag := p_request.envelope_tag;
l_response.doc :=
l_response.doc.extract('/'||l_response.envelope_tag||':Envelope/'||l_response.envelope_tag||':Body/
child::node()',

'xmlns:'||l_response.envelope_tag||'="http://schemas.xmlsoap.org/soap/envelope/"');
check_fault(l_response);
RETURN l_response;
END;
-- ------------------------------------------------------------
FUNCTION get_return_value (p_response    IN OUT NOCOPY t_response,
p_name       IN          VARCHAR2,
p_namespace   IN          VARCHAR2)
RETURN VARCHAR2
AS
-- ------------------------------------------------------------
BEGIN
RETURN p_response.doc.EXTRACT ('//' || p_name || '/child::text()',
p_namespace).getstringval ();

END;
-- ------------------------------------------------------------
PROCEDURE debug_on
AS
-- ------------------------------------------------------------
BEGIN
g_debug := TRUE;
END;
-- ------------------------------------------------------------
PROCEDURE debug_off
AS
-- ------------------------------------------------------------
BEGIN
g_debug := FALSE;
END;
```

```
-- ---------------------------------------------------------------
END exchangeMUPNBS.soap_api;
/
SHOW ERRORS
```

Sam paket jeste dovoljan da bi se ostvarila komunikacija, ali bi kod koji bi programer pisao bio dosta nečitljiv i pun konkatenacije stringova koji predstavljaju *XML* tagove, za samu implementaciju *SOAP* poziva. Zbog toga je *SOAP* na neki način apstrahovan kroz paket *SOAP_API* koji sakriva detalje *SOAP* funkcionalnosti od programera tj. enkapsulirana je *SOAP* funkcionalnost.

Paket se distribuira kao *open source* i može se preuzeti slobodno. Najbitnije metode, funkcije u paketu su:

- *soap_api.new_request* (funkcija)
- *soap_api.add_parameter* (procedura)
- *soap_api.invoke* (funkcija)
- *soap_api.get_return_value* (funkcija)

U odgovarajuće promenljive u praktičnom primeru, potrebno je setovati nekoliko konstanti koje su opisane u samom *WSDL*-u servisa koji je implementiran u praksi. Iz *url*-a servisa dobijamo sledeće parametre:

```
l_url:='http<//SERVER_ADDRESS/NBS_ZinMupExchange_WebService>
l_namespace   := 'xmlns="http://tempuri.org/"';
l_method      := 'GetServiceVersion';
l_soap_action := 'http://tempuri.org/ISendDataToMupService/GetServiceVersion';
l_result_name := 'GetServiceVersionResult';
l_result_name := 'return';
```

Iniciramo poziv pomoću metode i funkcije *soap_api.new_request:*

```
l_request := soap_api.new_request(p_method      => l_method,
                                  p_namespace   => l_namespace);
```

Ukoliko postoje parametri, sledi poziv metode *soap_api.add_parameter,* proporcionalno broju parametara koji je potreban za samu implementaciju:

```
soap_api.add_parameter(p_request => l_request,
                       p_name    => 'int1',
                       p_type    => 'xsd:integer',
                       p_value   => p_int_1);
```

Sledi poziv servisa pomoću *soap_api.invoke* metode:

```
 l_response := soap_api.invoke(p_request => l_request,
                               p_url     => l_url,
                               p_action  => l_soap_action);
 l_return := soap_api.get_return_value(p_response  => l_response, p_name  => l_result_name,
p_namespace => l_namespace)
```

Sa druge strane, kako bi se komunikacija zaštitila, od *Oracle* verzije *9i Release 2,* postojanje paketa *UTL_HTTP* omogućava korišćenje odnosno pozivanje servisa preko *https* protokola (engl. *HyperText Transfer Protoco*l *Secure*) [9] kako bi se omogućio pouzdan prenos podataka uključujući sigurnu identifikaciju servera. Ova funkcionalnost, podržava *PL/SQL* za korišćenje *HTTP request* i *HTTP response* odgovora. Uslovno, ovaj paket omogućava sve funkcionalnosti koje su potrebne da se kreira *http request* kao i samo parsiranje *response*-a. Ranije verzije, nudile su mogućnost pozivanje servisa putem drugih *Oracle utility*-a, poput *UTL_DBWS*, ali koji zbog svoje nepraktičnosti u smislu prevelikog broja *Java* klasa koje je potrebno implementirati (*wrapp*-ovan *JPublisher*) nije pronašao veću primenu u praksi.

Da bi se koristio *https* servis potrebno je prethodno konfigurisati *Oracle Wallet* na *DB* serveru gde će se izvršavati *PL/SQL* kod.

6. Upotreba *Oracle Wallet* rešenja za sigurno (*https*) korišćenje servisa iz *PL/SQL*-a

Oracle Wallet je lozinkom zaštićen kontejner (engl. *container*) koji se koristi za skladištenje kredencijala potrebnih za autentifikaciju i potpisivanje, uključujući privatne ključeve, sertifikate i

sertifikate od poverenja (*trusted certificates*) potrebne za *SSL* komunikaciju (engl. *Secure Socket Layer*). Drugačije rečeno, služi za upravljanje *PKI* (engl. *Public Key Infrastructure*) kredencijalima kako na klijentima, tako i na serverima. Kredencijali se mogu administrirati preko grafičkog okruženja *Oracle Wallet Manager* ili pomoću *orapki* komande operativnog sistema koja je prisutna, tačnije prethodno instalirana na *DB* serveru. Druga upotrebna moć ovog rešenja biće razmatrana u daljem tekstu ovog rada.

Razvojno okruženje, kao što je *Oracle* okruženje, zahteva nekoliko komponenti unutar mehanizma za hijerarhijsku organizaciju izdavanja korisničkih sertifikata. Kao što je napomenuto, počevši od *Oracle DB 9i*, paket pod nazivom *UTL_HTTP* se direktno isporučuje (kao standardni *Oracle Database* paket kao deo *data dictionary*-a) kako bi se funkcionalnost prenosa podataka uz identifikaciju servera uspešno implementirala. Ovakvu kombinaciju tehnologije šifrovanja podataka i servisa koji ujedno omogućavaju međusobnu (sigurnu) komunikaciju unutar Oracle okruženja čine:

- *CA* (engl. *Certificate Authority*) - sertifikaciono (ovlašćeno) telo, brine se o izdavanju sertifikata i validnosti istih kao treća strana od poverenja. Potpis sertfikata moguće je proveriti uz pomoć javnog ključa koji izdaje *CA*, čime se obezbeđuje integritet podataka. Kao centralni deo infrastrukture, ulogu da izdaje, generiše i (samo)poništava sertifikate upotpunjuje ulogom da potpisuje sertifikate svojim privatnim ključem. Proverom potpisa sertifikata utvrđujemo identitet vlasnika javnog/privatnog ključa.

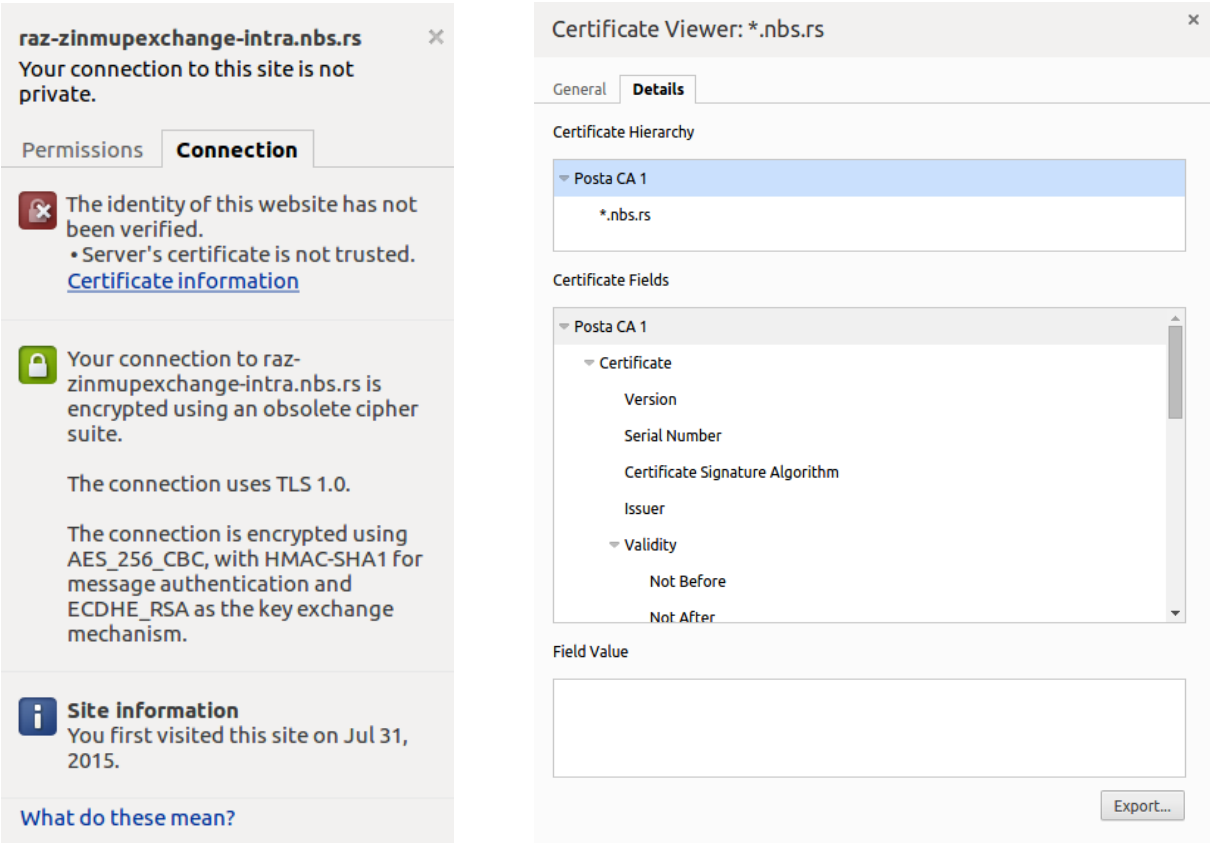
- Sertifikati (engl. *Certificates*) - digitalni sertifikat, potpisan dokument za proveru identiteta učesnika u komunikaciji. Ovako digitalno potpisanim dokumentom, potvrđuje se veza između javnog ključa i korisnika sertifikata. Njegovi sastavni delovi, javni ključ, informacije o vlasniku sertifikata, i digitalni potpis obezbeđuju zadovoljenje trijade sigurnosti bilo kog sistema ali istovremeno i identitet korisnika.

- *CRL* (engl. *Certificate Revocation List*) - binarna datoteka, lista opozvanih sertifikata (entiteti kojima se ujedno više ne veruje) sa razlogom njihovog povlačenja. *CA* periodično osvežava listu ovakvih sertifikata a u trenutku isticaja roka važenja sertifikata, status istog se više ne prikazuje u okviru *CRL* listi. Time se maksimalno minimizuje veličina *CRL* liste, se te smatra da status nema nikakav značaj za sam sertifikat.

U *Oracle* okruženju, svaki entitet koji komunicira preko *SSL*-a mora da ima *wallet* koji sadrži *X.509 ver 3* [10] sertifikata, privatni ključ i listu sertifikata kojima se veruje. Administratori koji se bave bezbednosnim zahtevima, koriste ovakav kontejner za upravljanje kredencijalima na serveru. Vlasnici *walleta* koriste isti za upravljanje kredencijala na klijentima. Globalno govoreći, ovakav vid upravljanja kredencijalima ima sledeće uloge:

- Generisanje para javnih/privatnih ključeva i *request*-a sertifikata
- Skladištenje sertifikata koji se uprauje sa privatnim ključem
- Konfiguracija sertifikata

Da bi se omogućio pristup servisu preko *https*-a potrebno je eksportovati sertifikat sa servera kome se pristupa. To se može uraditi najlakše tako što se u pretraživaču (npr. *Google Chrome browser*) otkuca putanja do servisa, klikne se na ikonu katanca u *url*-u kako bi se prikazao meni sertifikata. Klikom na *"Connection"* tab, a zatim na *"Certificate information"* link, dobijamo *"Details"* tab kao mogućnost eksportovanja sertifikata. Prikaz eksporta sertifikata može se videti na slici ispod. Za svaki čvor (lanac) sertifikata, ukoliko postoji, moguće je sačuvati informacije o samom sertifikatu u tekstualnom formatu.



Slika 5- Eksportovanje *https* sertifikata

Zatim je potrebno kreirati *Oracle wallet* i importovati gore pomenuti sertifikat ili više njih ukoliko postoji veći lanac sertifikata (u primeru je bilo potrebno uvrstiti i *intermediate* sertifikat). Na serveru baze podataka potrebno je prvenstveno kreirati direktorijum gde će se smestiti sertifikat, a zatim pristupamo kreiranju samog *wallet*-a. Specifičnost zahteva bila je da se komunikacija odvija uz korišćenje *SSL* sertifikata koji su izdati od strane suprotnog učesnika u komunikaciji. Taj sertifikat potpisan je svojim privatnim ključem i njega je potrebno uvrstiti u konfiguraciju servera baze podataka.

Povezivanje *Web* servisa u praktičnom slučaju, podrazumevalo je nesto složeniju implementaciju kao i konfiguraciju samog servera. Zahtevi sa strane koja učestvuje u komunikaciji razmene podataka značili su da je postojala potreba da se pristup servisima omogući uz obavezu korišćenja *SSL* sertifikata čiji su oni izdavaoci. Generisanjem *.csr* fajla (engl. *certificate signing request)* kao bloka nekriptovanih podataka o sertifikatu, prethodilo je potpisivanje privatnim

ključem učesnika na suprotnoj strani (privatni ključ *CA*), koji ne zahteva privatni ključ druge strane. Tačnije, tako generisani sertifikat sa određenim .csr fajlom ima svoju upotrebnu moć samo sa privatnim ključem potpisnika. U nastavku sledi opis koraka za smeštanje sertifikata, a korišćena je *orapki* [11] funkcionalnost za upravljanje elementima *PKI* sistema. Ovakva funkcionalnost koju pozivamo iz komandne linije omogućava da se kompletiraju sledeći zadaci:

- Kreiranje (potpisanih) sertifikata
- Upravljanje *Oracle wallet*-om:
    - Kreiranje i prikaz *wallet*-a
    - Dodavanje, brisanje zahteva za sertifikatom (engl. *certificate requests*)
    - Dodavanje, brisanje sertifikata
    - Dodavanje, brisanje sertifikata od poverenja (engl. *trusted certificates*).

Osnovna sintaksa *orapki* funkcionalnosti kroz komandu liniju je:

```
$ orapki module command  -parameter value
```

Vrednost parametra *module* može biti *wallet, crl, ili cert.* dok dalje komande zavise od izabranog modula. Tako da, ukoliko se radi sa *wallet* modulom, moguće izabrane opcije svode se na kreiranje, dodavanje sertifikata ili ključeva. Preostale mogućnosti *orapki* funkcionalnosti mogu se pronaći u zvaničnoj dokumentaciji. [12]

Kreiranje lokacije gde će se smeštati sertifikat:

```
$ mkdir -p <path_to_wallet_dir>
```

Kreiranje samog *wallet*-a vrši se sledećom komandom:

```
[instance@servername ~]$ orapki wallet create -wallet <path_to_wallet_dir> -pwd <wallet_pass> -auto_login

Oracle PKI Tool : Version 11.2.0.2.0 – Production Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
```

Ukoliko su kredencijali neodgovarajući, javlja se sledeća poruka:

Invalid password....
PASSWORD_POLICY : Passwords must have a minimum length of eight characters and contain alphabetic characters combined with numbers or special characters.

Informacije o novokreiranom *wallet*-u dobijamo:

[*instance@servername ~*]$ *--orapki wallet1 display -wallet <path_to_wallet_dir> -pwd <wallet_pass> -auto_login*

Informacije o samom sertifikatu, koji je inicijalno dobijen u ***.cer*** formatu, pre konverzije u odgovarajući ***.pem*** format (zbog potreba da server baze podataka bude na *Linux* operativnom sistemu, koji podržava ovakav tip sertifikata) dobijene su izvršavanjem sledećih komandi:

$ openssl x509 -inform der -in MupZin.cer -out MupZin.pem

$ openssl x509 -inform pem -in MupZin.pem -noout –text

Sledi faza testiranja sertifikata *wget* komandom, kako bi otklonili moguće probleme neuspešnog pristupa servisa na suprotnoj strani:

$ wget --private-key=/etc/pki/tls/private/mup.key --certificate=MupZin.pem --certificate-type=PEM --debug --no-check-certificate https://10.16.158.3:7010/MUP/ZinNbsPrijem?wsdl

Opcija *wget* omogućava da sakupimo potrebne informacije o sadržaju servera korišćenjem besplatnog *utility*-a (preko *HTTP*, *HTTPS*, i *FTP* protokola), dok je opcija *--no-check-certificate* omogućila ignorisanje neverifikovanih sertifikata. Verzija *Wget1.10* komande, u osnovi podrazumeva verifikaciju serverskog sertifikata u odnosu na *CA*, tako što razbija *SSL handshake* protokol i u slučaju da verifikacija ne prođe, uništava preuzimanje sertifikata. Iako ovakav način značajno osigurava komunikaciju i obezbeđuje sigurnije preuzimanje sadržaja, utiče na interoperabilnost sa starijim verzijama *wget* komande. Ukoliko se prilikom testiranja sertifikata javi upozorenje *"common name doesn't match requested host name"* to bi značilo da je moguće zaobići verifikaciju i nastaviti preuzimanje potrebnog sadržaja. Ova opcija uključuje se samo pod

uslovom da se veruje u autentičnost (sajta) odnosno ukoliko nam nije od važnosti validnost njegovog uverenja, ali nije preporučljivo za prenos poverenih informacija. Opciono, moguće je uključiti i komandu *–check-certificate=quiet* kako bismo zaobišli da *wget* štampa bilo kakvo upozorenje o nevažećim sertifikatima.

--Private-key opcija omogućila je da se isčitava privatni ključ iz fajla*;* opcija *--certificate-type* specifikuje tip klijentskog sertifikata *(.pem, .der),* dok opcija *debug* podrazumeva sigurniju podršku developerima u radu (pri kompajliranju grešaka).

Prebacimo prethodno sačuvan i exportovani sertifikat u direktorijum *<path_to_exported_cert_dir>* i dodamo ga u direktorijum gde se nalazi *wallet* sledećom komandom:

```
[instance@servername ~]$ orapki wallet add -wallet <path_to_wallet_dir>  -trusted_cert -cert <path_to_exported_cert_dir>/exported_cert_name"  -pwd <wallet_pass>

Oracle PKI Tool : Version 11.2.0.2.0 - Production

Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.

č[instance@servername ~]$ orapki wallet export -wallet <path_to_wallet_dir> -dn 'CN= <ip adress>,OU=IT-TEST,O=NBS-ZIN,L=Beograd,C=RS' -request <path_to_exported_cert_dir>/ -pwd  <wallet_pass>
```

Kod za pozivanje *https* servisa ostaje isti, samo je potrebno u kodu "setovati *wallet*" pomoću komande:

```
UTL_HTTP.set_wallet ('file: <path_to_wallet_dir> , <wallet_pass>);
```

Od verzije *Oracle 11g* potrebno je u nekim slučajevima definisati *ACL* [13] listu kako bi se dozvolilo *UTL_HTTP* paketu da komunicira sa eksternim *host*-om bez potrebe za dodatnim podešavanjima privilegija u samoj bazi. U prethodnim verzijama, pristup eksternim servisima omogućavao se pravima pristupa- grantovanim permisijama (*execute* privilegija).

6.1. *Oracle Wallet Manager*

Kao što je napomenuto, drugi način upravljanja kredencijalima i njihova administracija moguća je preko grafičkog (aplikativnog) okruženja *Oracle Wallet Manager*-a (Slika 6- *Oracle Wallet Manager*). U svojoj osnovi, predstavlja aplikaciju koju vlasnici *wallet*-a koriste za (bezbedniju) administraciju akreditiva, a ima ulogu da skladišti *X.509* sertifikate i privatne ključeve u *PKCS#12* formatu, kao i zahteve prema *PKCS#10* specifikaciji. Ovakve mogućnosti upravo čine interoperabilnu strukturu *wallet*-a koji može biti podržan od strane *third-part PKI* aplikacija i prenosivost putem operativnog sistema. Kreirani *wallet* može biti upotrebljen u samoj *Oracle* bazi, na *Oracle* aplikativnim serverima kao i *Oracle Identity Management* infrastrukturi. Pored toga, njegove osnove funkcionalnosti su:

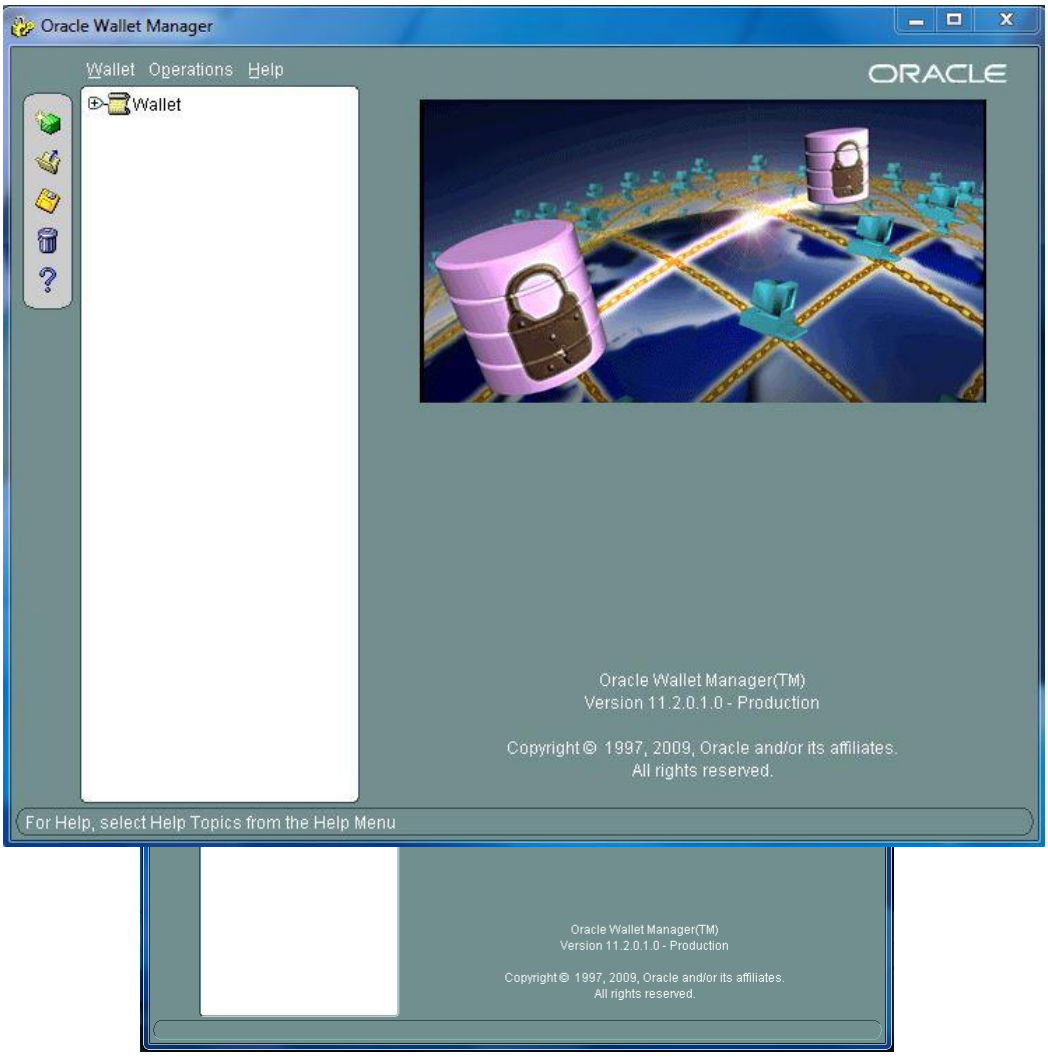
Slika 6- *Oracle Wallet Manager*

- Kreiranje *wallet*-a
- Generisanje zahteva za sertifikatom;
- Otvaranje *wallet*-a radi pristupa *PKI* servisima;
- Čuvanje kredencijala na kriptografsko-hardverskim uređajima, poput pametnih kartica (engl.*smart cards*) koristeći mnogobrojna *API* rešenja;
- Importovanje, eksportovanje *wallet*-a od strane nezavisnih (engls.*third-party*) okruženja;

U nastavku teksta biće prikazan način generisanja zahteva za sertifikatom.

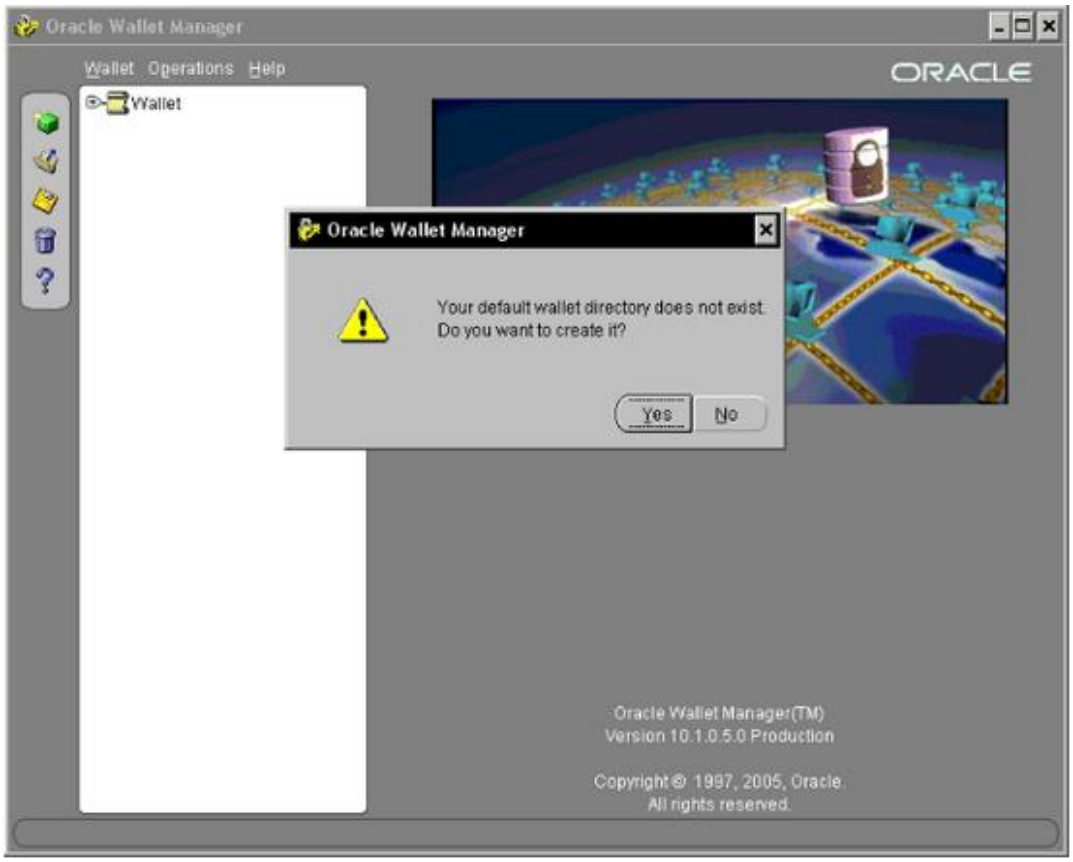
Na slici (Slika 7- *Wallet Manager*: otvaranje novog *wallet*-a) je prikazano inicijalno grafičko

okruženje prilikom pokretanja same aplikacije. Otvaranjem taba *Wallet* N*ew,* dobijamo prozor za kreiranje. Ukoliko se prvi put kreira *wallet*, njegov inicijalni fajl zapravo ne postoji. Potrebno je kreirati inicijalni fajl.



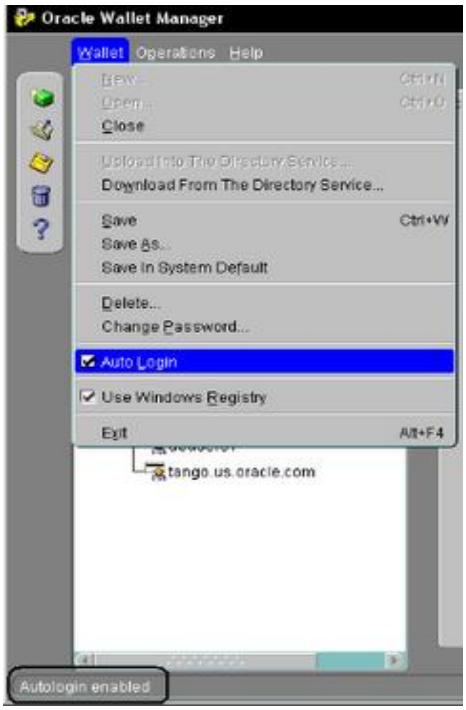
Slika 7- *Wallet Manager*: otvaranje novog *wallet*-a

Wallet je fizički smešten u određenom direktorijumu, ali ukoliko postoji potreba, korisnik može definisati putanju gde želi da bude kreiran. Takođe, korisnik opciono može da definiše *default*-nu putanju.



Slika 8- ***Wallet directory***

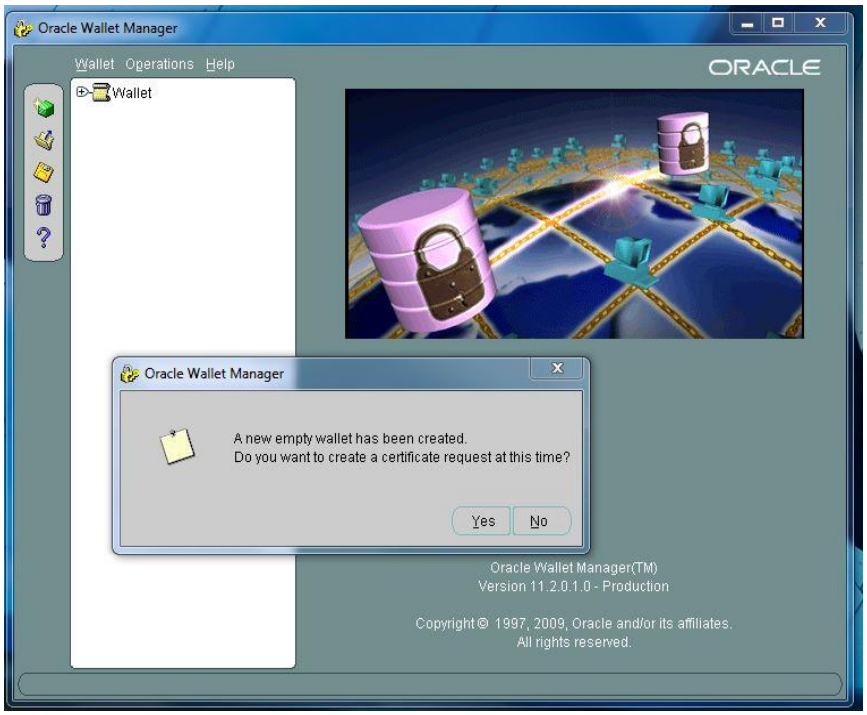
Fajl pod nazivom *ewallet.p12* biće kreiran na specifičnoj lokaciji. Opcija *oracle-ovog* 'novčanik'-a, *Auto Login* (Slika 9- *Auto Login, Windows Registry*) stvara prividno zamućenu (engl. *obfuscated copy*) kopiju novčanika i time omogućava *PKI* zasnovan pristup servisima bez korišćenja lozinke, iako ovo neće biti korišćena opcija. Kada je uključena ova opcija, odnosi se samo na korisnika operativnog sistema koji se kreirao taj 'novčanik'. Još jedna od mogućih opcija jeste uključivanje *SSO* (engl. *single-sign-on*) za pristup drugoj bazi podataka. Ove dve opcije su usko povezane, pa sa uključivanjem prethodne opcije, mogli bismo omogućiti pristup drugim bazama putem *SSO*-a.



Slika 9- ***Auto Login, Windows Registry***

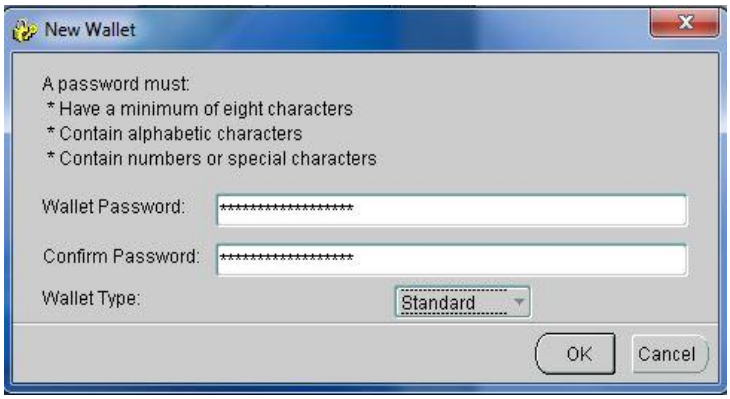
Naredna slika prikazuje definisanje parametara za pristup *wallet*-u, preko definisane šifre. Politika upravljanja kredencijalima, i ovde važi a osigurava kreiraje pasvorda minimalne dužine od 8 karaktera, sa preporučenim kombinavanjem alfanumeričkih karaktera.

Kako *Oracle wallet* sadrži kredencijale koji se koriste za autentifikaciju korisnika nad više baza, od velike važnosti je izabrati jaku lozinku za pristup. Ukoliko u ruke malicioznog korisnika dođu pristupni parametri samog *walleta*, u tom slučaju, omogućen je pristup svim bazama kojima legitimni korisnik ima pristup.



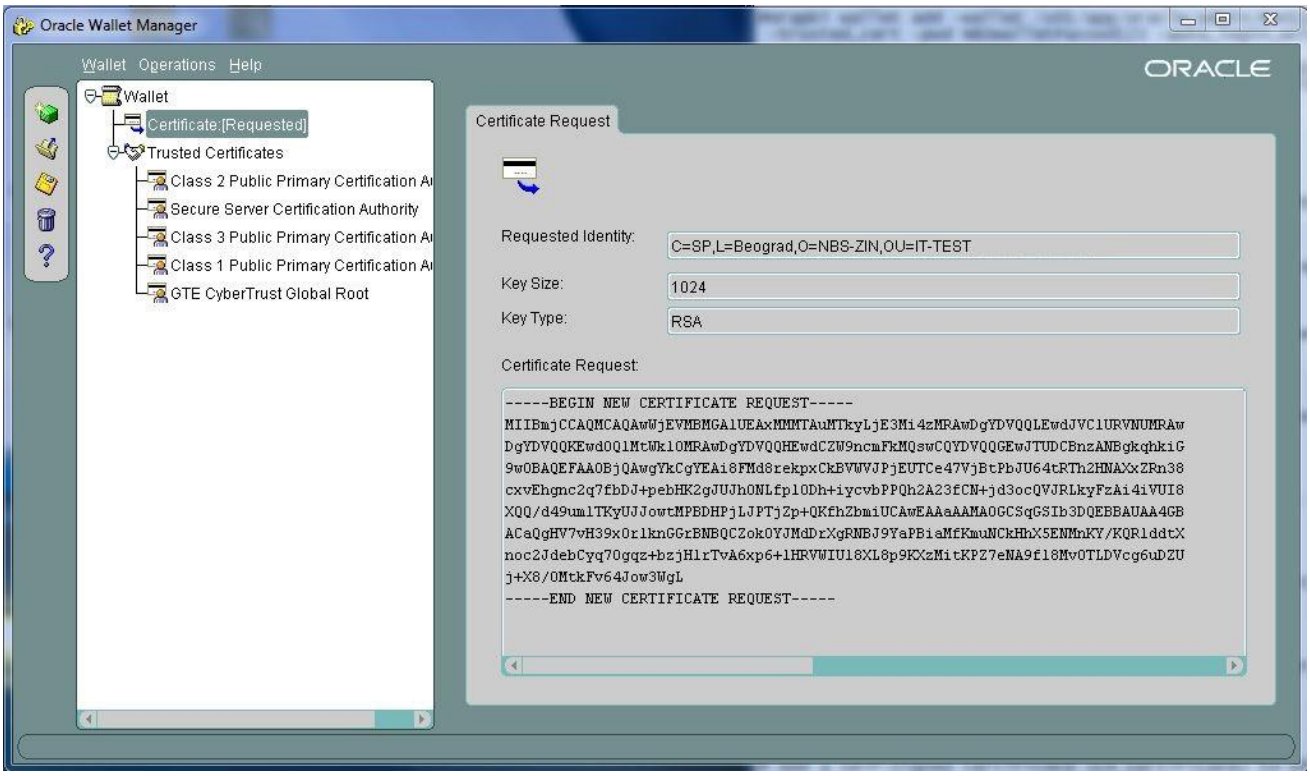
Slika 10- Podešavanje lozinke

Nakon podešavanja samog pristupa, kreiran je prazan *wallet*.



Slika 11- Prihvatanje novih kredencijala

Dalje, potrebno je generisati *request* za sam sertifikat (engl. *certificate request*). Ovaj zatev, drugačije se naziva *CSR*, i princip rada je identičan kao i kod kreiranja *wallet*-a na operativnom sistemu tačnije serveru baze podataka. Osnovne komponente su prikazane na slici (Slika 12- Komponente *CSR*-a) :



Slika 12- Komponente *CSR*-a

1. ***Common Name***: Nazivi domena koje želimo da osiguramo. Obavezno je polje.
2. ***Organizational Unit***: Naziv organizacione jedinice. Opciono je polje.
3. ***Organization***: Puni zvanični naziv organizacije. Opciono je polje.
4. ***Locality/City***: Grad/lokacija gde je organizacija smeštena. Opciono je polje.
5. ***Country***: Zemlja gde je organizacija smeštena. Obavezno je polje.
6. ***Key Size***: Odabir veličine ključa koji će se koristiti za kreiranje para javnog/privatnog ključa (*2048bit* je preporučena veličina ključa). Obavezno je polje. *CA* obično koristi *1024/2048 bitne* ključeve, a u slučaju dužeg čuvanja ključeva od strane vlasnika sertifikata, biraju se ključevi većih dužina (*3072, 4096 bit*).

Nakon što je uspešno kreiran, potrebno je sačuvati ga. Za primer kreiranja *wallet-a* preko aplikacije, korišćeno je smeštanje na *Windows file management system* ili *Microsoft Windows system registry*-a, kao centralizovane konfiguracione baze koja sadrži brojne parametre u vezi sa

operativnim sistemom. Čuvanje novčanika u registru, pruža mnoge pogodnosti, a pored lakše administracije, prednost je svakako bolja kontrola pristupa, jer kao što je poznato, oblasti *user profile area* unutar registra može pristupiti samo 'konektovan' korisnik. To bi značilo da ukoliko se korisnik isključi sa sistema na koji je logovan, automatski pristup tom novčaniku je isključen, onemogućen.

6.2. Upotreba *Apache SSL proxy* servera

U primeru iz prakse koji se koristio kao osnova za ovaj master rad, kreiran je *SSL Reverse Proxy* server. Korišćen je *Apache* kao *SSL proxy* server, u kome je definisan i dodat novi virtuelni host u okviru *httpd* konfiguracionog fajla. Imenovan je internim nazivom ***ProxyS***, a kako ne postoji kao takav u *DNS*-u (engl. *Domain name system*), unet je zapis u */etc/hosts* fajl.

Detaljnija implementacija ovakvog servera iz praktičnog primera neće biti tema ovog master rada, već će se obrađivati principi, kao i osnovni prikaz upotrebe ovakvog servera u cilju sticanja potrebnih informacija o načinu i složenosti ovakvog okruženja.

Osnovni cilj postojanja ovakvog servera i njegove konfiguracije jeste obezediti pristupačnost internih resursa kompanije eksternim korisnicima. Jednostavnije rečeno, ovakav tip servera ima ulogu posrednika između korisnika (udaljena lokacija ili interno) i Interneta, a najčešća upotreba je u vidu klasičnog filtera sadržaja (preuzimanje sadržaja koji se nalazi na Internetu sa tog *proxy* servera- engl. *content filtering*). Korišćenjem klasičnih *proxy* servera, u nešto drugačijem kontekstu, dobija se dodatni nivo smanjenja opterećenosti saobraćaja (engl. *load balancing*), zatim privremeno čuvanje sadržaja koji je potreban (engl. *cache podataka*) kao i prikrivanje stvarnih informacija o internoj strukturi organizacije. Mogu se primenjivati uz mnogobrojne druge Internet protokole (*SSL, FTP, SMTP*), ali je pravilo funkcionisanja skoro identično za svaki od njih - klijenti ne moraju biti svesni njegovog postojanja, a samim tim mogu biti smešteni u demilitarizovanoj zoni (po potrebi) interne mreže. U takvoj situaciji, internim resursima kompanije uvodi se dodatni nivo zaštite od neovlašćenih aktivnosti (spoljnih) korsinika koji im pristupaju sa Interneta, jer upravo oni pristupaju resursima samo preko *proxy* servera.

U praktičnom primeru implementiranog reverznog *SSL proxy* servera, primarni cilj bio je omogućiti prenos podataka putem autentifikacije korisnika (upotreba sertifikata), gde glavnu ulogu ima sam server, kao centralizovana tačka pristupa internim resursima, dok se prenos podataka vršio putem RSA algoritma, čime su se dobili slični rezultati kao da jek orišćena VPN tehnologija (engl. *Virtual Private Network*). Zbog ozbiljnosti samog projekta koji je rađen, upotreba ovakvog servera našla je svoju primenu promenom zapisa u *DNS* imenima. U slučaju kvara internih servera (server na kome je smeštena baza podataka), ili njegovoj promeni *DNS* imena, upotrebljen je *proxy* server kao privremeno rešenje. Kao takvo, pruža mnogo lakšu, bržu i novu konfiguraciju na samom proxy serveru, dok se nastali problemi ne uklone. Međutim, vrlo česta pojava je problem u radu samog proxy servera, pri čemu se to reflektuje na samoj nedostupnosti svih servisa na njemu, pri čemu se u ovakvim situacijama vodi računa da postoji dodatni, redudantni server koji će moći automatski da preuzme ulogu celog sistema. Pored toga, još jedan od mogućih problema nesigurne implementacije ovakvih servera je nedovoljno zaštićen sam *Web* servis. Širenjem napada preko *HTTP* protokola interni resursi neće biti imuni čak i sa pravilno konfigurisanim rešenjem proxy servera. U ovoj situaciji, predlog za sigurniji prenos podataka bio bilo bi korišćenje analize sadržaja paketa na mreži (engl.*content filtering*) čime bi se uveo dodatni nivo zaštite i podrške analizi sadržaja koji se prenosi. Tobi značilo da se lako može napraviti razlika (ne)legitimnih korisnika kao i dozvoliti prenos podataka prema internoj organizaciji.

U cilju zaštite internog servisa (servera), prvi korak ka implementaciji reverznog servera jeste statičko prevođenje adresa (engl. *Network Address Translation, NAT*) na internom *firewall* uređaju, na sledeći način:

- adresa *babeldoc-new* servera *(SSL Reverse Proxy* server) (1.2.3.4) *NAT* – ovana je u 5.6.7.8 *IP* adresu, a upravo je ovo adresa koja je postavljena prilikom generisanja sopstvenog *.csr* fajla (na novou operativnog sistema) potpisanog privatnim ključem suprotne strane u komunikaciji. Prevođenje adresa je bilo preporučljivo i zbog što manjeg broja tačaka pristupa servisu drugog učesnika u komunikaciji.

  CN=5.6.7.8,OU=IT-BABELDOC,O=NBS-ZIN,L=Beograd,C=RS

Reverznim mapiranjem adresa, u realnosti se prikrivaju stvarne adrese internih servera, a time i sakrivanje informacija o internoj infrastrukturi nekog sistema. Koristeći osnovna pravila *reverse mapping* tehnike, definisani proxy server će imati ulogu presretača odgovora internih servera, modifikovati ih tako da klijenti stiču utisak da odgovor dolazi sa stvarih servera u internoj mreži a ne direktno sa *proxy*-a. To bi u praksi značilo da će server posrednik u elementima *HTTP, HTTPS* odgovora adresu unutar izvora paketa komunikacije zameniti sa sopstvenom *IP* adresom (preusmerenje). Isto pravilo važi i u suprotnom smreu- kako bi klijenti toj novododeljenoj adresi pristupali preko posrednika, upotrebom istog reverznog pravila, modifikuje se odgovor i preusmerava se na sopstvenu *IP* adresu.

Implementacija *proxy* servera nudi brojne mogućnosti ali se mora voditi računa o nekim aspektima koji mogu direktno uticati na sigurnost samog sistema i razmeni podataka. Nedovršena i nezaštićena implementacija ovakvog servera može samo biti posledica nepravilno definisanih pravila *NAT*-ovanja i pogrešne sigurnosne politike mrežnih *firewall* uređaja. Preporuka je svakako korišćenje statičkih adresa za javne servere kako bi se na taj način smanjilo otvaranje informacija o unutrašnjoj organizaciji neke kompanije kao i raspoloživost poverljivih informacija. Druga preporuka jeste dozvola *HTTP* i *HTTPS* konekcija sa Interneta ali samo na *proxy* serverima (u *DMZ* zoni), sa bitnom napomenom da ovo pravilo važi i u suprotnom smeru- dozvolu *HTTP, HTTPS* konekcije sa *proxy* servera prema internim serverima organizacije. Uključivanjem podrške za *SSL* protokol, omogućavamo poverljivost razmene podataka između servera i klijenata, pa u tom slučaju vršimo enkrpciju podataka između servera i udaljenih klijenata , ali istovremeno i autentifikaciju pomoću sertifikata. Ovako implementirana arhitektura štiti mrežni saobraćaj koji putuje od klijenata do servera putem Interneta.

Kreiran je novi virtuelni host na *Apache* serveru, a konfigurisanje tog hosta prikazano je u nastavku.

Konfiguracija u Apache:

```
<VirtualHost NbsMupProxy:80>

SSLProxyEngine on

SSLProxyCACertificateFile /etc/pki/tls/certs/MUPCACerts.pem

####SSLProxyVerify optional_no_ca   -- linija za prihvatanje neverifikovanih certifikata
```

```
SSLProxyVerifyDepth  2
SSLProxyMachineCertificateFile /etc/pki/tls/certs/ClientMUP.pem
<Proxy *>
</Proxy>
<Location /mupzinfo>
ProxyPass  https://10.192.192.3:7010
ProxyPassReverse https://10.192.192.3:7010
###ProxyPass  https://10.16.158.3:7010
###ProxyPassReverse https://10.16.158.3:7010
</Location>
</VirtualHost>
```

Tehnologija reverznog *proxy* servera, zajedno sa svim svojim prednostima i nedostacima, našla je svoju primenu u praktičnom primeru povezivanja *Web* servisa dva produkciona okruženja različitih kompanija.

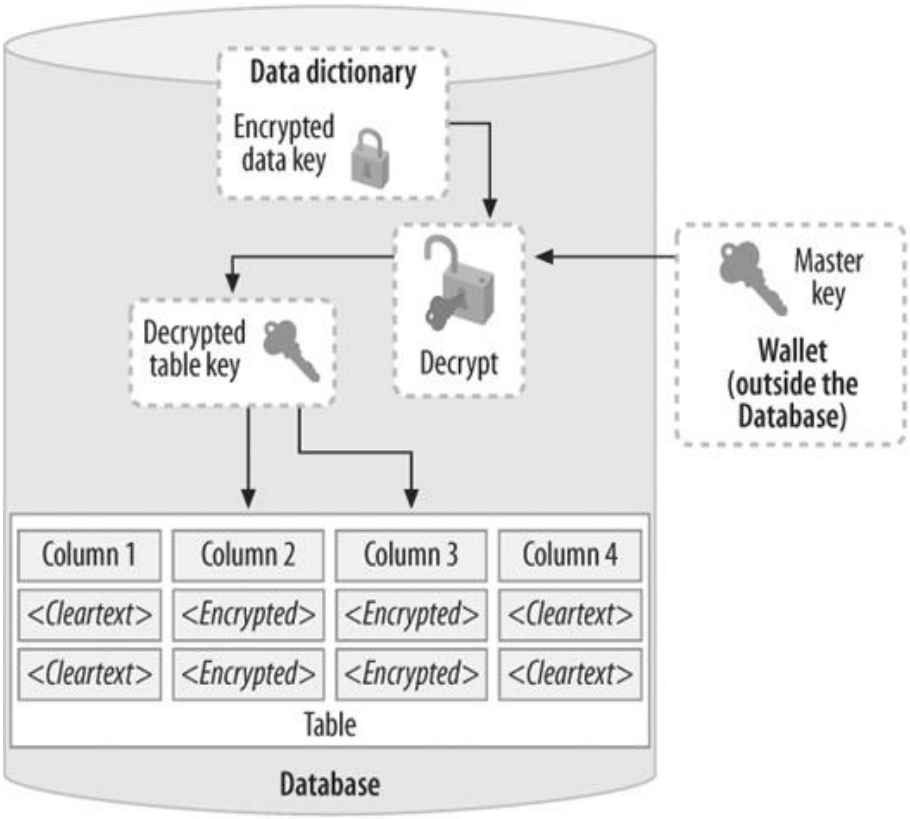
7. *Oracle Wallet* rešenje za transparentnu enkripciju podataka- *TDE*

Drugim rečima, *wallet* nije ništa drugo nego logički zaštićen kontejner (*ewallet.p12* fajl) koji ima ulogu da skladišti ključeve potrebne *Oracle* bazi za pristup podacima na *SSL* sajtovima. Svakako, ovo nije jedinstvena uloga, koristi se i u slučaju transparentne enkripcije podataka *TDE* (engl. *Transparent Data Encryption*). Radi upoznavanja, *TDE* predstavlja sastavni deo *Oracle Advanced Security Option (ASO)* i dostupan je samo u *Oracle Enterprise Edition* verziji.

Oracle 11g baza koristi metode autentifikacije, autorizacije i *auditing* mehanizme za zaštitu podataka unutar baze, ali ne i na *data* fajlovima na operativnom sistemu. Mnogobrojna bezbedonosna rešenja u relacionim bazama dostupna su godinama unazad, ali ni jedan od njih do sada nije predstavljao '*out-of-the-box'* metod za zaštitu podataka na nivou operativnog sistema.

U situacijama kada se ključevi za enkripciju i sami podaci skladište u samoj bazi podataka, otvara se put ka drugoj potencijalnoj pretnji po sigurnost tih podataka- ukoliko dođe do kompromitovanja (krađe) diska na kome su smeštene sve osetljive i informacije od značaja, sama baza postaje ranjiva. Čak i u situacijama kada je baza potpuno izolovana, potreba za zaštitom podataka ne jenjava. Rešenje koje se nameće jeste smeštanje podataka i ključeva na različitim lokacijama (odvajanje lokacija gde su smešteni podaci na diskovima i svih potrebnih elemenata za enkripciju- ključeva), a to se postiže upravo ovom funkcionalnošću, *TDE*- om. Drugačije rečeno, glavni cilj jeste osigurati fizičke fajlove baze, poput transakcionih log fajlova, ili *backup* fajlova. Iako ova funkcionalnost nije primarna tema ovog rada, cilj ovog poglavlja je ukratko upoznavanje sa generalnim konceptom zaštite osetljivih podataka uskladištenim na trakama ili diskovima kroz *Oracle wallet.* Ovaj koncept strogo vodi računa da podaci budu smešteni u šifrovanoj formi, pa prilikom pristupa podacima, oni se automatski dešifruju transparentno, tao da ne zahteva ni jednu liniju programskog koda određene aplikacije ili koda unutar same baze.

Princip rada, vidi se sa slike ispod (Slika 13- Upotreba ključeva u *TDE* modelu). Ovakav model koristi kombinaciju dve vrste ključeva: ključ koji se generiše za svaku tabelu koju želimo da šifrujemo/dešifrujemo odnosno kolone i *tablespace*-ove (*column encryption key*) i smešen je u enkriptovanoj formi u *dictionary* tabeli svih tabela (unutar same baze), kao i master ključ (*master key*), serverski ključ koji se smešta van baze podataka, i to u samom *wallet*-u, i njime se šifruju/dešifruju prvi ključevi kako bi u bazi mogli da perzistiraju u enkriptovanom obliku. Ključevi nisu smešteni u otvorenom obliku, *cleartext*-u, a isti ključ (za svaku tabelu) se koristi za sve redove u tabeli. Naime, kada tabela sadrži šifrovane kolone, isti jedinstveni ključ se koristi bez obzira na broj tih šifrovanih kolona. Svi ključevi kojima se prethodno šifruju tabele, zajednički su šifrovani master ključem, i nalaze se u samom *dictionary*-u, dok je master ključ smešten van dometa baze podataka, u eksternom modulu. Sve što je potrebno uraditi jeste izabrati kolone koje želimo da šifrujemo, tada *Oracle* generiše ključ za tu tabelu i smešta ga u *data dictionary*. Ovaj ključ se istovremene šifruje master ključem. Može biti softverski ili hardveski. U skladu sa time, *TDE* nam omogućava da definišemo onaj set podataka (kolone) koje želimo da šifrujemo, tako da preostali nešifrovani podaci ostaju, na disku, u svom izvornom obliku (engl. *cleartext*).



Slika 13- Upotreba ključeva u *TDE* modelu

Kada korisnik upisuje podatke u šifrovane kolone, ti podaci automatski postaju šifrovani od strane *TDE*-a. Ukoliko korisnik izabere šifrovane kolone podataka (*select* ili *insert* upitom), *Oracle* server pokušava da transparentno preuzima ključ šifrovane tabele iz *dictionary*-a zajedno sa master ključem iz kontejnera, dešifruje kolone te ih prikazuje korisniku u otvorenom obliku. Ukoliko su podaci na disku kompromitovani (ukradeni), podaci se ne mogu prikazati korisniku bez ključa, koji je smešten u *wallet*-u, šifrovanom tako master ključem. Rezultat toga je da podaci ne mogu da se dešifruju čak i ako se isti kompromituju sa diskova ili sa kopije fajlova. Algoritam šifrovanja podataka koji koristi *Oracle* baza uključuje *AES* (engl. *Advanced Encryption Standard*) algoritam, simetrični algoritam koji prilikom šifrovanja koristi otvoreni tekst kao ulazni parametar u algoritam ali i ključ za šifrovanje. U suprotnom smeru, isti algoritam se koristi i za dešiforvanje podataka, koristeći isti ključ. Ovaj ključ smešten je unutar baze, ali je ujedno šifrovan master ključem.

Prilikom definisanja kolona koje želimo šifrujemo, definišemo klauzulu *ENCRYPT USING* kojom presrećemo tekstualne vrednosti u otvorenom obliku, šifrujemo ih, a zatim skladištimo tako šifrovani format podataka. Kada korisnik uzima podatke iz tabela, vrednost kolona je transparentno dešifrovana. Pre samog korišćenja *TDE*, potrebno je podesiti a zatim kreirati *wallet* gde će se smeštati serverski *master key* .

Komande kojima se to obezbeđuje date su u nastavku. Uobičajena lokacija za smeštanje walleta je direktorijum *$ORACLE_BASE/admin/$ORACLE_SID/wallet*, ali može biti promenljiva, tada se njeno podešavanje vrši se putem parametarskog fajla *SQLNET.ORA.*

```
$ cd /u01/app/oracle/network/admin

$ mkdir tde_wall

$ nano sqlnet.ora

ENCRYPTION_WALLET_LOCATION =

 (SOURCE=

   (METHOD=file)

    (METHOD_DATA=

(DIRECTORY=/oracle_wallet)) -- (DIRECTORY=/u01/app/oracle/wallet_location…)
```

Sledi faza definisanja kredencijalna za pristup *walletu*, a obezbeđuje se *alter* naredbom unutar samog sistema za upravljanje podacima.

```
$sqlplus / as sysdba

ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY <password>;
```

Ovakva komanda daje mogućnost indirektnog kreiranja *walleta* na prethodno navedenoj lokaciji, uz kreiranje lozinke za pristup. Bez lozinke, master ključ ali i drugi ključevi ne mogu biti korišćeni za šifrovanje/dešiforvanje podataka.

TDE mehanizam podržava nekoliko algoritama za šifrovanje- *AES128, AES192, AES256, 3DES168,* a češće korišćena opcija je enkripcija čitavog *tablespace*-a. Ukoliko se šifruju *tablespace*-ovi, svi objekti unutar tog prostora biće šifrovani po automatizmu.

```
CREATE TABLESPACE test_tbs
DATAFILE '/u01/app/oracle/oradata/ora11g/test_tbs01.dbf' SIZE 20m ATOEXTEND ON NEXT5m
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO
ENCRYPTION USING '3DES168'
DEFAULT STORAGE (ENCRYPT);
```

Ova funkcionalnost ima minimalni uticaj na performanse baze, ali ne i ukoliko govorimo o dvostrukoj enkripciji- kolone i prostor (*tablespace*) istovremeno znaju činjenicu da dupla enkripcija znatno usporava performance sistema. Iako pruža brojne prednosti, postoje ograničenja sa kojima se susrećemo prilikom rada sa ovakvim vidom zaštite podataka. Važno je napomenuti da ne postoji mogućnost omogućiti ovakav vid enkripcije za objekte, tabele u *SYS* vlasništvu, a takođe, ne postoji mogućnost korišćenja *TDE* u kolonama koje su *BLOB*, *CLOB* tipa.

Podaci iz tabela su transparentno dekriptovani za korisnike, i ne zahtevaju ni jedan vid dodatnih akcija poput kreiranja *trigger*-a ili *view*-a za dešifrovanje istih. Time se postiže razdvajanje dužnosti, delegiranje odgovornosti klasičnih *db* administratora i administratora sigurnosti sistema. U tom slučaju, *db* administratoru je nepoznat pasvord *wallet*-a, i zahteva od administratora sigurnosti da ga obezbedi. Još jedan od benefita, jeste činjenica da aplikacije ne moraju biti modifikovane kako bi mogle da koriste šifrovane podatke-celokupno šifrovanje vrši se u sklopu baze. Poslednjih nekoliko godina, termin *Oracle walleta* često se usko vezuje za jedan novi pravac poznatiji kao *Secure External Password Store.* Kredencijali za pristup bazi podataka sada mogu biti smešteni na klijentskoj strani, unutar samog wallet-a, tog zaštićenog

kontejnera koji se koristi za skladištenje kredencijala i za sigurnu autentifikaciju. Aplikacijski kodovi, serija *batch* job-ova, skriptovi više nemaju upotrebu za ugrađenim korisničkim imenima i lozinkama za izvršenje. Time se smanjuje potencijalni nivo rizika, jer upravo kredencijali nisu prisutni u otvorenom obliku, čime se znatno smanjuju suštinske promene u kodu, ali i radu aplikacija, ukoliko se u nekom trenutku promeni korisnik koji pristupa bazi.

8. Zaključak

Globalno širenje ideje o sve raspodeljenijoj obradi podataka, a samim tim i deljenju funkcionalnosti sa drugim organizacijama proizvelo je jedan potpuno novi način projektovanja informacionih sistema. Vremenom, takve organizacije nisu ostale usmerene samo na sopstvena rešenja i aplikacije i time unapredile sopstveno poslovanje. Suprotno od toga, novim načinom i metodologijom razvoja sistema, poslovanje postaje otvoreno zajednici *IT* stručnjaka- dostupno široj javnosti. Najveće zasluge leže upravo u samim karatkeristikama servisa- sklalabilnost, interakcija između komponenata, nezavisnost i uopštenost interfejsa, postojanje komponenti posrednika koje povećavaju sigurnost time što smanjuju kašnjenje prilikom interakcije.

Mnoge poslovne funkcionalnosti, poput internet bankarstva, do sada mogle su da se obavljaju isključivo putem jedinstvenih web aplikacija. Takvo ograničenje, korišćenje samo jedne aplikacije u mnogome je usporavalo dalji razvoj i korišćenje usluga. Nastala je uzročno posledična veza - veća prilagodljivost jedne organizacije svojih usluga osim što je povećala reputaciju iste, istovremeno je omogućila nekim drugim organizacijama povećanje njihovih usluga. Korist je obostrana, a korisniku se pruža bolje iskustvo izlaganjem funkcionalnosti u obliku web servisa kao sada već obaveznim delom u svakodnevnom poslovanju. Porast upotrebe generalnog koncepta servisa povećao je potrebu za ispitivanjem postojanja sigurnosti za istim. Sigurnosne ranjivosti sada imaju veliki uticaj i na poslovanje, međutim susrećemo se sa problemom nepostojanja jedinstvene metodologije ispitivanja sigurnosti koja bi mogla da obuhvati sve moguće scenarije. Ispitivanje sigurnosti postaje mnogo složenije čak i u poređenju sa web aplikacijama, a tome ne doprinosi ni sve veći broj novih tehnologija i standarda.

Iako je *PL/SQL* inicijalno zamišljen kao jezik za manipulisanje podacima i izvršavanje upita nad podacima u sistemima za upravljanje istim, zahvaljujući *UTL_HTTP* paketu, moguće je na jednostavan način pozivati web servise. Inicijalna ali i jednokratna konfiguracija samog *DB* servera na kome su smešteni podaci, enkapsulacijom *SOAP* protokola u paket *SOAP_API*, omogućila je relativno jednostavan posao programerima. Pored toga, sigurnost je podignuta na viši nivo korišćenjem jedinstvenog kontejnera *Oracle Walleta*, za enkripciju podataka na disku. Pored upotrebe ovog modula u praktičnom primeru, njegovo kreiranje može biti od značajne

pomoći u cilju sprečavanja *hardcode*-ovanih kredencijala u svakodnevnim (*backup)* procesima baze podataka. Korišćenje *walleta*, kao *Secure External Password Store* koncepta na jednostavniji i sigurniji način povećavamo sigurnost baze podataka sa ciljem izbegavanja korišćenja lozinki u otvorenom tekstu u svakodnevnim skriptovima *Oracle* administratora i programera.

Literatura:

[1] W. D. B. W. F. /. H.-P. Hugo Haas, „Web Services Glossary,“ 2003.

[2] M. Kalin, Java Web Services: Up and Running, 2nd Edition, 2nd Edition ur., O'Reilly Media, 2013.

[3] „SOAP Version 1.2 Part 1: Messaging Framework (Second Edition),“ W3C.

[4] A. J. Mladen Veinović, US - Računarske mreže, Beograd: Univerzitet Singidunum, Beograd, p. 293.

[5] „Understanding SOAP,“ Microsoft, [Na mreži]. Available: https://msdn.microsoft.com/en-us/library/ms995800.aspx. [Poslednji pristup 2016].

[6] „WS-I Trademarks and Compliance claims requirements“.

[7] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code, John Wiley & Sons, Inc, 1996.

[8] T. Hall. [Na mreži]. Available: support.oracle.com, oracle-base.com.

[9] „Oracle UTL_HTTP,“ [Na mreži]. Available: http://psoug.org/reference/utl_http.html.

[10] W. F. W. P. D. S. R. Housley, „Internet X.509 Public Key Infrastructure,“ January 1999. [Na mreži]. Available: https://www.ietf.org.

[11] S. Jeloka, Advanced Security Administrator’s Guide 11g Release, 2014.

[12] S. Jeloka, Advanced Security Administrator’s Guide 11g Release 2 (11.2), 2009.

[13] „DBMS_NETWORK_ACL_ADMIN,“ [Na mreži]. Available: http://docs.oracle.com/cd/B28359_01/appdev.111/b28419/d_networkacl_adm.htm.

Spisak slika: