SIGURNOST I BEZBEDNOST ELEKTROENERGETSKOG SOFTVERA

PRAKTIKUM

2020/2021

Profesor : **Imre Lendak**
Asistenti: **Zorana Babić**
Stevan Boruš
Stefan Ruvčeski

Sadržaj

Uvod

Polaganje predmeta:

- Projekat – 60 bodova
- Test – 20 bodova
- Usmeni – 20 bodova

Vežbe :

- Alati koji se koristi prilikom izrade zadataka je Visual Studio (verzija po želji).
- Prisustvo na vežbama je obavezno (broj maksimalnih izostanaka je 2).
- Podela projakata će se održati online (30.11. i 01.12).
- Poslednje tri nedelje (od 14.12. do 10.01) se izrađuje projekat i tada prisustvo nije obavezno.
- Odbrane projekata će se održati u poslednjoj nedelji semestra 11.01. i 12.01.
- Odbrana projekata će se održati u prostorijama FTN-a nije moguća online odbrana.
- Projekat može ranije da se brani uz mogućnost nagradnih poena. ☺
- Postoji tri vrste projekata:
    - Projekat za 6
        - Maksimalni broj bodova koji može da se odvoji je 36.5
        - Minimalan broj bodova koji mora da se osvoji da bi student položio je 30.5
        - Radi se u timovima od po dva studenta
    - Redovan projekat
        - Maskimalan broj bodova koji može da se osvoji je 60
        - Minimalan broj bodova koji mora da se osvoji da bi student položio je 30.5
        - Radi se u timovima od po četiri studenta
    - Super projekat ☺
        - Maksimalan broj bodova koji može da se osvoji je 80
        - Minimalan broj bodova koji mora da se osvoji da bi student položio je 55
        - Studenti koji polože ovaj projekat ne moraju da izlaze na testić
        - Radi se u timovima od po dva studenta
        - Teorija koja je neophodna za ovaj projekat može da izlazi iz opsega teorije koja se radi na vežbama
- Odbrana projekata se sastoji od prikaza urađenog zadatka i dodele bodova na osnovu urađenog zadatka, to je maksimalan broj bodova koji može da osvoji svaki od članova tima, nakon toga svako od članova tima pojedinačno “brani” projekat i dobija broj bodova u zavisnosti od svog znanja i urađenog projekata.
- Oblasti koje će se obrađivati :
    - Autentifikacija, Impersonifikacija
    - Sertifikati, digitalni potpisi
    - Autorizacija, RBAC
    - Kriptografija
    - Auditing

Vezba 1 - Autentifikacija

Cilj ove vežbe je upoznavanje sa osnovnim elementima bezbedne komunikacije u .NET razvojnom okruženju, a zatim i sa mehanizmom autentifikacije.

Autentifikacija je jedan od osnovnih bezbednosnih mehanizama kojim se obezbeđuje validacija identiteta u okviru informacionog sistema. Da bi entitet mogao da pristupi sistemu potrebno je da dokaže da je on upravo onaj za koga se izjašnjava. Podaci kojima se korisnici predstavljaju sistemu i potvrđuju identitet nazivaju se kredencijali i mogu se podeliti u tri kategorije:

1) nešto što korisnik zna (npr. šifra),

2) nešto što korisnik ima (npr. pametna kartica),

3) nešto što korisnik jeste (npr. otisak prstiju).

Na primer, putnici se na carini autentifikuju pomoću pasoša tako što posedovanjem pasoša potvrđuju svoj identitet. Proces validacije pasoša, kao i procena sličnosti putnika sa slikom u pasošu predstavlja način autentifikacije kojim putnik dokazuje svoj identitet. U računarskim sistemima, entitet može biti korisnik, ali i računar, servis, aplikacija ili bilo koji uređaj. Autentifikacija treba da obezbedi validaciju identiteta i tako spreči pristup nevalidnim korisnicima.

Bezbedna komunikacija u .NET-u

Prvi korak prilikom uspostavljanja komunikacije između dva entiteta je definisanje komunikacionog protokola kako bi podaci bili poslati u formatu koji će druga strana moći da razume.

Bezbedna komunikacija uključuje skup bezbednosnih mehanizama koji će omogućiti zaštitu komunikacionog kanala, koji može da prolazi kroz neobezbeđene sisteme, od trećeg entiteta. Ovo se postiže izvršavanjem sledećih koraka:

- Definisanje autentifikacionog protokola kada se učesnici u komunikaciji dogovaraju na koji način će biti izvršena validacija identiteta svakog entiteta.
- Definisanje mera zaštite podataka od:
    - neovlašćenog pristupa, čitanja i otkrivanja (poverljivost) - definisanje algoritma za šifrovanje podataka, a zatim i razmena ključeva za dešifrovanje.
    - neovlašćenih izmena ili brisanja (integritet) – definisanje algoritma za detekciju izmene podataka.

U WCF .NET razvojnom okruženju se za povezivanje učesnika u komunikaciji koristi ***Binding.*** WCF nudi širok spektar ugrađenih bindinga, u zavisnosti od specifičnih zahteva aplikacije njihova podešavanja moguće je izmeniti ili definisati specifične custom bindinge.

Bindingom se se definiše:

- komunikacioni protokol: TCP, HTTP, IPC
- protokol za autentifikaciju: Windows autentifikacioni protokol ili autentifikacija upotrebom sertifikata
- bezbednosni mod kojim se definiše:
    - ***Transport Security*** mod koji obezbeđuje zaštitu uspostavljenog komunikacionog kanala, odnosno *point-to-point* zaštitu podataka
    - ***Message Security*** mod obezbezbeđuje zaštitu na nivou poruka koje se razmenjuju u komunikaciji.

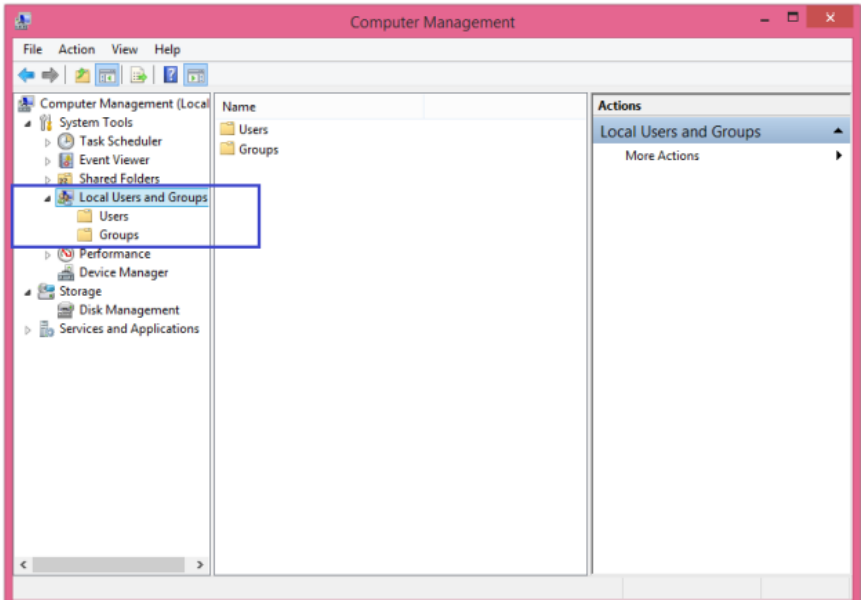
U nastavku je dat primer podešavanja različitih parametara ugrađenog ***NetTcpBinding*** tipa bindinga:

```
NetTcpBinding binding = new NetTcpBinding();
binding.Security.Mode = SecurityMode.Transport;

binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.Windows;

binding.Security.Transport.ProtectionLevel =
System.Net.Security.ProtectionLevel.EncryptAndSign;
```

Zadatak 1.1 Na primeru jednostane WCF klijent-servis aplikacije, u okviru koje WCF servis pruža mogućnost poziva metode AddUser, implementirati Windows autentifikaciju, i obezbediti zaštitu poverljivosti i integriteta podataka u okviru uspostavljene komunikacije.

Zadatak 1.2 Klijentsku i serversku aplikaciju iz prethodnog primera proširiti tako da prilikom startovanja ispisuju infomacije o identitetu korisnika (npr. ime) koji je pokrenuo proces. Zatim, podesiti radno okruženje tako da klijentska i serverska aplikacija budu pokrenute kao različiti korisnici.

Napomena: Za upravljanje lokalnim korisničkim nalozima i korisničkim grupama se koristi *Computer Management* Windows konzola prikazana na slici 1.



Slika 1- Computer managament

Zadatak 1.3 Proširiti metodu Read iz prethodnog zadatka tako da ispiše podatke o klijentu koji je pozvao ovu metodu. Potrebno je ispisati sledeće informacije: ime klijenta, njegov jedinstveni identifikator (*SecurityIdentifier*), informacije o tipu autentifikacije i o korisničkim grupama kojima dati korisnik pripada.

Autentifikacioni protokoli zasnovani na šiframa

Windows autentifikacioni model je zasnovan na SPNEGO (*Simple and Protected GSS API negotiation mechanism*) mehanizmu za pregovaranje između različitih realnih autentifikacionih mehanizmana u zavisnosti od okruženja. SSPI (*Security Support Provider Interface*) je Windows API koji implementira SPNEGO i predstavlja zajednički interfejs za različite Windows autentifikacione protokole (*Secure Service Provider*). Negotiate SSP je aplikativni protokol kojim je implementirano pregovaranje u Windows-u. Trenutno podržani protokoli su NTLM i Kerberos:

NTLM (*NT Lan Manager*) je autentifikacioni protokol zasnovan na *challenge-response* autentifikacionoj šemi, čime je omogućena autentifikacija bez slanja poverljivih podataka (šifre). Iako challenge-response spada u jake autentifikacione šeme jer nema razmene poverljivih podataka, problem ovakvih protokola je činjenica da servis mora da zna originalnu šifru svakog klijenta kako bi mogao da validira pristigli response. Dodatno, u ovako definisanom autentifikacionom protokolu izostaje verifikacija servisnog identiteta od strane klijenta, odnosno ovakav protokol **ne omogućava obostranu autentifikaciju.**

Da bi se obezbedila dvosmerna/obostrana autentifikacija u okviru challenge-response protokola, potrebno je da po istom principu kao što klijent dokazuje identitet servisu, i servis dokaže svoj identitet klijentu. Međutim, autentifikacioni protokoli koji se zasnivaju na obostranoj autentifikaciji i koji podrazumevaju razmenu poruka između dva učesnika u autentifikaciji na potpuno isti način su generalno nesigurni protokoli. Kako bi se obezbedili sigurniji protokoli, uvodi se treća strana od poverenja odnosno entitet kome veruju svi ostali učesnici u komunikaciji.

Kerberos *je dvosmerni autentifikacioni protokol koji* se zasniva na trećoj strni od poverenja i razmeni **ticketa** u cilju uspostavljanja bezbedne obostrane autentifikacije učesnika u komunikaciji bez razmene šifri. Kerberos je namenjen za domenska okruženja gde uslugu treće strane od poverenja ima posebno konfigurisani server, tzv. domen kontroler (DC). DC predstavlja autoritet na nivou celokupnog domena kome pripada skup računara i korisničkih naloga.

Tipično, Kerberos je siguran, dvosmerni autentifikacioni protokol preko koga Negotiate prvo pokušava da autentifikuje korisnike. Ukoliko iz bilo kog razloga Kerberos autentifikacija nije moguća, NTLM protokol će se koristiti.

Autentifikacija preko NTLM protokola neće biti realizovana u slučaju da je Kerberos autentifikacija pokušana, ali je bila neuspešna.

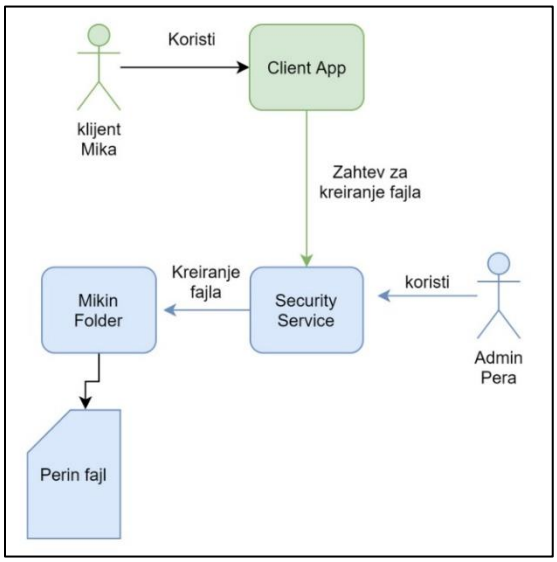
Zadatak 2.1 Proširiti WCF klijent-servis aplikaciju iz Zadatka 2. tako da prilikom uspostavljanja komunikacije bude definisan identitet servisa (*EndpointIdentity*).

Zadatak 2.2 Obezbediti da u slučaju kada Kerberos autentifikacija nije moguća, NTLM ne bude dozvoljen.

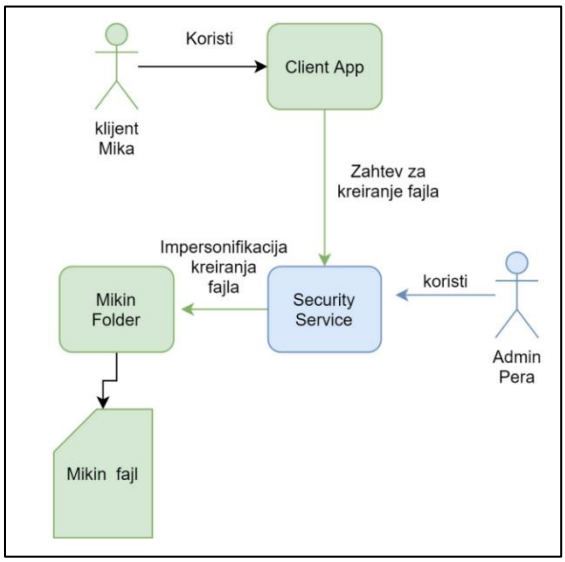
Vezba 2 – Impersonifikacija, Sertifikati

Impersonifikacija

Sposobnost niti da izvrši korišćenje različitih bezbednosnih informacija od procesa u čijem je vlasništvu nit. Obično se nit u aplikaciji servera impersonifikuje kao klijent. Ovo omogućava da nit servera deluje u ime tog klijenta da pristupi objektima na serveru ili da proveri valjanost pristupa objektima klijenta.



Dijagram 1- Kreiranje fajla, bez impersonifikacije



Dijagram 2 - Kreiranje fajla, sa impersonifikacijom

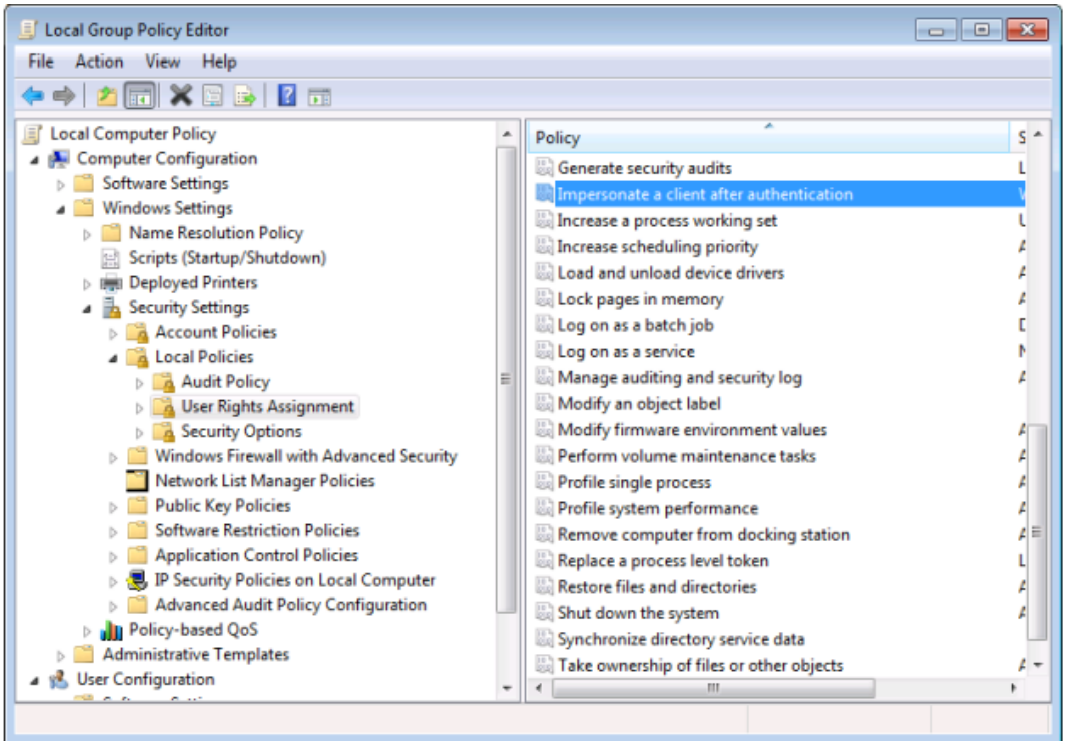
Na dijagramu 1. se može videti postupak kreiranja fajla od stane klijenta Mike bez impersonifikacije, dok se na dijagramu 2. koristi impersonifikacija. Bitno je primetiti razliku u tome ko je kreirao date fajlove sa i bez upotrebe imersonifikacije na servisnoj strani.

Zadatak 3.1. Proširiti zadatak rađen na vežbi 1 tako da bude omogućen poziv metode *CreateFile(string fileName)* u okviru koje se kreira datoteka unutar proizvoljnog foldera nad kojim samo servisni nalog ima pravo pristupa. Potvrditi da je fajl uspešno kreiran. U slučaju neuspešnog pokušaja kreiranja fajla, baciti izuzetak *SecurityException* i obraditi ga na klijentskoj strani.

Zadatak 3.2. Izmeniti metodu CreateFile tako da se izvršava pod klijentskim kredencijalima i potvrditi da će nakon toga kreiranje fajla unutar foldera nad kojim samo servisni nalog ima pravo biti neuspešno. Dodatno, verifikovati da je impersonifikacijom izmenjen identitet procesa.

Napomena: Za potrebe impersonifikacije potrebno je:

- Impersonifikovati odgovarajući deo koda.
- Podesiti proxy tako da klijent dozvoljava impersonifikaciju (TokenImpersonationLevel enumeracija definiše moguće nivoe).
- Grupnom polisom koja je prikazana na slici 2 je potrebno dozvoliti da servisni nalog impersonifikuje pozive klijenta.



Slika 2- Group Policy Editor

Zadatak 3.3. Implementirati metodu koja ima istu logiku kao *CreateFile* metoda, sa tom razlikom da se umesto implicitnog zahteva za impersonifikaciju koristi deklarativni način.

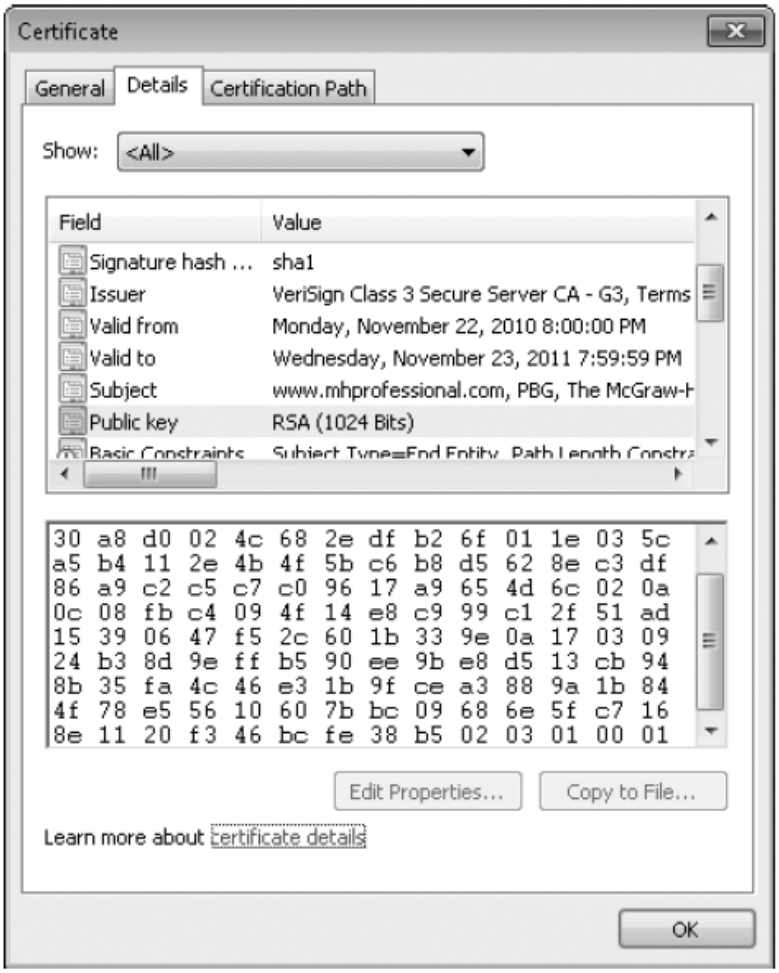
Napomena: Deklarativna impersonifikacija se definiše uz pomoć OperationBehavior atributa na sledeći način: *[OperationBehavior(Impersonation = ImpersonationOption.Required)]*

Sertifikati

Cilj ove vežbe je upoznavanje sa autentifikacionim modelom koji se zasniva na sertifikatima.

Sertifikat predstavlja digitalni identitet korisnika izdat od strane sertifikacionih tela (*certification authority, CA*). Na slici 3. je prikazan primer sertifikata. Sertifikat sadrži različite podatke: podaci o vlasniku sertifikata (*Subject*), validnost odnosno period važenja sertifikata (*ValidFrom, ValidTo*), informacije o izdavaocu sertifikata (*Issuer*). U sertifikat se ugrađuje javni ključ korisnika (uz identifikator algoritma primenjenog za generisanje ključa, npr. *RSA*), dok se tajni ključ ne razmenjuje.

Svaki sertifikat je digitalno potpisan od strane sertifikacionog tela koje ga izdaje čime se potvrđuje da sertifikat zaista pripada podnosiocu zahteva. Na ovaj način je takođe moguće detektovati izmene u okviru samog sertifikata jer digitalni potpis obezbeđuje integritet podataka.



Slika 3. - Primer sertifikata

Postupak izdavanja sertifikata

Sertifikaciono telo je komponenta zadužena za izdavanje i upravljanje sertifikatima. Odgovornosti svakog sertifikacionog tela su sledeće:

- **Verifikacija identiteta** (Verification of a certificate requestor) – Pre nego što izda sertifikat, odgovornost ovlašćenog lica, odnosno CA-a, je verifikacija identiteta onoga ko šalje zahtev. Kada korisnik šalje zahtev za sertifikat, on mora da pošalje sve neophodne informacije koje CA zatim ugrađuje u sertifikat. Tip sertifikata određuje njegov sadržaj, kao i namenu. Npr, IPSec sertifikat može da se koristi samo za autentifikaciju klijenta i servisa u okviru IPSec komunikacije.
- **Izdavanje sertifikata** (Issuing certificates to requestors) – Nakon uspešne verifikacije identiteta korisnika, računara, servisa, mrežnog uređaja CA izdaje digitalno potpisani sertifikat. Odnosno, sertifikat sa svim potrebnim informacijama će dodatno biti potpisan privatni ključem sertifikacionog tela kako bi se omogućila detekcija neovlašćenih izmena sadržaja sertifikata, ali i potvrda da je upravo određeni CA izdao sertifikat.
- **Povlačenje sertifikata** (Certificate revocation) – U nekim situacijama kao što je kompromitovanje sertifikata, potrebno je povući sertifikat iako je još uvek validan. CA ima listu povučenih sertifikata odnosno njihovih serijskih brojeva, uključujući i razlog zbog čega je svaki sertifikat povučen – CRL.

Uputsvo za izdavanje sertifikata

Potrebni alati:

- **makecert** – alat za generisanje sertifikata. Osim sertifikata sa odgovarajućim javnim ključem u okviru ***.cer*** fajla, moguće je generisati i dodatni ***.pvk*** fajl koji sadrži tajni ključ. Sertifikati kreirani na ovaj način su digitalno potpisani od strane testnih CA-eva.
- **pvk2pfx** – alat za generisanje .pfx fajla na osnovu javnog i privatnog ključa, odnosno testni sertifikat instaliran na osnovu .pfx fajla sadržaće i javni i privatni ključ. Za pristup privatnom ključu dodatno je potrebno dodeliti prava odgovarajućem korisniku nad prethodno instaliranim sertifikatom.
  Lokacija : *" C:\Program Files (x86)\Windows Kits\10\bin\[brojVerzije – razlicit kod svakoga]\x86" (ova dva alata se nalaze na istoj lokaciji)*
- **certmgr.msc** – mmc za rad sa sertifikatima. Pruža mogućnost rada sa sertifikatima specifičnim za trenutno logovanog korisnika na Windowsu (*currentuser*), kao i sa sertifikatima skladištenim u okviru određenog računara (npr. *localmachine*).
  Napomena*: Komanda certmgr.msc otvara storage za current user, dok se storage za local machine otvara na sledeći način:* ***Start → mmc.exe → File → Add/Remove Snap-in → Certificates → Add → Computer Account***

Potrebni korisnički nalozi

Korisnički nalozi: Kreirati lokalne korisničke naloge pod kojima će biti pokrenuta servisna, odnosno klijentska aplikacija. Lokalni nalozi se kreiraju u okviru "Computer Management" konzole.

- **wcfservice** – servisni nalog
- **wcfclient** – klijentski nalog

Primer generisanja sertifikata korišćenjem makecert alata

Napomena : Command Prompt pokrenuti kao Administrator

1. Generisanje TestCA sertifikata

```
> makecert -n "CN=TestCA" -r -sv TestCA.pvk TestCA.cer
```

Generisan je self-signed sertifikat, koji je potrebno instalirati na *"*Trusted Root Certification Authorities*"* lokaciji.

2. Generisanje WCFService sertifikata za uspostavljanje komunikacije

2.1. Generisanje .pvk i .cer fajla

```
> makecert -sv WCFService.pvk -iv TestCA.pvk -n "CN=wcfservice" -pe -ic TestCA.cer WCFService.cer -sr localmachine -ss My -sky exchange
```

Namena (uspostavljanje komunikacije) je definisano argumentom –sky exchange.

2.2. Generisanje .pfx fajla

```
> pvk2pfx.exe /pvk WCFService.pvk /pi 1234 /spc WCFService.cer /pfx WCFService.pfx
```

Karakteri koji se nalaze posle reči pi predstavljaju sifru sertifikata.
Ovaj fajl je potrebno instalirati (ili importovati iz certmgr.msc) na *"*Personal*"* lokaciji, koristeći opciju *"*Mark key as exportable*"*.

2.3. Dodeljivanje prava pristupa konkretnom korisničkom nalogu

U okviru certification managera neophodno je otici na sertifikat koji sadrzu privatni kljuc (.pfx).
Desni klik na sertifikat - > All Tasks - > Manage Private Keys i izvrsiti dodavanje korisnika kom treba da damo dozvolu da moze da koristi ovaj privatni ključ.
Napomena: **Ova komanda se izvršava nakon instaliranja sertifikata.**

3. Generisanje WCFClient sertifikata za uspostavljanje komunikacije

Na isti način kao u koraku 2 se generišu sertifikati koji služe za komunkikaciju, ono što je neophodno promeniti jeste subjectName "CN=wcfclient", i korisnički nalog kom se dodeljuje pristup je wcfclient.

4. Generisanje sertifikata za digitalno potpisivanje

4.1. Generisanje .pvk i .cer fajla

```
> makecert -sv Sign1.pvk -iv TestCA.pvk -n "CN=wcfclient_sign" -pe -ic TestCA.cer Sign1.cer -sr localmachine -ss My -sky signature
```

Namena (digitalno potpisivanje) je definisano argumentom –sky signature.
Ostali koraci se izvode kao i u koraku 3. s razlikom subjecName-a i naloga kom se dodeljuje pravo pristupa.
Napomena : Ovi sertifikati će se koristiti prilikom digitalnog potpisivanje koje će biti obrađivano u kasnijim lekcijama

Vezbe 3 – Autentifikacija upotrebom sertifikata

Obostrana autentifikacija uz pomoć sertifikata

Zadatak 4.1

Potrebno je obezbediti obostranu autentifikaciju uz pomoć sertifikata, pri čemu se validacija sertifikata zasniva na lancu poverenja (*chain trust*). Rešenje verifikovati pozivom metode *TestCommunication()*.

Prvi korak u realizaciji rešenja je generisanje sertifikata. S obzirom da je u pitanju obostrana autentifikacija sertifikatima, potrebno je izgenerisati sertifikat za svakog od učesnika u komunikaciji. *Sertifikate generisati tako da subjectName odgovara korisničkom imenu naloga*.

U nastavku su nabrojane komponente koje učestvuju u komunikaciji i čije sertifikate je neophodno instalirati na odgovarajuću lokaciju na računaru, a zatim za odgovarajuće korisnike dodeliti prava pristupa privatnom ključu odgovarajućeg sertifikata:

- ***TestCA*** – komponenta od poverenja zadužena za izdavanje ostalih testnih sertifikata. Prilikom instalacije kod korisnika izdatog sertifikata samo je javni ključ dozvoljeno instalirati. Privatni ključ je vlasništvo CA komponente i **nikad se ne instalira**. Instalira se u okviru ***Trusted Root Certification Authorities*** lokacije.
- ***WCFService*** – servisni sertifikat za autentifikaciju. Privatni ključ se instalira samo kod vlasnika sertifikata (odnosno na mašini na kojoj je pokrenut servis). Instalira se u okviru lokacije ***Personal***.
- ***WCFClient*** – klijentski sertifikat za autentifikaciju. Privatni ključ se instalira samo kod vlasnika sertifikata (odnosno na mašini na kojoj je pokrenuta klijentska aplikacija). Instalira se u okviru lokacije ***Personal***.

Potrebno je omogućiti čitanje sertifikata iz skladišta sertifikata, kao i iz fajlova (.cer i .pfx) za potrebe servisne i klijentske aplikacije. Implementacija ovih metoda treba da bude u posebnom CertificateManager DLL-u. Zaglavlja metoda su opisane u sledećem kodu:

```
public static X509Certificate2 GetCertificateFromStorage(StoreName storeName, StoreLocation storeLocation, string subjectName)

public static X509Certificate2 GetCertificateFromFile(string fileName)

public static X509Certificate2 GetCertificateFromFile(string fileName, SecureString pwd)
```

Prvi korak je podešavanje bindinga tako da podrži autentifikaciju uz pomoć sertifikata. Ovim je definisan tip kredencijala koji se očekuje od drugog učesnika u komunikaciji.

Zatim je potrebno da svaki učesnik u komunikaciju podesi svoj sertifikat kojim se predstavlja drugim učesnicima u komunikaciji.

- Svaka komponenta mora da podesi .pfx sertifikat, jer na taj način dokazuje da je vlasnik tog sertifikata (ima privatni ključ);
- Servis podešavanjem sertifikata na hostu: ***host.Credentials.ServiceCertificate.Certificate;***
- Klijent podešavanjem proxy-a: ***proxy.Credentials.ClientCertificate.Certificate;***
- .NET podržava X509 standard implementiran klasom ***X509Certificate2*** iz *System.Security.Cryptography.X509Certificates* namespace.

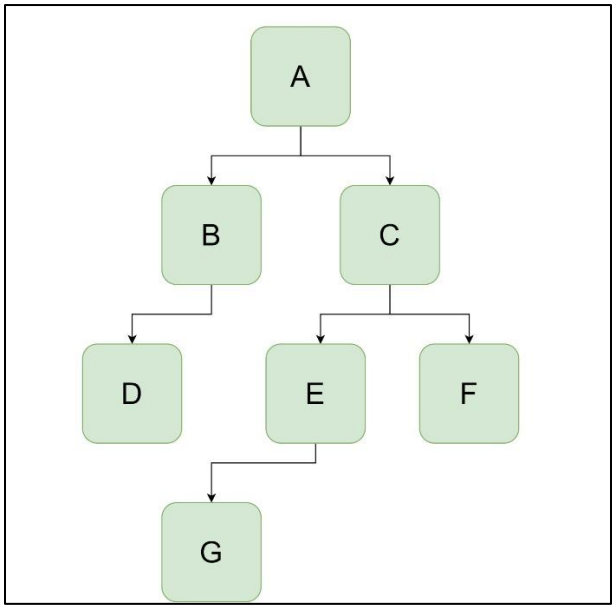
Kao deo obostrane autentifikacije, neophodno je ***podesiti identitet servisa koji klijent očekuje*** prilikom uspostavljanja komunikacije. Za to je potrebno proslediti *EndpointIdentity* prilikom uspostavljanja komunikacije. Objekat tipa *EndpointAddress* može se kreirati na sledeći način:

```
EndpointAddress address = new EndpointAddress(new Uri("net.tcp://localhost:9999/Receiver"),
new X509CertificateEndpointIdentity(srvCert)); //srvCer je .cer servisnog sertifikata
```

Napomena: Klijent mora instalirati serverski sertifika(.cer) na svoju mašinu u folder Thrusted People

Poslednji korak je definisanje tipa validacije sertifikata kod klijenta i servera.

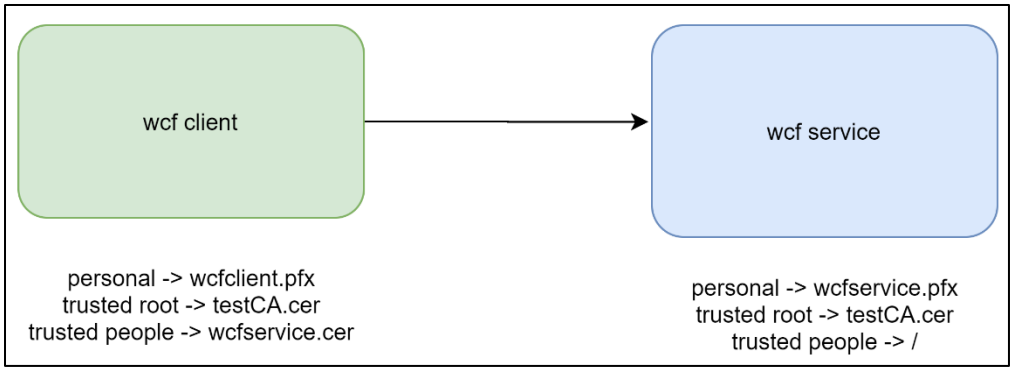
```
Credentials.ServiceCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.ChainTrust;
Credentials.ServiceCertificate.Authentication.RevocationMode =
       X509RevocationMode.NoCheck;

host.Credentials.ClientCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.ChainTrust;
host.Credentials.ClientCertificate.Authentication.RevocationMode =
       X509RevocationMode.NoCheck;
```



Slika 4- Chain Trust

Kod ChainTrust validacije se na osnovu izdavaoca sertifikata zaključuje da li će se verovati određenoj strani, odnosno ukoliko je sertifikat sa kojim se neko predstavlja izdat od strane nekoga kome određena strana veruje taj sertifikat će se smatrati vallidnim.

Na slici 4 možemo da vidimo jedan primer izdavanja sertifikata. Ukoliko koristimo ChainTrust validaciju možemo da kažemo da strana D i strana F mogu da veruju jedna drugoj jer je F izdao C, koji je izdao A, dok je D izdao B, kojeg je izdao A. I sam tim zaključujemo da su oba ova sertifikata potekla od istog sertifikata kom veruju obe strane.



Slika 5 - Instalacija sertifikata

Sa slike 5 možemo da vidimo na kojim sve lokacijama i koji sve sertifikati su nam neophodni za realizaciju zadatka.

Zadatak 4.2

Potrebno je izmeniti prethodni zadatak tako da se umesto *ChainTrust* validacije sertifikata koristi Custom validacija.

Za potrebe ovog primera, implementirati validatore (*nasleđuju klasu X509CertificateValidator*):

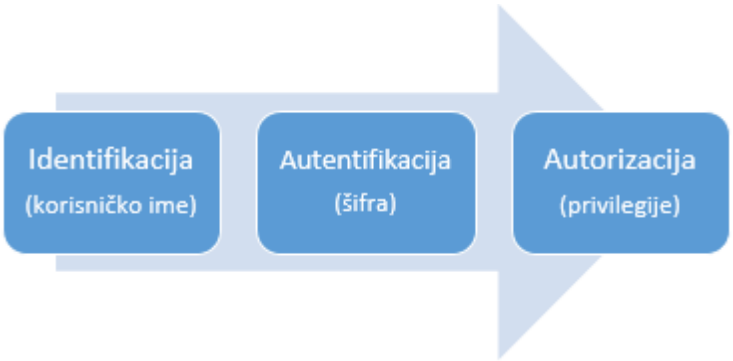
- ***ServiceCertValidator*** klasa – klijentski sertifikat je validan ukoliko je potpisan od strane istog sertifikacionog tela kao i servisni sertifikat;
- **ClientCertValidator** klasa – sevisni sertifikat je validan ukoliko nije self-signed.

```
Credentials.ServiceCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.Custom;
Credentials.ServiceCertificate.Authentication.CustomCertificateValidator =
       new ClientCertValidator();

host.Credentials.ClientCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.Custom;
host.Credentials.ClientCertificate.Authentication.CustomCertificateValidator =
       new ClientCertValidator();
```

Vezba 4 - Autorizacija

Cilj ove vežbe je upoznavanje sa mehanizmom autorizacije. Mehanizam autentifikacije treba da obezbedi validaciju identiteta i tako spreči pristup nevalidnim korisnicima. Međutim, različiti korisnici mogu imati različita ovlašćenja u sistemu kome pristupaju. Na primeru putnika koji se na carini autentifikuje tako što posedovanjem pasoša potvrđuju svoj identitet, to bi značilo da uspešno autentifikovani putnici mogu imati različita prava u zemlji u koju ulaze, jedni su državljani, drugima su prava definisana vizom pa tako možemo imati putnike koji smeju da borave u zemlji mesec dana, putnike kojima je dozvoljen boravak godinu dana, itd. Autentifikacijom nije moguće definisati različit nivo ovlašćenja za validne korisnike u sistemu. Proces definisanja ovlašćenja validnim entitetima u sistemu, kao i proces odlučivanja kojim resursima tog sistema entitet može da pristupi i koje operacije nad resursima može da izvrši se naziva autorizacija.

Iz prethodno opisanog primera mogu se uočiti tri procesa u okviru provere prava pristupa resursima u sistemu: identifikacija, autentifikacija i autorizacija. Ova tri procesa zajedno čine mehanizam kontrole pristupa kojim se definiše da li i na koji način korisnici mogu pristupiti resursima u sistemu, odnosno “ko šta može da radi u sistemu”. Na slici je konceptualno prikazan sistem kontrole pristupa. Identifikacija je proces u okviru koga se utvrđuje da li je predstavljeni entitet poznat sistemu. Entitet se predstavlja sistemu korišćenjem jedinstvenog identifikatora (npr. korisničko ime), ali se ne utvrđuje verodostojnost ove tvrdnje. Autentifikacija je proces validacije identiteta u sistemu. Da bi entitet mogao da pristupi sistemu potrebno je da dokaže da je on upravo onaj za koga se izjašnjava (npr. šifra). Autorizacija je proces provere ovlašćenja u sistemu, odnosno proces odlučivanja kojim resursima entitet može da pristupi i koje operacije može da izvršava nad resursima u sistemu.



Slika 6 - Sistem za kontrolu pristupa

U okviru ove vežbe, studenti će prvo biti upoznati sa osnovnim konceptima mehanizma kontrole pristupa koji je podržan u .NET razvojom okruženju, a zatim će biti implementiran model kontrole pristupa zasnovan na korisničkim ulogama (eng. *role-based access control – RBAC*).

Osnovni mehanizam kontrole pristupa u .NET-u

Zadatak 5.

U okviru ovog zadatka potrebno je obezbediti proveru prava korisnika koji pristupaju servisu pre nego što se dozvoli izvršvanje pozvanih metoda na sledeći način.

Servis nudi tri metode:

- Metoda **`Read`** – ovu metodu je moguće izvršiti ukoliko je korisnik član **`Reader`** grupe.
- Metoda **`Modify`** – ovu metodu je moguće izvršiti ukoliko je korisnik član **`Modifier`** grupe.
- Metoda **`Delete`** – ovu metodu je moguće izvršiti ukoliko je korisnik član **`Admin`** grupe.

U slučaju neuspešnog pokušaja pristupanja metodi, potrebno je baciti izuzetak (SecurityException) i obraditi ga na klijentskoj strani. U okviru poruke, izuzetak treba da vrati sledeće informacije: korisnik koji je pokušao da pristupi servisu, metoda servisa kojoj je pokušao da pristupi, grupa kojoj korisnik treba da pripada da bi mogao da pristupi metodi i vreme poziva.

Prilikom pokretanja aplikacija koristiti lokalne korisničke naloge i korisničke grupe. Podesiti radno okruženje tako da klijentska i serverska aplikacija budu pokrenute kao različiti korisnici, npr. servisna aplikacija da bude pokrenuta kao **`wcfservice`** korisnik, a klijentska aplikacija kao **`wcfclient`** korisnik.

Napomena: Kontrolu pristupa na WCF servisu moguće je implementirati na dva načina:

- Deklarativna provera privilegija koristeći **`PrincipalPermissionAttribute`** atribut.
- Imperativna provera privilegija pozivom metode **`IsInRole`** **`IPrincipal`** interfejsa.

Zadatak 5.1 Deklarativna provera privilegija

Deklarativna provera privilegija se izvršava pre ulaska u pozvanu metodu.

```
[PrincipalPermission(SecurityAction.Demand, Role="Ime_privilegije")]
void MetodaServisa()
{ ... }
```

Zadatak 5.2 Imperativna provera privilegija

Imperativna provera privilegija podrazumeva eksplicitan poziv metode za proveru privilegija na proizvoljnom mestu u telu metode.

Zadatak 5.3

Izmeniti Zadatak 5 tako da ukoliko korisnik nije član **Reader** grupe ne može da pristupi nijednoj metodi. Ukoliko je korisnik član **Reader** grupe, proveriti prava pristupa metodama servisa kao što je opisano u Zadatku 5.

Napomena: WCF nudi mogućnost centralizovane kontrole pristupa u slučajevima kada postoji zajednički preduslov za pristup svim metodama servisa. **ServiceAuthorizationManager** je ugrađena .NET klasa nudi mogućnost override-a **CheckAccessCore** metode u okviru koje se definiše ponašanje servisa prilikom poziva metoda koje on izlaže. Podrazumevano ponašanje ove metode je da vraća vrednost **true**, odnosno da je pristup dozvoljen.

Jedan način registracije objekta ServiceAuthorizationManager klase na WCF servisu je:

```
ServiceHost host = new ServiceHost(typeof(WCFService));
host.AddServiceEndpoint(typeof(WCFService), binding, address);

host.Authorization.ServiceAutorizationManager = new MyAuthorizationManager();
```