CARNet
HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Sigurnosni propusti ASP .Net aplikacija

CCERT-PUBDOC-2008-05-229

+CERT.hr u suradnji s LSS

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada je i ovaj 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 4

2. OSNOVE ASP.NET PROGRAMIRANJA 5

2.1. POSLUŽITELJSKE UPRAVLJAČKE OZNAKE 5

2.1.1. Vrste upravljačkih oznaka i primjeri 5

2.1.2. Sintaksa Web Formi i čuvanje stanja stranice 7

2.2. RAD S PODATKOVNIM OBJEKTIMA I BAZAMA PODATAKA 8

2.2.1. Podatkovni objekti 8

2.2.2. Baze podataka 9

3. OBLIKOVANJE SIGURNIH ASP.NET PROGRAMA 11

3.1. PODRUČJA RANJIVOSTI PROGRAMA 11

3.1.1. Provjera ulaznih podataka 11

3.1.2. Provjera identiteta 12

3.1.3. Provjera ovlasti 12

3.1.4. Upravljanje konfiguracijom 12

3.1.5. Upravljanje sjednicama 13

3.1.6. Kriptografija 13

3.1.7. Zlouporaba parametara 13

3.1.8. Upravljanje posebnim događajima 13

3.1.9. Praćenje korisničkih aktivnosti 14

3.2. SMJERNICE SIGURNOG PROGRAMIRANJA 14

3.2.1. Provjera ulaznih podataka 14

3.2.2. Zaštita osjetljivih podataka 14

3.2.3. Sigurno upravljanje sjednicama i konfiguracijom aplikacije 15

3.2.4. Sigurno upravljanje parametrima i iznimkama 16

3.2.5. Praćenje korisničkih aktivnosti 16

3.3. APLIKACIJE ZA PROVJERU PROGRAMSKOG KODA 16

3.3.1. FxCop 17

3.3.2. Pretraživanje teksta 18

4. OBLIKOVANJE POSLUŽITELJSKIH I KLIJENTSKIH APLIKACIJA 20

4.1. POSLUŽITELJSKA APLIKACIJA 21

4.2. KLIJENTSKA APLIKACIJA 22

5. ZAKLJUČAK 24

6. REFERENCE 24

1. Uvod

ASP.NET je razvojno okruženje za izgradnju web aplikacija, a razvila ga je tvrtka Microsoft. Nasljednik je ASP (eng. Active Server Pages) tehnologije, a prvi put je izdan u siječnju 2002. godine uz programski paket .NET Framework. ASP.NET programerima omogućuje razvoj dinamičkih web stranica i usluga, a budući da koristi CLR (eng. Common Language Runtime) sastavnicu, razvojni jezik može biti bilo koji podržani .NET programski jezik (C#, Visual Basic).

ASP tehnologija, u odnosu na statične HTML stranice, daje mogućnost ugrađivanja izvršnog koda u sadržaj stranice koji se izvodi se na poslužitelju prije nego se zahtijevana stranica pošalje pregledniku korisnika (klijenta). Time je omogućen dinamičan i interaktivan rad. Novost koju ASP.NET uvodi u odnosu na klasičnu ASP tehnologiju jest mogućnost odvajanja izvršnog koda i HTML sadržaja. Za ugradnju izvršnog koda koristi se poseban <script runat="server"></script> blok čime se poboljšava čitljivost i olakšava održavanje programa.

Uz nove pogodnosti, nove tehnologije donose i nove rizike. U ovom slučaju radi se o zlouporabi interaktivnog načina rada. Ukoliko se web usluga oblikuje s neopravdano velikim povjerenjem u korisnika i način na koji će on uslugu koristiti, konačan će proizvod sadržavati mnoge ranjivosti. U slučaju web aplikacija bilo kakva pretpostavka o dobronamjernosti korisnika, neopravdano je veliko povjerenje, a time i izvor nove ranjivosti. Iz tog razloga jedan je od osnovnih principa u oblikovanju programske potpore defenzivnost. Ovaj dokument bavi se sigurnošću web aplikacija razvijenih uz pomoć ASP.NET alata i načinima na koje se postojeće opasnosti mogu ukloniti ili barem umanjiti.

2. Osnove ASP.NET programiranja

ASP.NET web stranice u osnovi su vrlo slične običnim HTML stranicama, ali u imenu sadrže ".aspx" nastavak. Razlika u načinu na koji će poslužitelj tretirati zahtjeve za „.htm" i „.aspx" stranicama jest ta što se „.htm" stranice šalju pregledniku bez ikakvih izmjena sadržaja, dok unutar „.aspx" stranica mogu biti uključene skripte koje se izvode na poslužitelju prije slanja stranice pregledniku.

U klasičnoj ASP tehnologiji izvršni kod postavlja se unutar <% %> oznaka, a poziva se na onom mjestu u HTML kodu gdje se očekuje prikazivanje povratne vrijednosti (Primjer 1.). Posljedica ovakvih zahtjeva na programiranje je nemogućnost odvajanja izvršnog koda od HTML sadržaja i strukturiranog programiranja. ASP.NET podržava ASP sintaksu, ali uvodi i upravljačke oznake.

```
<html>
<body>
<center>
<% Response.Write("Hello World!") %>
</center>
</body>
</html>
```

Primjer 1. Jednostavan „Hello world!" program

2.1. Poslužiteljske upravljačke oznake

Poslužiteljske upravljačke oznake programirljivi su objekti koje koristi poslužitelj, a najčešće se rabe u programskom ostvarenju elemenata korisničkog sučelja kao što su polja za unos teksta, padajući izbornici, gumbi za predaju unesenog sadržaja (eng. submmit) i sl. Prednost uporabe poslužiteljskih upravljačkih oznaka u odnosu na <% %> sintaksu jest mogućnost programiranja vrlo složenog prikaza dinamičke web stranice uz korištenje relativno malih programskih odsječaka. Osim strukturiranog programiranja i izvedbe reakcija na događaje potaknute od strane klijenta (pritisak mišem na određeni gumb, unos podataka), upravljačke oznake pojednostavljuju čuvanje stanja prilikom uzastopnih razmjena stranice između klijenta i poslužitelja, npr. kod provjere lozinke prilikom prijave na sustav.

Upravljačke oznake deklariraju se unutar HTML oznaka s obavezno navedenim atributom *runat="server"*. Sintaksa stvaranja standardne poslužiteljske upravljačke oznake jest:

```
<asp:ime_oznake id="id_oznake" runat="server" />
```

2.1.1. Vrste upravljačkih oznaka i primjeri

Postoji cijeli niz upravljačkih objekata, a ovdje se razrađuju jednostavni primjeri standardnih HTML oznaka i oznaka za provjeru ulaznih podataka.

Primjer 2 ilustrira uporabu standardnih elemenata *textbox*, *label* i *button*. Ukoliko korisnik upiše „Jerry" u polje Ime i „miš" u polje Vrsta, nakon pritiska na gumb Sacuvaj, na stranici ce se ispisati „Jerry je miš".

```
<script language="VB" runat="server">
Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
    Poruka.Text = HttpUtility.HtmlEncode(Ime.Text) & " je " & Vrsta.Text
End Sub
</script>
```

```
<html>
<body>
<center>
<form id="Form1" action="d.aspx" method="post" runat="server">
<p>
<h3>
Ime:  <asp:textbox id="Ime" runat="server"/>
Vrsta:<asp:textbox id="Vrsta" runat="server"/>
</h3>
<asp:button Id="Button1" text="Sacuvaj" OnClick="SubmitBtn_Click" runat="server"/>
</p>
<asp:label id="Poruka" runat="server"/>
</form>
</center>
</body>
</html>
```

Primjer 2. Upotreba standardnih oznaka label, textbox i button

Primjer 3 primjer je upotrebe HTML upravljačke oznake *HtmlAnchor* koja omogućuje preusmjeravanje na druge web stranice. U ovom slučaju otvaranjem „Posjeti CERT!“ simboličke veze korisnik se preusmjerava na http://www.cert.hr stranicu.

```
<script language="VB" runat="server">
Sub Page_Load(sender As Object, e As EventArgs)
        odrediste.HRef = "http://www.cert.hr"
End Sub
</script>

<html>
<body>
<form runat=server>
<p>
<a id=odrediste runat="server">
Posjeti CERT!
</a>
</form>
</body>
</html>
```

Primjer 3. Upotreba HTML oznake HtmlAnchor

Primjer 4 prikazuje upotrebu upravljačkog objekta za provjeru korisničkog unosa. Od korisnika se traži unos broja između 1 i 10. *RangeValidator* objekt provjerava uneseni broj i u slučaju pogreške ispisuje poruku o neispravnom unosu.

```
<html>
<body>
<form runat="server">
<p>Upisite broj od 1 do 10:
<asp:TextBox id="broj" runat="server" />
<br /><br />
```

```
<asp:Button Text="Ucitaj" runat="server" />
</p><p>
<asp:RangeValidator
ControlToValidate="broj"
MinimumValue="1"
MaximumValue="10"
Type="Integer"
Text="Trazi se broj izmedu 1 i 10!"
runat="server" />
</p></form>
</body>
</html>
```

Primjer 4. Upotreba RangeValidator oznake

2.1.2. Sintaksa Web Formi i čuvanje stanja stranice

U prethodnim primjerima uočljivo je da se svi poslužiteljski upravljački objekti nalaze unutar <form></form> bloka koji ima definiran *runat="server"* atribut. Jedna „.aspx" stranica može sadržavati isključivo jedan takav blok, a njegov sadržaj čine HTML kod i upravljački objekti. Pretpostavljena metoda bloka je *method = „post"* što znači da je uključen rad s podacima koji utječu na stanje poslužitelja (korisnik može promijeniti podatke na poslužitelju). Alternativa ovoj metodi je *method = „get"* koja uključuje samo preuzimanje podataka s poslužitelja.

Osim već navedene sintakse za prikaz sadržaja na stranici u primjeru Primjer1, može se koristiti i <%= %> blok. Izvođenje naredbe:

```
<% Response.Write("Hello World"); %>
```

Ovaj poziv daje iste rezultate kao izvođenje naredbe:

```
<%="Hello World" %>
```

Sve metode i globalne varijable stranice deklariraju se unutar <script runat="server"> bloka (Primjer 2 i Primjer 3). Oni doprinose ostvarenju jasne i logične strukture programskog koda.

Komentari, dijelovi koda koji se zanemaruju pri obradi, omotavaju se <%-- --%> oznakama, a uključivanje sadržaja datoteke u prikaz stranice obavlja se pomoću

<-- #Include File="ime_datoteke" --> oznake.

```
<html>
<body>
 <%--Primjer ispisuje sadrzaj datoteke
"datoteka.inc" na web stranici --%>
<!-- #Include File="datoteka.txt" -->
</body>
</html>
```

Primjer 5. Upotreba komentara i datoteka

Očuvanje stanja stranice podrazumijevana je postavka za ASP.NET web forme, a znači da se sve unesene vrijednosti elemenata korisničkog sučelja čuvaju nakon predaje stranice poslužitelju. Postavka je definirana u skrivenom polju unutar svake stranice s

<form runat="server"> blokom, a promjena njezine vrijednosti ostvaruje se dodavanjem EnableViewState="false" atributa bilo kojem upravljačkom objektu, ili uključivanjem

<%@ Page EnableViewState="false" %> direktive na početku „.aspx“ stranice.

2.2. Rad s podatkovnim objektima i bazama podataka

Svaka dinamička web stranica radi s podacima, bilo da koristi baze podataka, XML datoteke ili podatkovne objekte. Najčešće akcije pritom su dodavanje, brisanje, mijenjanje, sortiranje, izbor određenog podskupa podataka, njihov prikaz na stranici i slično. ASP.NET koristi dvije vrste objekata za rad s podacima. Jedna su objekti za povezivanje s izvorima podataka, a drugi su objekti za prikaz podataka na stranici.

2.2.1. Podatkovni objekti

Podatkovni objekti (eng. *data objects*) programske su strukture za čuvanje podataka u određenom obliku. Neke jednostavne strukture su:

- *ArrayList* - lista čiji se članovi sastoje od jednog podatkovnog elementa.
- *SortedList* – lista čiji su članovi uz vrijednost definirani i ključem po kojem se lista automatski sortira, npr. prvo slovo prezimena.
- *Hashtable* – članovi raspršenih tablica sadrže elemente ključ i vrijednost. Elementi su u tablici postavljaju na mjesto definirano ključem što uvelike ubrzava pretraživanje.

Sve strukture uključuju i odgovarajuće metode za upravljanje podatkovnim objektima.

Povezivanje podatkovnih objekata s upravljačkim objektima obavlja se uz pomoć *DataBind* metode upravljačkog objekta. Nakon *DataBind* poziva izvršava se kod unutar <%# %> bloka nadređenog upravljačkog objekta, kao što je prikazano u nastavku.

```
<script runat=server>
Sub Page_Load(Sender As Object, E As EventArgs)
   Dim Lista As New ArrayList
   Lista.Add("ASP")
   Lista.Add("ASP.NET")

   ASPvsASP.NET.DataSource = Lista
   ASPvsASP.NET.DataBind()
End Sub

Sub Klik(Sender As Object, E As EventArgs)
   Poruka.Text = "Izabrali ste " & ASPvsASP.NET.SelectedItem.Text
End Sub
</script>

<html>
<body>
<center>
<form runat=server>
<asp:dropdownlist id="ASPvsASP.NET" runat=server>
<asp:button text="Izaberi" OnClick="Klik" runat="server"/>
<p>
```

```
<asp:label id="Poruka" runat="server"/>
</form>
</center>
</body>
</html>
```

Primjer 6. Primjer stvaranja i povezivanja podatkovnih objekata

2.2.2. Baze podataka

Bazu podataka čine međusobno povezani podaci, pohranjeni bez zalihosti, a prilikom rada s takvim skupom podataka koristi se kontrolirani pristup. Baze podataka osobito su pogodne za čuvanje velikog skupa podataka, a mogu se koristiti u različitim programskim primjenama. Web aplikacije vrlo često koriste baze podataka, te se zato na jednostavnom primjeru prikazuje povezivanje i korištenje baze podataka u ASP.NET programima. U primjeru 7 prilikom otvaranja stranice otvara se i veza s bazom podataka *Baza.mdb* i izvodi se SQL upit nad tablicom *predmet* koji kao rezultat vraća sve postojeće zapise u tablici. Pojedini se zapis pritom sastoji od stupaca *naziv* i *ECTS* koji predstavljaju naziv predmeta i broj ECTS bodova koje on nosi. Rezultat SQL upita sprema se u varijablu *Rezultat* koja se koristi kao izvor podataka prilikom povezivanja podataka s upravljačkim objektom. U ovom slučaju to je *DataList*, niz podataka čiji je uzorak definiran u bloku <ItemTemplate>. Nakon što je rad s bazom završen, veze i korišteni objekti se zatvaraju kako bi se izbjeglo nepotrebno zauzimanje memorije.

```
<%@ Import Namespace="System.Data.OleDb" %>

<script  runat="server">
sub Page_Load
dim dbVeza,SQLUpit,dbNaredba,Rezultat
dbVeza=New  OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;  Data
Source=c:\Baza.mdb;")
dbVeza.Open()
SQLUpit="SELECT * FROM predmet"
dbNaredba=New OleDbCommand(SQLUpit,dbVeza)
Rezultat=dbNaredba.ExecuteReader()
predmeti.DataSource=Rezultat
predmeti.DataBind()
Rezultat.Close()
dbVeza.Close()
end sub
</script>

<html>
<body>
<form runat="server">
<asp:DataList id="predmeti" runat="server">

<ItemTemplate>
<%#Container.DataItem("naziv")%>  nosi
<%#Container.DataItem("ECTS")%> ECTS bodova.
</ItemTemplate>
```

```
</asp:DataList>
</form>
</body>
</html>
```

Primjer 7. Primjer rada s bazom podataka

3. Oblikovanje sigurnih ASP.NET programa

Poznavanje očekivane usluge i programskog jezika dovoljno je za izgradnju web aplikacije koja će uspješno obavljati očekivane zadaće, no kvalitetan programer dužan je voditi računa i o sigurnosti. Programska potpora za korištenje bankovnog računa putem Interneta mora korisniku omogućiti pristupanje vlastitom računu, ali jednako je važno onemogućiti pristupanje računima za koje korisnik nema ovlasti. Sigurnost je iznimno bitan element u oblikovanju Web aplikacija jer je Internet globalna mreža putem koje je moguće pristup bilo kojoj usluzi, od bilo kuda, a stvarnu, fizičku osobu koja pristupa prilično je teško identificirati.

Voditi računa o sigurnosti nužno je na svim slojevima, od mrežne infrastrukture, preko računala poslužitelja do dizajna samog programa. Osnovni principi u oblikovanju sigurne usluge su:

- **Provjera korisnika** – utvrđuje se identitet korisnika, čovjeka ili računala, koji pokušava pristupiti zaštićenom sustavu. Najčešće se izvodi pomoću korisničkog imena i lozinke.
- **Autorizacija** – postupak kojim se prepoznatom korisniku dodjeljuju pripadna prava. To mogu biti prava pristupa datotekama, bazama podataka, prava pokretanja programa i sl.
- **Praćenje korisničkih aktivnosti** – bilježe se aktivnosti svakog korisnika na sustavu.
- **Očuvanje tajnosti povjerljivih podataka** – zaštita dodjelom prava pristupa ili enkripcijom osjetljivih podataka.
- **Očuvanje integriteta podataka** – zaštita od slučajne ili namjerne neovlaštene promjene.
- **Očuvanje dostupnosti** – sustav treba ostati dostupan ovlaštenim korisnicima.

3.1. Područja ranjivosti programa

Pojam računalne sigurnosti veže se uz različite slojeve računala i mreže, počevši od fizičkog sloja pa sve do programske potpore. Kod mrežnog rada potrebno je voditi računa o sigurnosti na svim razinama, uključujući fizičko ostvarenje, podatkovnu poveznicu, transportni sloj, mrežni sloj i aplikacijski sloj. Pojava propusta na bilo kojoj od razina otvara prostor za zlouporabu. Budući da se ovaj dokument bavi sigurnošću ASP.NET aplikacija, razraditi će se ranjivosti vezane uz aplikacijski sloj, uz pretpostavku da su ostali, niži slojevi prikladno zaštićeni.

3.1.1. Provjera ulaznih podataka

Pretpostavka da će korisnik unijeti podatke u onom formatu koji se od njega traži čini program ranjivim jer neočekivani podaci mogu dovesti do rušenja programa ili do pokretanja proizvoljnog programskog koda. Posljedice nedovoljne provjere ulaznih podataka mogu biti:

1) Pojava prepisivanja spremnika (eng. buffer overflow) – može dovesti do rušenja programa ili do pokretanja proizvoljnog programskog koda. Pažljivo oblikovani ulazni podaci mogu prebrisati programski stog i promijeniti povratnu adresu funkcije. Na taj način moguće je, kao adresu na kojoj se nalazi prva iduća naredba za izvođenje, podmetnuti adresu ubačenog, zlonamjerno oblikovanog programskog koda.
2) XSS (eng. Cross Site Scritping) napad – ovaj napad cilja na korisnika nesigurno oblikovanog programa. Ako web usluga ne provjerava ulazne podatke, u njima se može podmetnuti zlonamjerno oblikovana skripta (unutar *Uniform Resource Locator* URL zahtjeva) što će dovesti do njezinog izvršavanja u korisnikovom pregledniku. Uspješnim XSS napadom mogu se ukrasti korisnički identifikacijski podaci koji se zatim mogu koristiti za neovlašteno pristupanje web odredištu i lažno predstavljanje.
3) SQL injekcija – ukoliko se prilikom oblikovanja SQL upita koriste neprovjereni korisnički podaci, moguće je izvršiti SQL upit ili naredbu za koju napadač nema ovlasti

4) Kanonizacija – vezana je uz korisnički unos URL zahtjeva i putanja do datoteka (eng. path), a može se zlorabiti ukoliko se uneseni podaci ne pretvaraju u standardizirani, kanonski oblik.

3.1.2. Provjera identiteta

Neodgovarajuća izvedba mehanizma za provjeru identiteta lokalnom ili udaljenom napadaču omogućuje stjecanje neovlaštenog pristupa sustavu. Moguće ranjivosti vezane uz postupak provjere identiteta su:

1) Mrežno prisluškivanje – nezaštićeni prijenos lozinki i korisničkih imena čini ih dostupnima napadaču koji posjeduje alat za hvatanje mrežnog prometa.
2) Otkrivanje lozinki – slabe lozinke podložne su računalnim metodama pogađanja (eng. Brute Force Attack). Ako se provjera lozinki obavlja preko funkcija koje sažimaju podatke na fiksnu duljinu (eng. hash), moguće je izvesti tzv. „Dictionary Attack", napad koji se sastoji od računanja vrijednosti za različite riječi iz rječnika jednog ili više jezika i uspoređivanja s podacima u tablici. Ukoliko napadač uoči odgovarajuće iznose, lozinka je pogođena.
3) Otkrivanje kolačića za ovjeru identiteta – kao i kod mrežnog prisluškivanja, nezaštićena komunikacija napadaču omogućuje otkrivanje autentifikacijskih kolačića i njihovo korištenje za pristupanje sustavu.
4) Krađa osjetljivih podataka – pristupanje identifikacijskim atributima u priručnoj memoriji (eng. cash) preglednika, ili otkrivanje nesigurno pohranjenih korisničkih podataka.

3.1.3. Provjera ovlasti

Na temelju identiteta korisnika dodjeljuju mu se pripadne ovlasti za pristupanje pojedinim resursima sustava. Nepažljivo programsko ostvarenje mehanizama za provjeru ovlasti omogućuje:

1) Stjecanje dodatnih ovlasti na sustavu – ukoliko napadač uspije povisiti ovlasti do administratorske razine može preuzeti potpunu kontrolu nad programom i sustavom.
2) Neovlašteno pristupanje podacima – neodgovarajuća provjera ovlasti napadaču može omogućiti otkrivanje i izmjenu podataka kojima ne bi smio pristupati, tj. mijenjati
3) Napadi iz zasjede – izvedivi su ako korisnik ili program sam nema ovlasti za izvođenje pojedinih radnji, ali ima pristup programskom kodu koji te ovlasti posjeduje. Korištenjem tog programskog koda (onoga koji posjeduje odgovarajuće ovlasti) napadač može pokrenuti izvršavanje radnji za koje nije ovlašten.

3.1.4. Upravljanje konfiguracijom

Dozvolu pristupa konfiguracijskim podacima sustava u pravilu posjeduju korisnici s administratorskim ovlasti. Neoprezno upravljanje konfiguracijom web usluge neovlaštenim korisnicima ostavlja prostora za neovlašteno obavljanje administratorskih uloga:

1) Slabo osiguranje administratorskog sučelja i konfiguracijskih podataka - olakšava stjecanje neovlaštenog pristupa tim bitnim dijelovima programske usluge.
2) Nedovoljno praćenje korisničkih aktivnosti - onemogućuje uočavanje potencijalnih prijestupa i pokušaja narušavanja sigurnosti sustava.
3) Nepotrebno visoke ovlasti – dodjela nepotrebno visokih ovlasti koje uključuju izmjenu konfiguracijskih podataka na sustavu stvara opasnost od zlouporabe.

3.1.5. Upravljanje sjednicama

Upravljanje sjednicama zadaća je aplikacijskog sloja, a ukoliko programsko ostvarenje te zadaće nije razvijeno s dovoljnim naglaskom na sigurnosti, otvara se prostor za izvođenje nekoliko vrsta napada:

1) Otimanje sjednica – događa se u sličnim situacijama kao i krađa identiteta. Ukoliko prisluškivač mrežnog prometa uspije uhvatiti autentifikacijski kolačić može ga iskoristiti za lažno predstavljanje i pristupanje aplikaciji.
2) Ponavljanje sjednica – ukoliko napadač presretne korisnički autentikacijski kolačić i iskoristi ga za zaobilaženje postupka ovjere identiteta i korištenje usluge.
3) Uplitanje u komunikaciju - ukoliko napadač uspije presresti poruke odaslane od strane klijenta ili poslužitelja može ih izmijeniti i vratiti u promet. Na taj način neovlašteni subjekt u sredini lažira komunikaciju između usluge i ovlaštenog korisnika. Ova vrsta napada poznata je pod nazivom „Man in the Middle Attack".

3.1.6. Kriptografija

Kriptografija je važan element zaštite osjetljivih podataka, bilo da su pohranjeni ili se prenose mrežom. Ranjivosti kriptiranih podataka proizlaze iz:

1) Slabe zaštite kriptografskih ključeva – ukoliko su lako izvedivi ili nesigurno pohranjeni. Poznavanje kriptografskog ključa omogućuje dekripciju podataka.
2) Slabih kriptografskih algoritama – omogućuju lako pogađanje kriptografskih ključeva.
3) Korištenje funkcija za sažimanje – ne štite od promjene podataka u prijenosu.

3.1.7. Zlouporaba parametara

Korisnik lako može utjecati na oblik pojedinih parametarskih podataka jer nisu zaštićeni u prijenosu. Pritom se misli na polja web obrazaca (eng. web form), HTTP zaglavlje, kolačiće (eng. cookie) i znakovne nizove koji predstavljaju upite poslužitelju. Zato je potrebno uvesti provjere koje će otkloniti mogućnost narušavanja sigurnosti sustava njihovim zlonamjernim oblikovanjem. Najčešće ranjivosti vezane uz zlouporabu parametarskih podataka su:

1) Korištenje HTTP GET metode u pravilu ne bi smjelo utjecati na stanje poslužitelja, no način na koji će poslužitelj rukovati ovim zahtjevom nije tehnički ograničen. Zbog toga neoprezno programiranje HTTP GET metodu može učiniti opasnom.
2) Slanje osjetljivih podataka u znakovnim nizovima koji predstavljaju upit čini aplikaciju ranjivom jer je izmjena tih podataka jednostavno izvediva (budući da su oni dio URL zahtjeva prikazanog u web pregledniku).
3) Također HTML obrasci šalju se kao običan tekst putem HTTP POST protokola. Budući da ih napadač može lako izmijeniti, oslanjanje na njihovu regularnost programe čini ranjivima.
4) Oslanjanje sigurnosti web usluge na ispravnost HTTP zaglavlja također je izvor ranjivosti.

3.1.8. Upravljanje posebnim događajima

Upravljanje posebnim događajima (eng. exception) koje klijentu odaje previše informacija o sustavu također je jedan od izvora ranjivosti .NET razvojnog okruženja. Sustavi za opravljanje posebnim događajima su važni programerima koji razvijaju programsku potporu, ali propuštanje tih informacija do krajnjeg korisnika ne samo da je nepotrebno, već je i opasno. Krajnji korisnik neće razumjeti opis iznimke, ali napadač iz njega može izvući informacije koje će mu koristiti u osmišljavanju daljnjih strategija napada. Postoje dvije vrste napada koje posljedica lošeg upravljanja iznimkama:

1) Napad uskraćivanja usluge (eng. Denial of Service) – ukoliko program ne obavi sve provjere i ne predvidi sve moguće iznimne situacije, podmetanje vješto oblikovanih podataka može dovesti do rušenja aplikacije.

2) Ukoliko su iznimke predviđene, ali klijentu šalju previše podataka o sustavu, npr. inačice operacijskog sustava i podatke o bazi podataka, napadač ih može iskoristiti u novim napadima na sustav.

3.1.9. Praćenje korisničkih aktivnosti

Praćenje korisničkih aktivnosti važno je u prepoznavanju pokušaja napada prije nego izvedba napada doista i uspije te u povezivanju postupaka s počiniteljem. Nekorištenje alata za praćenje aktivnosti i slaba zaštita dnevničkih zapisa onemogućuju:

1) Utvrđivanje počinitelja određene aktivnosti – ne može se dokazati da je korisnik izvršio određenu akciju, osim ukoliko sam ne prizna.
2) Utvrđivanje o počinjenim zlouporabama – napadač može počiniti sumnjive aktivnosti koje neće biti zabilježene. Čak i ako su radnje zabilježene, dnevnički zapisi mogu biti slabo zaštićeni što opet omogućuje brisanje podataka.

3.2. Smjernice sigurnog programiranja

Jednom kada je programer web aplikacije upoznat s mogućim prijetnjama i zlouporabama, može razviti strategiju sigurnog programiranja kojim će ukloniti ili minimalizirati ranjivosti. U ovom odjeljku daju se smjernice za sigurno programiranje koje se temelje na opasnostima navedenim u prethodnom odjeljku.

3.2.1. Provjera ulaznih podataka

Temeljita provjera ulaznih podataka uključuje:

- Provjeru tipa, dužine, formata i dosega podataka
- Ukoliko se pozivaju funkcije koje same ne vrše provjere, potrebno je osigurati da će im se kao parametri predati isključivo očekivani tipovi podataka.
- Posebno je važno provjeriti odgovaraju li ulazni kolačići, upiti i HTML obrasci aplikaciji. Za ovu vrstu provjere preporuča se korištenje regularnih izraza.
- Ukoliko izlazne vrijednosti stranice uključuju korisničke unose preporuča se kodirati ih *HTMLEncode* ili *URLEncode* funkcijama jer će one izvršne skripte pretvoriti u bezopasni HTML.
- Korištenje */GS* zastavice prilikom prevođenja C++ koda uključuje sigurnosne provjere koje isključuju mogućnost uobičajenih pojava prepisivanja spremnika
- Kod SQL ulaznih podataka važno je provjeriti da upiti ne sadrže izraze koji se tretiraju kao izvršne naredbe. Preporuča se korištenje pohranjenih procedura ili SQL parametara prilikom izgradnje SQL naredbi. Osim toga preporuča se i korištenje korisničkih računa s minimalnim potrebnim ovlastima.
- Nije preporučljivo dozvoliti korisniku definiranje putanja do datoteka i URL zahtjeva. Poželjno je rabiti apsolutne putanje koje nisu podložne korisničkim izmjenama. Ukoliko se ipak dozvoli korisnički unos, potrebno je provjeriti njegovu ispravnost i smislenost u okviru aplikacije te provjeriti ispravnost kodiranja kako bi se onemogućili različiti prikazi putanja do datoteka. To se može obaviti pomoću *requestEncodnig* i *requestDecoding* atributa.

3.2.2. Zaštita osjetljivih podataka

Zaštita osjetljivih podataka uključuje enkripciju i strogu kontrolu pristupa. Preporučeni mehanizam kontrole pristupa su ACL (eng. Access Control List) liste dozvola. One definiraju tko smije pristupati objektu i što smije s njim činiti. Osim jake kontrole pristupa poželjno je kriptirati osjetljive podatke u bazama podataka i konfiguracijskim datotekama.

Kako bi izbjegli napade iz zasjede, nužno je definirati dozvole pristupa programima čije bi pokretanje moglo dovesti do otkrivanja osjetljivih podataka. .NET Framework ugrađene

provjere prava pristupa autoriziraju pozivajući program svaki put kada se pristupa zaštićenim podacima ili se pozivaju zaštićene metode koje rukuju njima.

Mjere zaštite priliko slanja osjetljivih podataka putem mreže su enkripcija podataka i korištenje MAC (eng. Message Authentication Code) ili HMAC (eng. Hashed Message Authentication Code) zaštitne sume koje osiguravaju autentičnost i integritet poruke. Posebno osjetljive podatke kao što su lozinke preporuča se ne slati putem Interneta. Mehanizmi koji omogućavaju ovjeru identiteta bez slanja lozinki su Kerberos protokol i Windows provjera ovlasti. Ukoliko se lozinke ipak prenose mrežom obavezno je korištenje sigurnih komunikacijskih protokola kao što je SSL (eng. Secure Sockets Layer) protokol. Zaštita lozinki uključuje:

- Korištenje snažnih, složenih lozinki koje se sastoje od nepredvidljivih pojmova te kombiniraju mala i velika slova, brojeve i posebne znakove.
- Korištenje jednosmjernih funkcija za transformaciju i pohranu lozinki u memoriji.
- Zaključavanje korisničkog računa nakon nekoliko neuspješnih pokušaja prijave u sustav.
- Postaviti korisniku kao izbor korištenje, odnosno nekorištenje, prijave na sustav preko priručne memorije web preglednika, ili tu mogućnost potpuno isključiti.

Budući da je kriptografija čest mehanizam zaštite podataka u računarstvu, važno je rabiti odgovarajuće i sigurne algoritme. Razvoj vlastitih kriptografskih algoritama ne preporuča se jer nepotrebno uzima vrijeme i teško da će nadmašiti već postojeće optimalne metode. Umjesto toga poželjno je koristiti ugrađene mehanizme operacijskog sustava koji uključuju i zaštitu ključeva. Na Windows operacijskim sustavima to je DPAPI (eng. Data Protection Application Programming Interface). Sigurno upravljanje kriptografskim ključevima uključuje:

- Korištenje snažnih funkcija za generiranje slučajnih vrijednosti ključeva.
- Pohrana ključeva u datoteke zaštićene ACL mehanizmom.
- Korištenje DPAPI sustava kao dodatne zaštite ključeva.
- Redovita izmjena vrijednosti ključeva

3.2.3. Sigurno upravljanje sjednicama i konfiguracijom aplikacije

Sigurno upravljanje sjednicama podrazumijeva zaštitu kolačića za ovjeru identiteta korisnika na računalnoj mreži. Siguran prijenos postiže se korištenjem SSL i HTTPS protokola. Kratki vremenski razmaci u kojima se prihvaća zahtjev bez ponovne provjere identiteta smanjuju napadaču vjerojatnost hvatanja kolačića u prometu i lažnog predstavljanja.

Ukoliko korisnik otvara novu sjednicu nakon odjave trenutne, sustav mora zahtijevati ponovnu provjeru identiteta. Također, ponovnu prijavu potrebno je tražiti prije pokretanja osobito osjetljivih radnji, kao što su npr. novčane transakcije. Također poželjno je onemogućiti čuvanje podataka o sjednici u klijentskom programu.

Zaštita konfiguracijskih datoteka web aplikacije uključuje pohranu tih podataka izvan web prostora kako bi se napadaču onemogućio pristup njima putem Interneta. Također se predlaže korištenje ACL lista za upravljanje pravima pristupa. Posebno osjetljive podatke kao što su lozinke potrebno je dodatno zaštititi kriptografskim metodama. Također, aplikacije i usluge ne bi trebale imati mogućnost mijenjanja vlastitih konfiguracijskih podataka ukoliko to nije nužno.

Zajednički korisnički računi su loša praksa jer onemogućuju praćenje aktivnosti pojedinog korisnika. Zato je osobito važno da administratorski računi nikada ne budu dijeljeni. Broj administratorskih sučelja valja minimizirati i, ukoliko je moguće, dopustiti im samo lokalno pristupanje. Za udaljeno pristupanje administratorskim sučeljima savjetuje se korištenje VPN (eng. Virtual Private Network) tehnologije ili SSL protokola.

3.2.4. Sigurno upravljanje parametrima i iznimkama

U prethodnom poglavlju navedeni su načini na koje zlonamjeran korisnik može oblikovati pojedine parametre i narušiti sigurnost web aplikacije. Kako bi se umanjili rizici savjetuje se izbjegavanje donošenja odluka koje utječu na sigurnost na temelju vrijednosti parametara. To se odnosi na HTML obrasce i HTTP zaglavlja te znakovni nizovi koji predstavljaju upite. Umjesto osjetljivih podataka u parametrima preporuča se njihovo pohranjivanje zajedno s podacima o sjednici i korištenje identifikacijskih atributa sjednice za identifikaciju klijenta.

Kako bi se izbjegle ranjivosti vezane uz loše upravljanje iznimkama potrebno je dobro provjeriti sve ulazne podatke i uvesti upravljanje iznimkama na svim dijelovima programa. Sve poruke o iznimkama koje dospijevaju do klijenta potrebno je obraditi i oblikovati u općenite i napadaču neiskoristive obavijesti.

3.2.5. Praćenje korisničkih aktivnosti

Uspješno praćenje korisničkih aktivnosti omogućuje povezivanje počinitelja s radnjom i prepoznavanje pokušaja narušavanja sigurnosti sustava prije nego zlouporaba doista uspije. Siguran sustav za praćenje korisničkih aktivnosti podrazumijeva:

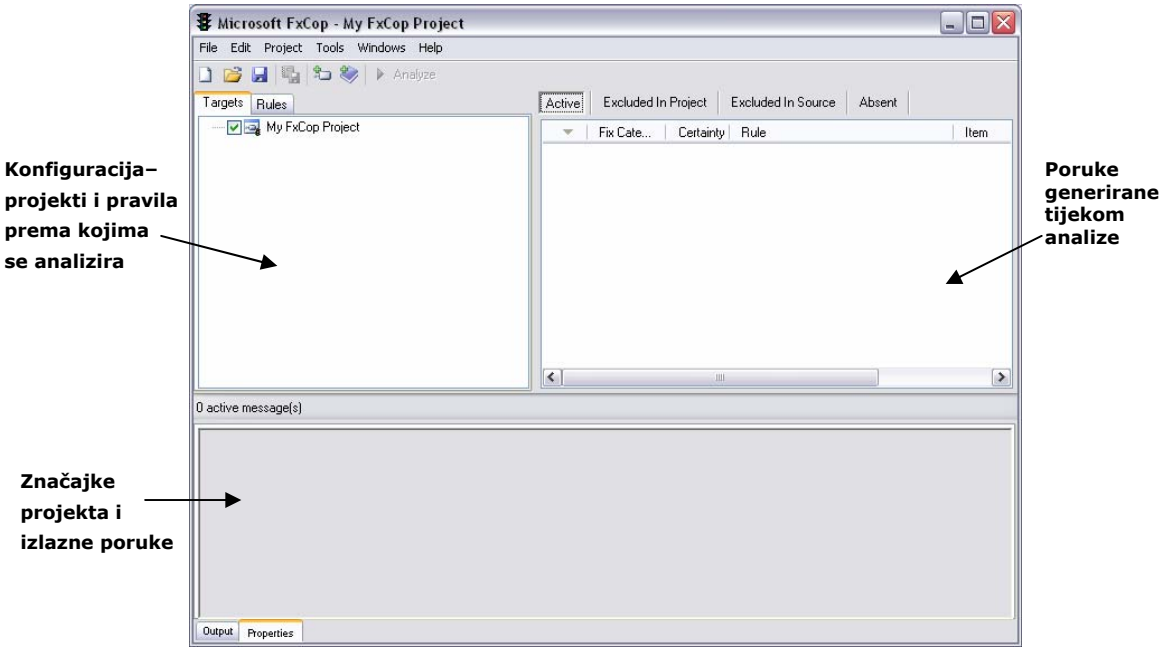
- Praćenje svih aktivnosti na web poslužitelju, poslužitelju baze podataka i na aplikacijskom poslužitelju, ukoliko se koristi.
- Praćenje aktivnosti na razini operacijskog sustava kako bi se zabilježili važni događaji kao što su prijave i odjave korisnika na sustav, pristupanje datotečnom sustavu i neuspjeli pokušaji pristupanja pojedinim objektima.
- Izbjegavanje uporabe dijeljenih korisničkih računa jer se onemogućuje identificiranje korisnika.
- Dnevnički zapisi osjetljivi su podaci pa treba koristiti ACL liste za njihovu zaštitu i smjestiti ih na zaštićene lokacije u memoriji
- Stvaranje sigurnosnih kopija dnevničkih zapisa i redovito provjeravanje kako bi se na vrijeme uočili pokušaji napada na sustav.

3.3. Aplikacije za provjeru programskog koda

Ukoliko programer razvija programsku potporu vodeći računa o mogućim ranjivostima, neke će od njih ukloniti već u prvoj inačici programa. No, previdi i propusti u programiranju redovna su pojava pa je svaki program potrebno provjeriti i prepraviti više puta. Pažljivo i sistematizirano programiranje smanjuje količinu posla utrošenog u ispravljanje programa i olakšava taj postupak, ali ga ne može potpuno zamijeniti.

Dorada programskog koda uključuje traženje propusta i njihovo ispravljanje. Budući da postoje programski alati koji olakšavaju ovaj postupak, preporuča se njihova primjena.

3.3.1. FxCop



Slika 1. Korisničko sučelje FxCop programa

FxCop je programski alat za analizu izvršnog koda. Pomoću njega je moguće utvrditi je li program oblikovan u skladu sa zahtjevima kvalitetnog programiranja definiranim za .NET razvojno okružje. Ti se zahtjevi, između ostalog, odnose i na:

- Izbor imena projekata, parametara, tipova i dr. - dobar odabir imena osigurava lakše upravljanje programskim sredstvima i smanjuje mogućnost pojave višeznačnosti.
- Upotrebu statičkih i apstraktnih klasa, sučelja, nabrajanje (eng. enumeration) i struktura.
- Oblikovanje svojstava, konstruktora, metoda, događaja i operatora.
- Upravljanje posebnim događajima – oblikovanje, dojavu i rukovanje posebnim događajima.
- Oblikovanje programskih biblioteka koje se jednostavno mogu nadograđivati.
- Upotrebu polja i atributa.

Osim navedenih zahtjeva FxCop provjerava koriste li datoteke izvršnog koda snažna imena. To su imena zaštićena kriptografskim ključem koja doprinose očuvanju tajnosti i integriteta podataka. Primjer 8 sadrži izlaznu poruku vezanu uz nedostatak snažnog imena u analiziranom programu *program.exe*. U poruci se navode mogući razlozi ove pogreške:

- sadržaj datoteke mijenjan je nakon dodjele imena
- dodjela imena nije uspjela
- dodijeljen je samo javni ključ.

```
CriticalError, Certainty 95, for AssembliesShouldHaveValidStrongNames
{
Target     : program.exe  (IntrospectionTargetModule)
Resolution  : "Sign 'program.exe' with a strong name key."
```

```
Help       : http://msdn2.microsoft.com/ms182127(VS.90).aspx  (String)
Category   : Microsoft.Design  (String)
CheckId    : CA2210  (String)
RuleFile   : Design Rules  (String)
Info       : "Either the assembly has no strong name, an invalid
one, or the strong name is valid only because of the
computer configuration. The assembly should not be
deployed in this state. The most common causes of this
are: 1) The assembly's contents were modified after
it was signed. 2) The signing process failed. 3) The
assembly was delay-signed. 4) A registry key existed
that allowed the check to pass (where it would not
have otherwise)."
Created    : 28.5.2008 8:12:49  (DateTime)
LastSeen   : 28.5.2008 8:12:49  (DateTime)
Status     : Active  (MessageStatus)
Fix Category : NonBreaking  (FixCategories)
}
```

Primjer 8. Primjer izlazne poruke FxCop analize

3.3.2. Pretraživanje teksta

Pretraživanje ključnih riječi u kodu jednostavan je ali učinkovit način pronalaženja velikog broja mogućih ranjivosti. Visual Studio .NET ima ugrađenu mogućnost „Find in Files“ za pretraživanje programskog koda. Druga je mogućnost korištene alata „Findstr“ koji se koristi putem naredbenog retka (eng. command prompt), a dio je Windows operacijskog sustava. Neke od mogućnosti Findstr alata su navedene u tablici Tablica 1. Primjer korištenja naredbe

```
findstr /N „lozinka“ *.aspx
```

pronalazi sva mjesta pojave niza „lozinka“ u svim „.aspx“ datotekama u trenutnom direktoriju. Primjeri nizova koji otkrivaju mjesta mogućih ranjivosti su:

- „System.Runtime.InteropServices“ – nalazi se kod poziva nenadgledanog programskog koda, mjestima koja predstavljaju mogući rizik od pojave prepisivanja spremnika.
- „SQLCommand“,“OleDbCommand“,“OdbcCommand“ – ukoliko se SQL naredbe tvore uz pomoć korisničkih unosa, moguće je ubacivanje SQL koda.
- „.Write“ – ako naredba ispisuje izlazne podatke, a uključuje i neobrađene korisničke unose, predstavlja moguću XSS ranjivost.

Mogućnost	Opis
/S	Uključuje poddirektorije u pretragu
/M	Ispisuje samo imena datoteka
/I	Uključuje neosjetljivost na veličinu slova (eng. case insensitive)
/N	Ispisuje poziciju retka u datoteci kod svakog pojavljivanja traženog niza

Tablica 1. Mogućnosti *findstr* naredbe

Uz *findstr* naredbu operacijskog sustava, Visual Studio uključuje i program **ildasm.exe** za pretraživanje znakovnih nizova u binarnim datotekama koje sadrže izvršni kod. Ildasm.exe pretvara PE (eng. Portable Executable) binarne datoteke s MSIL (eng. Microsoft Intermediate Language) asemblerskim kodom i pretvara ih u tekstualne datoteke. Nakon toga moguće ih je pretraživati pomoću *findstr* alata. Uključene su datoteke „.exe“, „.dll“, „.obj“, i „.lib“, a neke od dostupnih mogućnosti navedene su u tablici Tablica 2.

Mogućnost	Opis
/output = *filename*	Umjesto ispisa rezultata na korisničkom sučelju stvara izlaznu datoteku filename
/rtf	Izlazni podaci su u RTF (eng. Rich Text Format) formatu
/text	Ispisuje rezultat u naredbenom retku.
/html	Izlaz je u HTML formatu (koristi se jedino uz /output mogućnost)
/?	Ispisuje sintaksu i mogućnosti alata

Tablica 2. Mogućnosti ildasm.exe alata

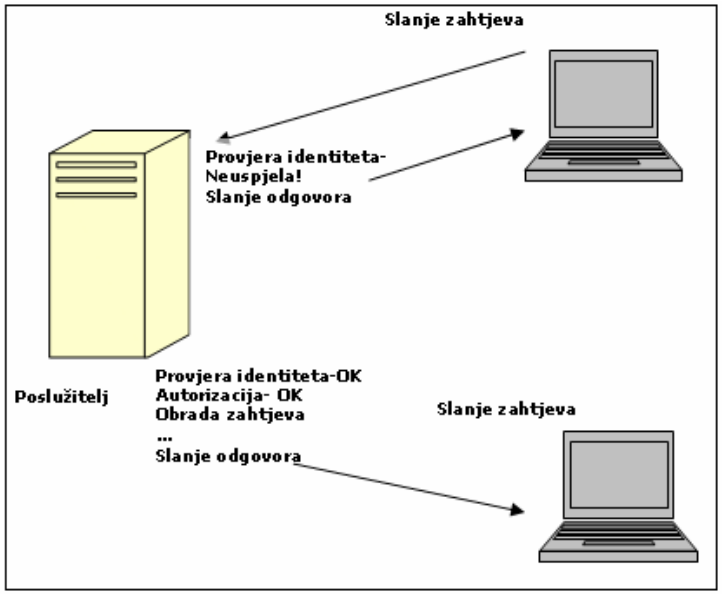
U primjeru zajedničkog korištenja *findstr* i ildasm.exe naredbi (Primjer 9) prvo se sadržaj datoteke aplikacija.dll pretvara u tekst. Zatim se u tekstu pretražuju svi reci koji sadrže MSIL naredbu ldstr (eng. load string on the stack) koja sprema znakovne nizove na programski stog. Na ovaj način otkriva se postupak povezivanja s bazom podataka i lozinka dobro poznatog korisničkog računa *korisnik*.

```
Ildasm.exe aplikacija.dll /text | findstr ldstr
IL_000c: ldstr "RegistriraniKorisnik"
IL_0027: ldstr "@korisnickoIme"
IL_0046: ldstr "@lozinkaHash"
IL_0065: ldstr "@salt"
IL_008b: ldstr "Pogreska prilikom dodavanja racuna. "
IL_000e: ldstr "potraziKorisnika"
IL_0027: ldstr "@KorisnickoIme"
IL_007d: ldstr "SHA1"
IL_0097: ldstr "Pogreska prilikom provjere lozinke. "
IL_0009: ldstr "SHA1"
IL_003e: ldstr "Prijava uspjela: Korisnik je prepoznat"
IL_0050: ldstr "Pogresno ime ili lozinka"
IL_0001: ldstr "Server=AppServer;database=users;
username='korisnik' password=lozinka"
```

Primjer 9. Pretraživanje MSIL koda pomoću *findstr* i ildasm.exe naredbi

4. Oblikovanje poslužiteljskih i klijentskih aplikacija

Komunikacija putem računalne mreže najčešće se oblikuje na temelju poslužitelj-klijent modela (Slika 2). Poslužiteljska aplikacija ona je strana modela koja osluškuje i izvršava zahtjeve, a klijenti ih šalju. Ako poslužitelj prihvati primljeni zahtjev, pristupa njegovoj obradi i na kraju klijentu šalje odgovor. Time je komunikacija završena dok klijent ne pošalje novi zahtjev.



Slika 2. Klijent/poslužitelj arhitektura

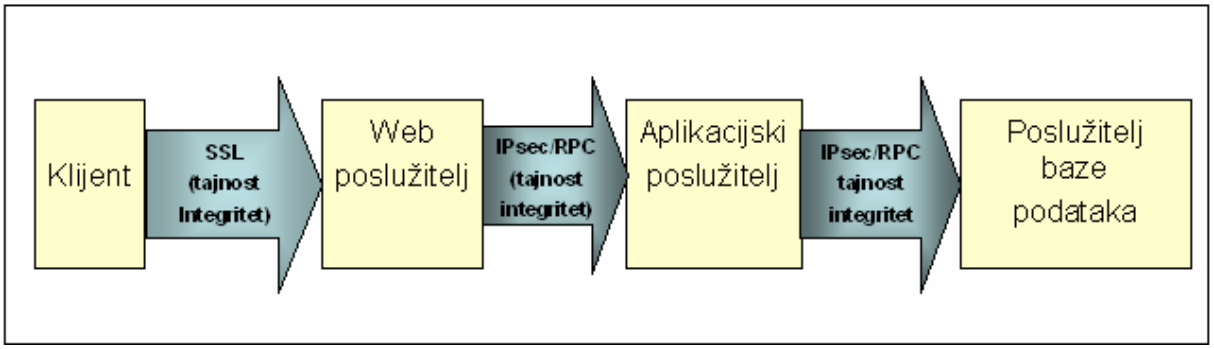
Među protokolima koji koriste ovaj modelu je i HTTP protokol. Web preglednik je u ovom slučaju klijentska aplikacija koja šalje URL zahtjeve pojedinim HTTP poslužiteljima i prikazuje odgovore krajnjem korisniku prikazuje u odgovarajućem obliku, npr. web stranici. Ovo poglavlje daje smjernice za uspostavljanje sigurnih klijentskih i poslužiteljskih web usluga unutar ASP.NET razvojnog okružja.

.NET razvojno okružje podržava COM (eng. Component Object Model) oblikovanje koje se zasniva na ideji izgradnje programa od nezavisnih dijelova. Prilikom korištenja gotovih komponenti nije potrebno znati njihovu unutrašnju izvedbu, a upotrebljive su i u okruženjima različitim od onih u kojima su razvijene. Nezavisni programski dijelovi oblikovani prema ovom modelu nazivaju se COM sastavnice. Na ovaj način ubrzava se postupak izgradnje aplikacija jer se jednom oblikovane programske sastavnice mogu koristiti neograničeno mnogo puta (eng. reusability).

.NET okružje sadrži ugrađena COM rješenja tradicionalnih usluga kao što su upravljanje istodobnim radnjama, raspodijeljenim transakcijama, prozivanje objekata i dr. Ona su dostupna unutar *EnterpriseServices.ServicedComponent* osnovne klase. Osobitosti sigurne – *EnterpriseServices* arhitekture su:

- Upotreba RPC (eng. Remote Procedure Call) tehnologije pri ovjeri identiteta korisnika koja osigurava upotrebu Kerberos ili NTLM (eng. NT Lan Manager) protokola. Ukoliko se provjera želi izbjeći programer je mora sam isključiti.
- Autorizacija korisnika obavlja se kroz COM+ uloge definirane unutar COM+ kataloga, a kojima se upravlja pomoću *Component Services* alata.
- Kada ASP.NET aplikacija poziva metode *ServicedComponent* klase, obavlja se provjera identiteta i ovlasti pripadne uloge. Ukoliko je pozivatelj ovlašten za korištenje usluge, ona se izvršava, inače se poziv odbacuje. Moguće je definirati i sigurnosni događaj koji će zabilježiti neuspješni pokušaj neovlaštenog korištenja usluge.
- Za zaštićenu DCOM (eng. Distributed COM) komunikaciju između klijenta i poslužitelja koristi se RPC Packet Integrity ili RPC Packet Privacy razina ovjere identiteta za očuvanje tajnosti i integriteta podataka.

Sigurna komunikacija klijentskih i poslužiteljskih aplikacija (Slika 3) uključuje korištenje SSL protokola koji čuva tajnost i integritet podataka prilikom prijenosa kroz računalnu mrežu. Klijent koji koristi SSL rabi HTTPS protokol i definira „https“ URL zahtjev, a poslužitelj osluškuje zahtjeve na portu 443. SSL također zahtjeva korištenje certifikata kao jamstva sigurnosti poslužiteljskih aplikacija.



Slika 3. Udaljena komunikacija poslužiteljskih i klijentskih aplikacija

IPsec protokol koristi provjeru identiteta korisnika i enkripciju podataka, čime čuva njihov integritet i tajnost. Ne može se programski upravljati njegovim postavkama, a ukoliko se koristi dostupni su IPsec filtri za kontrolu prometa.

4.1. Poslužiteljska aplikacija

Ukoliko aplikacija zahtjeva provjeru identiteta korisnika poželjno ju je ostvariti kao poslužiteljski program. Svojstvo se definira dodavanjem atributa:

```
[assembly:ApplicationActivation(ActivationOption.Server)]
```

Razinu ovjere identiteta za poslužiteljske aplikacije definira programer, dok se kod bibliotečnih aplikacija ona nasljeđuje od klijentskog procesa.

Deklarativno uspostavljanje ovjere identiteta korisnika za aplikaciju može se obaviti postavljanjem *ApplicationAccessControl* atributa na sljedeći način:

```
[assembly: ApplicationAccessControl( Authentication = AuthenticationOption.Call)]
```

Omogućavanje autorizacije korisnika na razini aplikacije obavlja se postavljanjem vrijednosti atributa *ApplicationAccessControl* na *true*:

```
[assembly: ApplicationAccessControl(true)]
```

Izvršavanjem sljedeće naredbe omogućuje se provjera prava pristupa na razini procesa i komponenti:

```
[assembly: ApplicationAccessControl(AccessChecksLevel=
AccessChecksLevelOption.ApplicationComponent)]
```

Za bibliotečne i poslužiteljske programe preporuča se dodjela uloga na razini sučelja, klasa i metoda. Dodavanje nove uloge *zaposlenik* na razini aplikacije moguće je izvršiti naredbom:

```
[assembly:SecurityRole("zaposlenik")]
```

Ukoliko se radi o ulozi klase postupak je sljedeći:

```
[SecurityRole("zaposlenik")]
public class Tvrtka :
ServicedComponent
{
}
```

Dodatno, označavanje klase atributom [*SecureMethod]* omogućuje uspostavljanje uloga pomoću ugrađenih alata:

```
[ComponentAccessControl]
[SecureMethod]
public class Tvrtka :
ServicedComponent
{
}
```

ServicedComponent dinamičke biblioteke poslužiteljske aplikacije zahtijevaju registraciju u globalnoj priručnoj memoriji izvršnog koda. U tu svrhu koristiti se alat „gacutil.exe".

```
gacutil –i MyServicedComponent.dll
```

Registracija u COM+ katalogu stvoriti će COM+ aplikaciju i omogućiti uređivanje uloga pomoću *ServicedComponent* ugrađenih alata.

```
regsvcs.exe MyServicedComponent.dll
```

4.2. Klijentska aplikacija

Uspostava komunikacije između klijentske aplikacije i COM komponenti ostvaruje se pomoću DCOM tehnologije. Riječ je o izvedbi komunikacije između zaštićenih dijelova programske potpore koji se nalaze na različitim umreženim računalima. Oblikovanje sigurnog klijentskog programa uključuje postavljanje razine vrijednosti atributa *comAuthenticationLevel* na neku od sljedećih vrijednosti:

```
comAuthenticationLevel= "[Default|None|Connect|Call|Pkt|PktIntegrity|PktPrivacy]"
```

Atribut se nalazi u konfiguracijskoj datoteci

%windir%\Microsoft.NET\Framework\v1.0.3705\CONFIG\Machine.config

, a kratki opis različitih vrijednosti u tablici Tablica1. Ukoliko klijent i poslužitelj imaju postavljene različite razine ovjere identiteta, primjenjuje se ona s višim sigurnosnim zahtjevima.

Razina ovjere identiteta	Opis
Default	Provode se uobičajeni koraci ovjere identiteta
None	Ovjera identiteta korisnika se ne provodi
Connect	Provjeri identitet samo kod uspostave veze
Call	Provjeri identitet kod svakog udaljenog poziva procedure (RPC)
Pkt	Provjeri sve primljene pakete
PktIntegrity	Provjeri integritet svih primljenih podataka
PktPrivacy	Provjeri tajnost svih primljenih podataka

Tablica 3. Opis različitih razina provjere identiteta korisnika

Osim uspostave razine ovjere identiteta korisnika, klijentska aplikacija mora odrediti i dopuštenu razinu oponašanja identiteta (eng. impersonation). Riječ je o svojstvu dretve (eng. thread) koje joj omogućuje izvođenje u sigurnosnim uvjetima različitim od roditeljskog procesa. Dretva preuzima identifikacijska svojstva klijenta kako bi imala pristup istim računalnim sredstvima kao i klijent. Najčešće se koristi za provjeru prava pristupa klijenta ili u slučajevima kada jedna COM sastavnica pristupa drugoj COM sastavnici unutar strane poslužiteljske aplikacije. Dopušteni način oponašanja identiteta, definiran u klijentskoj aplikaciji, određuje mogućnosti oponašanja dostupne poslužiteljskoj aplikaciji. U već spomenutoj datoteci „Machine.config“ definira se vrijednost atributa *comImpersonationLevel*.

```
comImpersonationLevel= "[Default|Anonymous|Identify|Impersonate|Delegate]"
```

Od navedenih vrijednosti atributa „Anonymous“ i „Default“ nisu dostupne, a kratki opisi preostalih mogućnosti nalaze se u tablici Tablica 2.

Razina oponašanja identiteta	Opis
Identify	Omogućuje poslužitelju provjeru identiteta klijenta i provjeru njegovih prava pristupa oponašanjem
Impersonate	Poslužitelj može pristupati lokalnim računalnim sredstvima koristeći identifikacijsku oznaku klijenta
Delegate	Poslužitelj može pristupati udaljenim računalnim sredstvima koristeći identifikacijsku oznaku klijenta. Ova mogućnost zahtjeva korištenje protokola Kerberos i posebno oblikovanje korisničkog računa.

Tablica 4. Opis različitih razina oponašanja identiteta klijenta

U .NET razvojnom okruženju klijentski identitet pretpostavlja se unutar *CoImpersonateClient* i *CoRevertToSelf* poziva, a prosljeđuje se i svim pozivima smještenim unutar tog bloka (eng. dynamic cloaking).

5. Zaključak

ASP.NET razvojno je okruženje za izgradnju interaktivnih i dinamičnih web stranica. Pri oblikovanju web aplikacija podjednako je važno voditi računa o uspješnosti izvedbe zadanih usluga i o sigurnosti korištenja. Razvoj sigurne aplikacije podrazumijeva poznavanje ranjivosti i mogućih zlouporaba na temelju kojih se poduzimaju mjere zaštite. Kod web usluga to je prvenstveno zaštita osjetljivih računalnih sredstava, podataka i metoda. Osjetljivi podaci uključuju korisnička imena i lozinke, konfiguracijske podatke, identifikacijske atribute sjednica i dr. Oni se osiguravaju kriptografskom zaštitom i pohranjivanjem na sigurnim mjestima te provjerom identiteta korisnika i njegovih prava pristupa računalnim sredstvima. Izrada sigurnih aplikacija uključuje i kriptografsku zaštitu podataka u prijenosu čiji je cilj osigurati njihovu tajnost i integritet. Osim toga potrebno je pratiti korisničke aktivnosti kako bi se na vrijeme uočili pokušaji napada na sustav i omogućile pravovremene reakcije.

Nakon što je napisan, program je potrebno pregledati jer je moguća pojava previda i pogrešaka. Traženje problematičnih dijelova koda moguće je izvesti pomoću alata za pretraživanje teksta, ili pomoću programa koji ispituju zadovoljava li programsko oblikovanje određene standarde. Takav je program FxCop, koji provjerava je li aplikacija oblikovana u skladu sa zahtjevima definiranim u okviru .NET razvojnog okružja.

Na kraju dokumenta dan je uvid u metode sigurnog oblikovanja klijentskih i poslužiteljskih sastavnica na temelju *EnterpriseServices.ServicedComponent* osnovne klase. Riječ je o ASP.NET izvedbi COM modela koji omogućuje izgradnju nezavisnih, zatvorenih programskih dijelova. Njih je nakon izgradnje moguće koristiti za oblikovanje drugih programa, a da pritom nije potrebno znate detalje izvedbe same sastavnice. Time je zaokružen pregled sigurnosnih propusta i odgovarajućih protumjera vezanih uz ASP.NET oblikovanje.

U usporedbi s alternativnom PHP tehnologijom za izgradnju web stranica, ASP.NET oblikovanju najčešće se zamjera cijena, nemogućnost korištenja na Unix/Linux operacijskim sustavima i zahtjevnost koda u pogledu vremena izvršavanja i upotrebe memorije. Što se sigurnosti tiče, na razini programskog koda obje tehnologije pružaju podjednake mogućnosti zaštite. Ipak, kad je riječ o sigurnosti velikih baza podataka, ASP.NET nudi više mogućnosti zaštite. Zato se kod web aplikacija koje rade s velikim bazama podataka preporuča primjena ASP.NET tehnologije.

6. Reference

[1] Microsoft: Building Secure ASP.NET Applications Authentication, Authorization, and Secure Communication, http://msdn2.microsoft.com/en-us/library/aa302415.aspx, svibanj 2008.

[2] Microsoft: Improving Web Application Security Threats and Countermeasures, http://msdn2.microsoft.com/en-us/library/ms994921.aspx, svibanj 2008.

[3] http://en.wikipedia.org/wiki/ASP.NET, svibanj 2008.

[4] ASP.NET – Web pages, http://www.w3schools.com/aspnet/aspnet_pages.asp, svibanj 2008.