Osnove informacione bezbednosti u infrastrukturnim sistemima

Predmetni projekat

Servis za nadzor softvera

Tim 1:

- PR 1/2020 Zlatko Čikić
- PR 26/2020 Bojana Bratić
- PR 34/2020 Ilija Spasić
- PR 56/2020 Dušan Borovićanin

Sadržaj

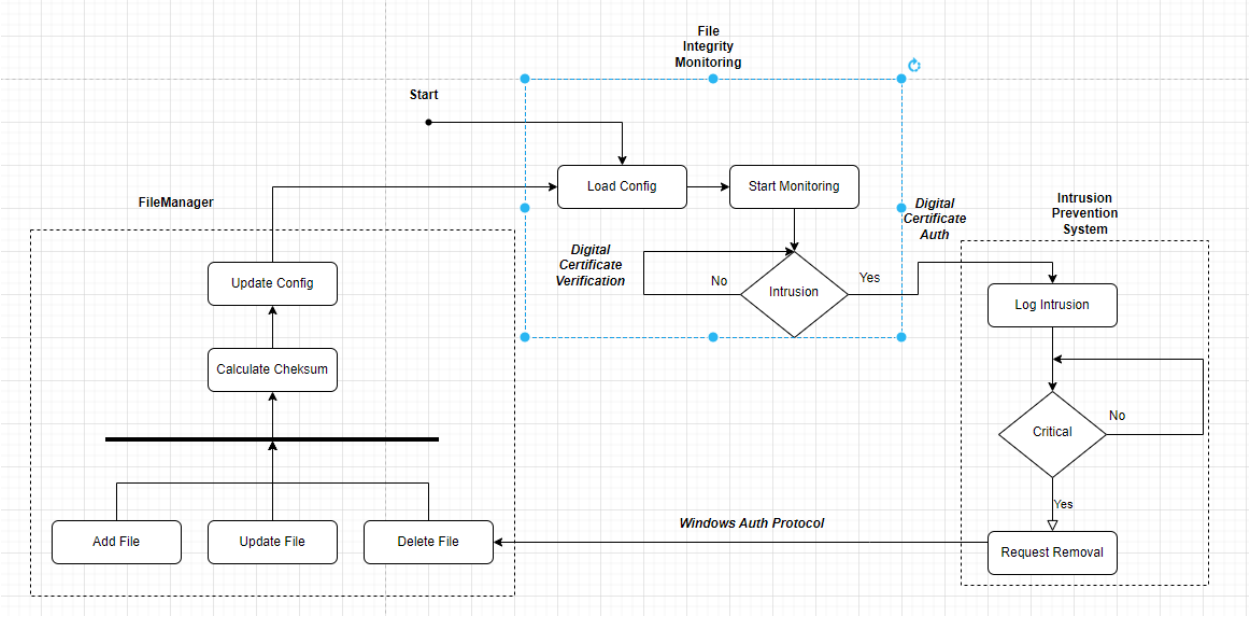
Sadržaj **2**
Osnovne komponente **4**
Fim servis 4
IPS servis 4
FM servis 5
Client 5
FIM servis **6**
Konfiguracija servisa 6
ConfigurationManager 8
CreateConfig 8
Metode ConfigurationManager-a 9
FileIntegrityMonitoring 10
StartMonitoring 10
FileIntegrityMonitoringService 11
AddFile metoda 11
UpdateFile metoda 12
RemoveFile metoda 12
ReadFile metoda 13
ReadFileNames metoda 13
IPS servis **14**
Konfiguracija servisa 14
IntrusionPreventionService 15
LogIntrusion metoda 15
Audit 16
LogIntrusion metoda 16
FM servis **17**
Konfiguracija servisa 17
FileManagerService 18
RequestRemoval metoda 19
AddFile metoda 19
ReadFile metoda 20
UpdateFile metoda 20
ReadFiles metoda 21

Client **22**
Konfiguracija klijenta 23
AddFile metoda 23
UpdateFile metoda 24
Common - pomoćne klase **26**
CustomConsolePrint 26
CustomException 27
Intrusion 27
IFile - MonitoredFile 28
TripleDesAlgorithm 29

Osnovne komponente

Servis za nadzor softvera se sastoji iz tri servisa i klijentske aplikacije :

1. FIM (File integrity monitoring)
2. IPS (Intrusion prevention system)
3. FM (File manager)
4. Client



Fim servis

FIM servis vrši sam nadzor datoteka i proveru njihovih integriteta na podešeni period. Pored samog nadzora ima CRUD funkcionalnosti za pomenute datoteke koje autorizovani klijenti mogu koristiti.

IPS servis

IPS servis kao glavnu ulogu ima vođenje evidencije otkrivenih problema poput narušavanja integriteta neke od datoteka koje FIM servis nadgleda. Evidencija se čuva u Windows Logs.

Ukoliko se detektuje problem nivoa “Critical” IPS kreira zahtev za brisanje datoteke i prosleđuje isti FM servisu.

FM servis

FM servis predstavlja proxy namenjen klijentima za komunikaciju sa FIM servisom. Takođe omogućava brisanje datoteke na zahtev IPS servisa.

Client

Klijentska aplikacija pruža mogućnost dodavanja nove datoteke ili čitanja i uređivanja postojeće datoteke. U samoj aplikaciji postoji **ConsoleEditor** koji pruža mogućnost uređivanja datoteke u samoj konzoli aplikacije.

FIM servis

Konfiguracija servisa

Konfiguracija servisa sadrži sledeća podešavanja:

```
<add key="MonitoredPath" value="MonitoredFiles"/>
<add key="srvCertCN" value="ipsservice"/>
<add key="signCertCN" value="FIMCert"/>
<add key="MonitoringPeriod" value="5000"/>
<add key="ConfigFile" value="config.xml"/>
<add key="ipsAddress" value="net.tcp://localhost:6002/IIntrusionPreventionSystem"/>
```

- **MonitoredPath** - putanja do foldera koji servis nadgleda
- **srvCertCN** - CommonName sertifikata za komunikaciju sa IPS servisom
- **signCertCN** - CommonName sertifikata za digitalno potpisivanje datoteka
- **MonitoringPeriod** - perioda provere integriteta nadgledanih datoteka
- **ConfigFile** - datoteka sa zapisima nadgledanih datoteka i njihovih digitalnih potpisa
- **ipsAddress** - adresa IPS servisa

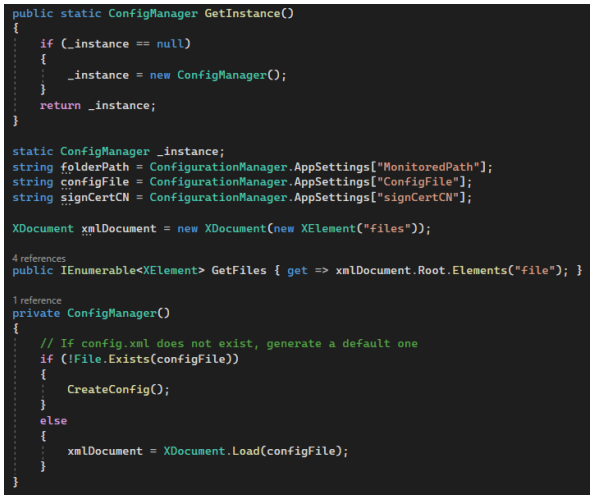
Podešavanja samog servisa:

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceDebug includeExceptionDetailInFaults="true" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <services>
    <service name="FileIntegrityMonitoringProject.FileIntegrityMonitoringService">
      <host>
        <baseAddresses>
          <add baseAddress="net.tcp://localhost:6000"/>
        </baseAddresses>
      </host>
      <endpoint address="IFileIntegrityService"
                        binding="netTcpBinding"
                        bindingConfiguration="WinConfig"
                        contract="Common.IFileIntegrityService"
                        />
    </service>
  </services>
  <bindings>
    <netTcpBinding>
      <binding name="WinConfig">
        <security mode="Transport">
          <transport clientCredentialType="Windows" />
        </security>
      </binding>
    </netTcpBinding>
  </bindings>
</system.serviceModel>
```

```
1 reference
public FileIntegrityMonitoring()
{
    /// Create private key
    SymmetricAlgorithm symalg = TripleDESCryptoServiceProvider.Create();
    key = Encoding.ASCII.GetString(symalg.Key);

    /// cltCertCN.SubjectName should be set to the client's username. .NET WindowsIdentity class provides information about Windows user running the given process
    string cltCertCN = Formatter.ParseName(WindowsIdentity.GetCurrent().Name);

    certificateSign = CertManager.GetCertificateFromStorage(StoreName.My, StoreLocation.LocalMachine, signCertCN);

    if(certificateSign == null)
    {
        CustomConsole.WriteLine("No certificate for signing found", MessageType.Error);
        throw new Exception("No certificate for signing found");
    }

    NetTcpBinding binding = new NetTcpBinding();
    binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.Certificate;

    /// Use CertManager class to obtain the certificate based on the "srvCertCN" representing the expected service identity.
    X509Certificate2 srvCert = CertManager.GetCertificateFromStorage(StoreName.TrustedPeople, StoreLocation.LocalMachine, srvCertCN);
    EndpointAddress address = new EndpointAddress(new Uri(ipsAddress),
                              new X509CertificateEndpointIdentity(srvCert));
    channelFactory = new ChannelFactory<IIntrusionPreventionSystem>(binding, address);
    channelFactory.Credentials.ServiceCertificate.Authentication.CertificateValidationMode = System.ServiceModel.Security.X509CertificateValidationMode.Custom;
    channelFactory.Credentials.ServiceCertificate.Authentication.CustomCertificateValidator = new ClientCertValidator();
    channelFactory.Credentials.ServiceCertificate.Authentication.RevocationMode = X509RevocationMode.NoCheck;

    /// Set appropriate client's certificate on the channel. Use CertManager class to obtain the certificate based on the "cltCertCN"
    channelFactory.Credentials.ClientCertificate.Certificate = CertManager.GetCertificateFromStorage(StoreName.My, StoreLocation.LocalMachine, cltCertCN);
    ips = channelFactory.CreateChannel();
}
```

ConfigurationManager

ConfigurationManager upravlja **ConfigFile** datotekom.ConfigurationManager implementira *Singleton pattern* koji osigurava integritet **ConfigFile** datoteke.



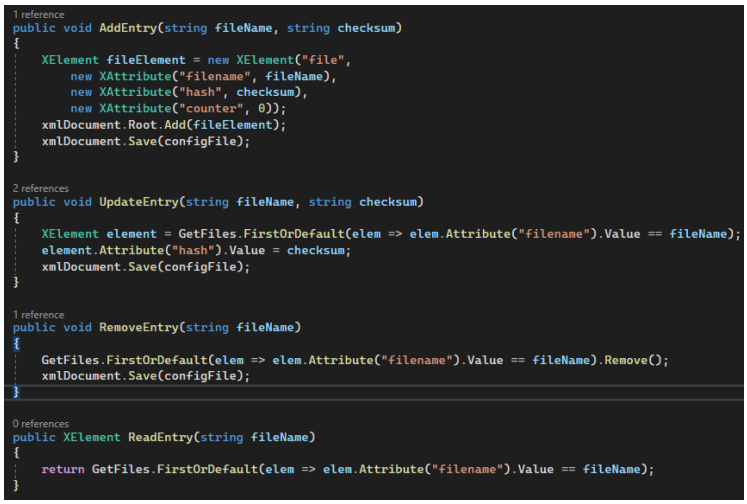
```
public static ConfigManager GetInstance()
{
    if (_instance == null)
    {
        _instance = new ConfigManager();
    }
    return _instance;
}

static ConfigManager _instance;
string folderPath = ConfigurationManager.AppSettings["MonitoredPath"];
string configFile = ConfigurationManager.AppSettings["ConfigFile"];
string signCertCN = ConfigurationManager.AppSettings["signCertCN"];

XDocument xmlDocument = new XDocument(new XElement("files"));

4 references
public IEnumerable<XElement> GetFiles { get => xmlDocument.Root.Elements("file"); }

1 reference
private ConfigManager()
{
    // If config.xml does not exist, generate a default one
    if (!File.Exists(configFile))
    {
        CreateConfig();
    }
    else
    {
        xmlDocument = XDocument.Load(configFile);
    }
}
```

CreateConfig

```
//metoda za kreiranje pocetnog config fajla od predefinisanog
//foldera sa fajlovima
1 reference
public void CreateConfig()
{
    //sve fajlove pakujemo u xml config fajl
    foreach (string filePath in Directory.GetFiles("MonitoredFiles"))
    {
        string filename = Path.GetFileName(filePath);

        string hash = string.Empty;
        byte[] data = File.ReadAllBytes(Path.Combine(folderPath, filename));


        X509Certificate2 certificateSign = CertManager.GetCertificateFromStorage(StoreName.My,
        StoreLocation.LocalMachine, signCertCN);
        hash = Convert.ToBase64String(DigitalSignature.Create(data, certificateSign));


        //za svaki fajl pamtimo ime, hash i broj
        //neovlascenih izmena
        XElement fileElement = new XElement("file",
            new XAttribute("filename", filename),
            new XAttribute("hash", hash),
            new XAttribute("counter", 0));
        xmlDocument.Root.Add(fileElement);
    }

    xmlDocument.Save(configFile);
}
```

Ukoliko navedeni **ConfigFile** ne postoji, generiše se konfiguraciona datoteka na osnovu sadržaja **MonitoredPath**.

Metode ConfigurationManager-a



```
1 reference
public void AddEntry(string fileName, string checksum)
{
    XElement fileElement = new XElement("file",
        new XAttribute("filename", fileName),
        new XAttribute("hash", checksum),
        new XAttribute("counter", 0));
    xmlDocument.Root.Add(fileElement);
    xmlDocument.Save(configFile);
}

2 references
public void UpdateEntry(string fileName, string checksum)
{
    XElement element = GetFiles.FirstOrDefault(elem => elem.Attribute("filename").Value == fileName);
    element.Attribute("hash").Value = checksum;
    xmlDocument.Save(configFile);
}

1 reference
public void RemoveEntry(string fileName)
{
    GetFiles.FirstOrDefault(elem => elem.Attribute("filename").Value == fileName).Remove();
    xmlDocument.Save(configFile);
}

0 references
public XElement ReadEntry(string fileName)
{
    return GetFiles.FirstOrDefault(elem => elem.Attribute("filename").Value == fileName);
}
```

Metode ConfigurationManager-a:

- AddEntry - dodavanje novog zapisa u konfiguracionu datoteku
- UpdateEntry - ažuriranje postojećeg zapisa konfiguracione datoteke
- RemoveEntry - brisanje zapisa iz konfiguracione datoteke
- ReadEntry - čitanje zapisa iz konfiguracione datoteke

FileIntegrityMonitoring

Klasa zadužena za samo nadgledanje datoteka i komunikaciju sa IPS servisom čije poruke su enkriptovane **TripleDES** algoritmom. Tajni ključ algoritma se iznova generiše pri pokretanju servisa.

StartMonitoring

```
1 reference
public void StartMonitoring()
{
    //ucitavanje configa, potpisivanje jednog po jednog fajla i
    //proveravanje njihovih hash vrednosti
    do
    {
        /// Exit logic
        if (Console.KeyAvailable)
        {
            if(Console.ReadKey(intercept:true).Key == ConsoleKey.Escape)
            {
                break;
            }
        }
        CustomConsole.WriteLine("Started scan...", MessageType.Info);
        foreach (XElement element in ConfigManager.GetInstance().GetFiles)
        {
            string filename = element.Attribute("filename").Value;

            byte[] data = File.ReadAllBytes(Path.Combine(folderPath, filename));
            if (!DigitalSignature.Verify(data, Convert.FromBase64String(element.Attribute("hash").Value), certificateSign))
            {
                int counter = int.Parse(element.Attribute("counter").Value);
                counter = counter + 1 > 3 ? 3 : counter + 1;

                CustomConsole.WriteLine("-----------------------------------------------------", MessageType.Warning);
                CustomConsole.WriteLine(filename + " - " + counter, MessageType.Warning);
                CustomConsole.WriteLine("-----------------------------------------------------", MessageType.Warning);

                element.Attribute("counter").Value = counter.ToString();

                string hash = Convert.ToBase64String(DigitalSignature.Create(data, certificateSign));

                element.Attribute("hash").Value = hash;


                ConfigManager.GetInstance().UpdateEntry(filename, hash);

                Intrusion intrusion = new Intrusion()
                {
                    TimeStamp = DateTime.Now,
                    FileName = filename,
                    Location = folderPath,
                    CompromiseLevel = (CompromiseLevel)counter,
                };

                ips.LogIntrusion(TripleDesAlgorithm.Encrypt(intrusion, key), key);
            }
        }
        CustomConsole.WriteLine("Scan finished...", MessageType.Info);

        Thread.Sleep(monitoringPeriod);
    } while (true);
}
```

StartMonitoring metoda ima za ulogu proveru integriteta datoteka iz **ConfigFile**-a kao i prijavu narušavanja integriteta IPS servisu.
Servis koristi “config.xml” datoteku u kojoj imamo zapis datoteka za nadgledanje kao i njihove digitalne potpise.

FileIntegrityMonitoringService

Servis implementira **IFileIntegrityService** interfejs koji je ujedno i **WCF Contract**.

```
namespace Common
{
    [ServiceContract]
    [ServiceKnownType(typeof(MonitoredFile))]
    8 references
    public interface IFileIntegrityService
    {
        [OperationContract]
        [FaultContract(typeof(CustomException))]
        2 references
        void AddFile(IFile file);

        [OperationContract]
        [FaultContract(typeof(CustomException))]
        2 references
        void UpdateFile(IFile file);

        [OperationContract]
        [FaultContract(typeof(CustomException))]
        2 references
        void RemoveFile(string fileName);

        [OperationContract]
        [FaultContract(typeof(CustomException))]
        2 references
        IFile ReadFile(string fileName);

        [OperationContract]
        [FaultContract(typeof(CustomException))]
        2 references
        List<string> ReadFileNames();
    }
}
```

AddFile metoda

```
[OperationBehavior(AutoDisposeParameters = true)]
2 references
public void AddFile(IFile file)
{
    if (File.Exists(Path.Combine(monitoredPath, file.Name)))
    {
        string message = $"File {file.Name} already exists";
        CustomConsole.WriteLine(message, MessageType.Error);
        throw new FaultException<CustomException>(new CustomException { FaultMessage = message},message);
    }
    else
    {
        using (FileStream fs = File.Create(Path.Combine(monitoredPath, file.Name)))
        {
            file.File.WriteTo(fs);
            X509Certificate2 certificateSign = CertManager.GetCertificateFromStorage(StoreName.My,
            StoreLocation.LocalMachine, signCertCN);

            if (certificateSign == null)
            {
                throw new FaultException<CustomException>(new CustomException{ FaultMessage = "No signature for signing was found!" });
            }

            string hash = Convert.ToBase64String(DigitalSignature.Create(file.File.ToArray(), certificateSign));
            // Console.WriteLine(hash);
            ConfigManager.GetInstance().AddEntry(file.Name, hash);
            CustomConsole.WriteLine($"File {file.Name} added", MessageType.Success);
        }
    }
}
```

AddFile metoda dodaje novu datoteku ukoliko ista već ne postoji. Također novododatu datoteku potpisuje digitalnim sertifikatom FIM servisa i potpis dodaje u konfiguraciju FIM servisa.

UpdateFile metoda

```
[OperationBehavior(AutoDisposeParameters = true)]
2 references
public void UpdateFile(IFile file)
{
    string path = Path.Combine(monitoredPath, file.Name);
    if (File.Exists(path))
    {
        File.WriteAllBytes(path,file.File.ToArray());
        X509Certificate2 certificateSign = CertManager.GetCertificateFromStorage(StoreName.My,
        StoreLocation.LocalMachine, signCertCN);

        if (certificateSign == null)
        {
            throw new FaultException<CustomException>(new CustomException { FaultMessage = "No signature for signing was found!" });
        }

        string hash = Convert.ToBase64String(DigitalSignature.Create(file.File.ToArray(), certificateSign));

        ConfigManager.GetInstance().UpdateEntry(file.Name, hash);
        CustomConsole.WriteLine($"File {file.Name} updated", MessageType.Success);

    }
    else
    {
        string message = $"File {file.Name} does not exist";
        CustomConsole.WriteLine(message, MessageType.Error);
        throw new FaultException<CustomException>(new CustomException { FaultMessage = message},message);
    }
}
```

UpdateFile metoda ažurira već postojeću datoteku, ponovo potpisuje datoteku i ažurira njen potpis u konfiguraciji.

RemoveFile metoda

```
2 references
public void RemoveFile(string fileName)
{
    string path = Path.Combine(monitoredPath, fileName);
    if (File.Exists(path))
    {
        ConfigManager.GetInstance().RemoveEntry(fileName);

        File.Delete(path);
        CustomConsole.WriteLine($"File {fileName} deleted", MessageType.Success);
    }
    else
    {
        string message = $"File {fileName} does not exist";
        CustomConsole.WriteLine(message, MessageType.Error);
        throw new FaultException<CustomException>(new CustomException { FaultMessage = message}, message);
    }
}
```

RemoveFile metoda briše datoteku ukoliko ona postoji i uklanja njen zapis iz konfiguracije.

ReadFile metoda

```
2 references
public IFile ReadFile(string fileName)
{
    string path = Path.Combine(monitoredPath, fileName);
    MonitoredFile mf = new MonitoredFile();

    if (File.Exists(path))
    {
        mf.Name = fileName;
        mf.Hash = String.Empty;

        using (FileStream fs = File.Open(path, FileMode.Open, FileAccess.Read))
        {
            fs.CopyTo(mf.File);
        }
        CustomConsole.WriteLine($"File {fileName} forwarded", MessageType.Info);
        return mf;
    }
    else
    {
        string message = $"File {fileName} does not exists";
        CustomConsole.WriteLine(message, MessageType.Error);
        throw new FaultException<CustomException>(new CustomException { FaultMessage = message},message);
    }
}
```

ReadFile metoda čita datoteku ukoliko ona postoji i prosleđuje istu FM servisu na zahtev klijenta.

ReadFileNames metoda

```
public List<string> ReadFileNames()
{
    List<string> fileNames = new List<string>();
    DirectoryInfo di = new DirectoryInfo(monitoredPath);

    foreach (FileInfo fi in di.GetFiles())
    {
        fileNames.Add(fi.Name);
    }

    return fileNames;
}
```

ReadFileNames metoda izlistava sve nazive datoteka na zahtev klijenta.

IPS servis

Konfiguracija servisa

```
<appSettings>
  <add key="serviceAddress" value="net.tcp://localhost:6002/IIntrusionPreventionSystem"/>
</appSettings>
```

- **serviceAddress** - adresa IPS servisa

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceDebug includeExceptionDetailInFaults="true" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <bindings>
    <netTcpBinding>
      <binding name="WinConfig">
        <security mode="Transport">
          <transport clientCredentialType="Windows" />
        </security>
      </binding>
    </netTcpBinding>
  </bindings>
  <client>
    <endpoint name="IFileManager"
          address="net.tcp://localhost:6001/IFileManager"
          binding="netTcpBinding"
          bindingConfiguration="WinConfig"
          contract="Common.IFileManager"
              />
  </client>
</system.serviceModel>
```

```
/// srvCertCN.SubjectName should be set to the service's username.  .NET WindowsIdentity class provides information about Windows user running the given process
string srvCertCN = Formatter.ParseName(WindowsIdentity.GetCurrent().Name);

NetTcpBinding binding = new NetTcpBinding();
binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.Certificate;

string address = ConfigurationManager.AppSettings["serviceAddress"];
ServiceHost host = new ServiceHost(typeof(IntrusionPreventionService));
host.AddServiceEndpoint(typeof(IIntrusionPreventionSystem), binding, address);
ServiceSecurityAuditBehavior auditBehavior = new ServiceSecurityAuditBehavior();

auditBehavior.AuditLogLocation = AuditLogLocation.Application;
auditBehavior.ServiceAuthorizationAuditLevel = AuditLevel.SuccessOrFailure;

host.Description.Behaviors.Remove<ServiceSecurityAuditBehavior>();
host.Description.Behaviors.Add(auditBehavior);



///Custom validation mode enables creation of a custom validator - CustomCertificateValidator
host.Credentials.ClientCertificate.Authentication.CertificateValidationMode = X509CertificateValidationMode.Custom;
host.Credentials.ClientCertificate.Authentication.CustomCertificateValidator = new ServiceCertValidator();

///If CA doesn't have a CRL associated, WCF blocks every client because it cannot be validated
host.Credentials.ClientCertificate.Authentication.RevocationMode = X509RevocationMode.NoCheck;

///Set appropriate service's certificate on the host. Use CertManager class to obtain the certificate based on the "srvCertCN"
host.Credentials.ServiceCertificate.Certificate = CertManager.GetCertificateFromStorage(StoreName.My, StoreLocation.LocalMachine, srvCertCN);

try
{
    host.Open();

    CustomConsole.WriteLine("Intrusion prevention service started. Press Esc to exit...", MessageType.Info);
    while (Console.ReadKey(intercept: true).Key != ConsoleKey.Escape) ;
}
catch (Exception e)
{
    CustomConsole.WriteLine(e.Message, MessageType.Error);
}
finally
{
    host.Close();
}
```

IntrusionPreventionService

IPS servis implementira **IIntrusionPreventionSystem** interface koji je ujedno i **WCF contract**.

```
[ServiceContract]
5 references
public interface IIntrusionPreventionSystem
{
    [OperationContract]
    [FaultContract(typeof(CustomException))]
    2 references
    void LogIntrusion(string data, string secret_key);
}
```

LogIntrusion metoda

```
2 references
public void LogIntrusion(string data, string secret_key)
{
    try
    {
        Intrusion intrusion = TripleDesAlgorithm.Decrypt(data, secret_key);
        CustomConsole.WriteLine($"'{intrusion.TimeStamp}' - Intrusion level '{intrusion.CompromiseLevel}' logged for file '{intrusion.FileName}'
        Audit.LogIntrusion(intrusion);
        if (intrusion.CompromiseLevel == CompromiseLevel.Critical)
        {
            CustomConsole.WriteLine("Requesting file removal...", MessageType.Info);
            if (fileManager.RequestRemoval(intrusion.FileName))
            {
                CustomConsole.WriteLine("File removed successfully", MessageType.Success);
            }
            else
            {
                CustomConsole.WriteLine("File removal failed!", MessageType.Error);
            }
        }
    }
    catch (FaultException<CustomException> fe)
    {
        CustomConsole.WriteLine(fe.Detail.Message, MessageType.Error);
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
}
```

LogIntrusion metoda dekriptuje **Intrusion** podatak dobijen od FIM servisa i upisuje u **Audit**. Ukoliko je dobijeni **Intrusion** nivoa **Critical** zahteva od FM servisa brisanje datoteke.

Nivoi **Intrusion**-a:

```
[DataContract]
3 references
public enum CompromiseLevel
{
    [EnumMember]
    Info = 1,
    [EnumMember]
    Warning = 2,
    [EnumMember]
    Critical = 3
};
```

Audit

Klasa zadužena za upisivanje **Intrusion**-a u windows log.

```
private static EventLog customLog = null;
const string SourceName = "IntrusionPreventionSystem.Audit";
const string LogName = "IntrusionLog";

0 references
static Audit()
{
    try
    {
        if (!EventLog.SourceExists(SourceName))
        {
            EventLog.CreateEventSource(SourceName, LogName);
        }
        customLog = new EventLog(LogName,
            Environment.MachineName, SourceName);
    }
    catch (Exception e)
    {
        customLog = null;
        Console.WriteLine("Error while trying to create log handle. Error = {0}", e.Message);
    }
}
```

LogIntrusion metoda

LogIntrusion metoda upisuje prosleđeni **Intrusion** u windows log.

```
1 reference
public static void LogIntrusion(Intrusion intrusion)
{
    if (customLog != null)
    {
        string message = $"[{intrusion.TimeStamp}] [{intrusion.CompromiseLevel.ToString()}] - Intrusion logged for file '{intrusion.FileName}' at '{intrusion.Location}'";
        customLog.WriteEntry(message);
    }
    else
    {
        throw new ArgumentException(string.Format("Error while trying to write event to event log."));
    }
}
```

FM servis

Konfiguracija servisa

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceDebug includeExceptionDetailInFaults="true" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <services>
    <service name="FileManagerProject.FileManagerService">
      <host>
        <baseAddresses>
          <add baseAddress="net.tcp://localhost:6001"/>
        </baseAddresses>
      </host>
      <endpoint address="IFileManager"
                        binding="netTcpBinding"
                        bindingConfiguration="WinConfig"
                        contract="Common.IFileManager"
                        />
      <endpoint address="IClient"
                        binding="netTcpBinding"
                        bindingConfiguration="WinConfig"
                        contract="Common.IClient"
                        />
    </service>
  </services>
  <bindings>
    <netTcpBinding>
      <binding name="WinConfig">
        <security mode="Transport">
          <transport clientCredentialType="Windows" />
        </security>
      </binding>
    </netTcpBinding>
  </bindings>
  <client>
    <endpoint name="IFileMonitoring"
              address="net.tcp://localhost:6000/IFileIntegrityService"
              binding="netTcpBinding"
              bindingConfiguration="WinConfig"
              contract="Common.IFileIntegrityService"
              />
  </client>
</system.serviceModel>
```

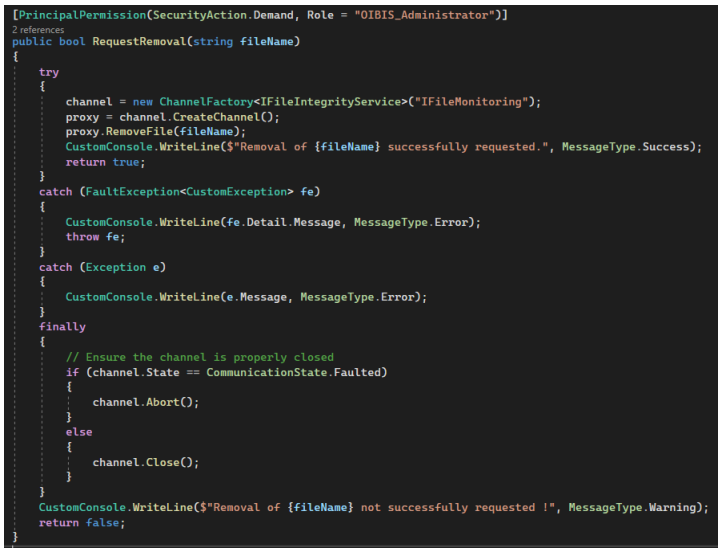
FileManagerService

FM servis implementira **IFileManager** i **IClient** interfejse koji su ujedno i **WCF contracts**.

```
[ServiceContract]
4 references
public interface IFileManager
{
    [OperationContract]
    [FaultContract(typeof(CustomException))]
    2 references
    bool RequestRemoval(string fileName);
}
```

```
[ServiceContract]
[ServiceKnownType(typeof(MonitoredFile))]
6 references
public interface IClient
{
    [OperationContract]
    [FaultContract(typeof(CustomException))]
    2 references
    void AddFile(IFile file);

    [OperationContract]
    [FaultContract(typeof(CustomException))]
    2 references
    IFile ReadFile(string fileName);

    [OperationContract]
    [FaultContract(typeof(CustomException))]
    2 references
    void UpdateFile(IFile file);

    [OperationContract]
    [FaultContract(typeof(CustomException))]
    2 references
    List<string> ReadFiles();
}
```

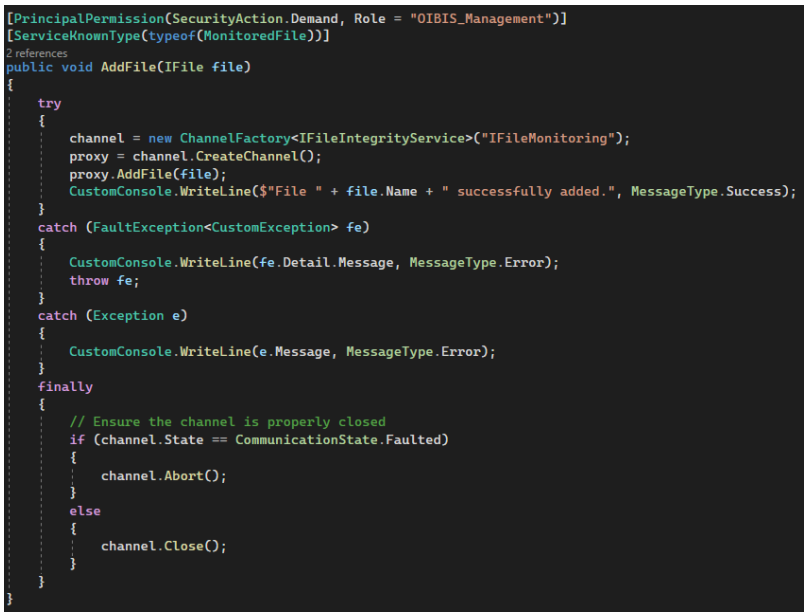
RequestRemoval metoda



```
[PrincipalPermission(SecurityAction.Demand, Role = "OIBIS_Administrator")]
2 references
public bool RequestRemoval(string fileName)
{
    try
    {
        channel = new ChannelFactory<IFileIntegrityService>("IFileMonitoring");
        proxy = channel.CreateChannel();
        proxy.RemoveFile(fileName);
        CustomConsole.WriteLine($"Removal of {fileName} successfully requested.", MessageType.Success);
        return true;
    }
    catch (FaultException<CustomException> fe)
    {
        CustomConsole.WriteLine(fe.Detail.Message, MessageType.Error);
        throw fe;
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
    finally
    {
        // Ensure the channel is properly closed
        if (channel.State == CommunicationState.Faulted)
        {
            channel.Abort();
        }
        else
        {
            channel.Close();
        }
    }
    CustomConsole.WriteLine($"Removal of {fileName} not successfully requested !", MessageType.Warning);
    return false;
}
```

RequestRemoval metoda prosleđuje zahtev za brisanje datoteke IPS servisa FIM servisu ukoliko je autorizacija uspešna.

AddFile metoda



```
[PrincipalPermission(SecurityAction.Demand, Role = "OIBIS_Management")]
[ServiceKnownType(typeof(MonitoredFile))]
2 references
public void AddFile(IFile file)
{
    try
    {
        channel = new ChannelFactory<IFileIntegrityService>("IFileMonitoring");
        proxy = channel.CreateChannel();
        proxy.AddFile(file);
        CustomConsole.WriteLine($"File " + file.Name + " successfully added.", MessageType.Success);
    }
    catch (FaultException<CustomException> fe)
    {
        CustomConsole.WriteLine(fe.Detail.Message, MessageType.Error);
        throw fe;
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
    finally
    {
        // Ensure the channel is properly closed
        if (channel.State == CommunicationState.Faulted)
        {
            channel.Abort();
        }
        else
        {
            channel.Close();
        }
    }
}
```

AddFile metoda prosleđuje dobijenu datoteku od klijenta FIM servisu ukoliko je autorizacija uspešna.

ReadFile metoda

```
[PrincipalPermission(SecurityAction.Demand, Role = "OIBIS_Management")]
2 references
public IFile ReadFile(string fileName)
{
    try
    {
        channel = new ChannelFactory<IFileIntegrityService>("IFileMonitoring");
        proxy = channel.CreateChannel();
        var x = proxy.ReadFile(fileName);
        CustomConsole.WriteLine($"File {fileName} was read successfully.", MessageType.Success);
        return x;
    }
    catch (FaultException<CustomException> fe)
    {
        CustomConsole.WriteLine(fe.Detail.Message, MessageType.Error);
        throw fe;
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
    finally
    {
        // Ensure the channel is properly closed
        if (channel.State == CommunicationState.Faulted)
        {
            channel.Abort();
        }
        else
        {
            channel.Close();
        }
    }
    return new MonitoredFile();
}
```

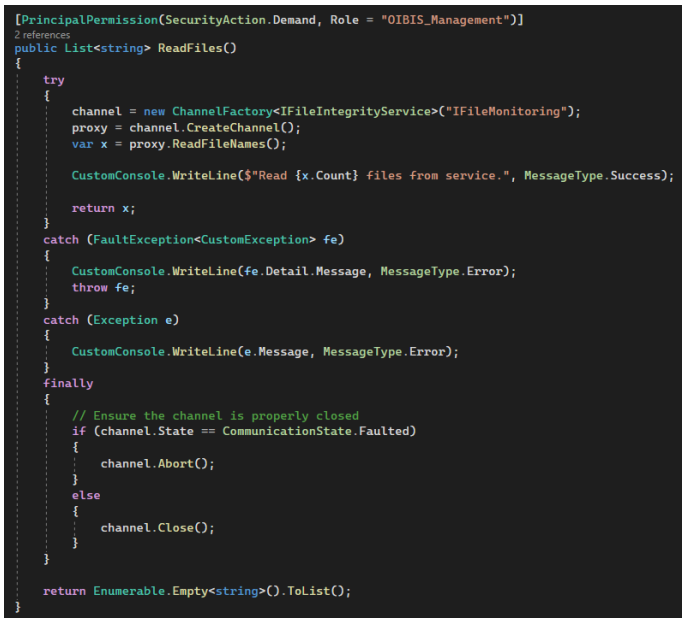
ReadFile metoda prosleđuje pročitanu datoteku sa FIM servisa klijentu ukoliko je autorizacija uspešna.

UpdateFile metoda

```
[PrincipalPermission(SecurityAction.Demand, Role = "OIBIS_Management")]
[OperationBehavior(AutoDisposeParameters = true)]
2 references
public void UpdateFile(IFile file)
{
    try
    {
        channel = new ChannelFactory<IFileIntegrityService>("IFileMonitoring");
        proxy = channel.CreateChannel();
        proxy.UpdateFile(file);
        CustomConsole.WriteLine($"File {file.Name} successfully updated.", MessageType.Success);
    }
    catch (FaultException<CustomException> fe)
    {
        CustomConsole.WriteLine(fe.Detail.Message, MessageType.Error);
        throw fe;
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
    finally
    {
        // Ensure the channel is properly closed
        if (channel.State == CommunicationState.Faulted)
        {
            channel.Abort();
        }
        else
        {
            channel.Close();
        }
    }
}
```

UpdateFile metoda prosleđuje ažuriranu datoteku dobijenu od klijenta FIM servisu ukoliko je autorizacija uspešna.

ReadFiles metoda



```
[PrincipalPermission(SecurityAction.Demand, Role = "OIBIS_Management")]
2 references
public List<string> ReadFiles()
{
    try
    {
        channel = new ChannelFactory<IFileIntegrityService>("IFileMonitoring");
        proxy = channel.CreateChannel();
        var x = proxy.ReadFileNames();

        CustomConsole.WriteLine($"Read {x.Count} files from service.", MessageType.Success);

        return x;
    }
    catch (FaultException<CustomException> fe)
    {
        CustomConsole.WriteLine(fe.Detail.Message, MessageType.Error);
        throw fe;
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
    finally
    {
        // Ensure the channel is properly closed
        if (channel.State == CommunicationState.Faulted)
        {
            channel.Abort();
        }
        else
        {
            channel.Close();
        }
    }

    return Enumerable.Empty<string>().ToList();
}
```

ReadFiles metoda čita imena svih datoteka sa FIM servisa i prosleđuje imena klijentu ukoliko je autorizacija uspešna.

Client

Primer klijentske aplikacije koja komunicira sa sistemom za nadgledanje datoteka.

```
static void Main(string[] args)
{
    try
    {
        /// Create a client proxy
        cf = new ChannelFactory<IClient>("Client");
        proxy = cf.CreateChannel();
        /// User menu
        string key = string.Empty;
        do
        {
            try
            {
                // Print username of a user who is running a service
                Formatter.PrintCurrentUser();
                Console.Write("---------------------\nOptions:\n\tA - Add file\n\tU - update file\n\tQ - Quit process\n\nPick: ");
                key = Console.ReadLine().ToUpper();
                switch (key)
                {
                    case "A": AddFile(); break;
                    case "U": UpdateFile(); break;
                    default: break;
                }
            }
            catch (FaultException<CustomException> fe)
            {
                CustomConsole.WriteLine(fe.Detail.FaultMessage, MessageType.Error);
                if (cf.State == CommunicationState.Faulted)
                {
                    cf = new ChannelFactory<IClient>("Client");
                    proxy = cf.CreateChannel();
                }
            }
            catch (Exception e)
            {
                CustomConsole.WriteLine(e.Message, MessageType.Error);
                if (cf.State == CommunicationState.Faulted)
                {
                    cf = new ChannelFactory<IClient>("Client");
                    proxy = cf.CreateChannel();
                }
            }
        } while (key != "Q");
    }
    catch (Exception e)
    {
        CustomConsole.WriteLine(e.Message, MessageType.Error);
    }
    finally
    {
        /// Close channel on exit
        if (cf != null)
        {
            if (cf.State == CommunicationState.Faulted)
            {
                cf.Abort();
            }
            else
            {
                cf.Close();
            }
        }
    }

}
```

Konfiguracija klijenta

```
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <netTcpBinding>
        <binding name="WinConfig">
          <security mode="Transport">
            <transport clientCredentialType="Windows" />
          </security>
        </binding>
      </netTcpBinding>
    </bindings>
    <client>
      <endpoint name="Client"
            address="net.tcp://localhost:6001/IClient"
            binding="netTcpBinding"
            bindingConfiguration="WinConfig"
            contract="Common.IClient"
                />
    </client>
  </system.serviceModel>
</configuration>
```

AddFile metoda

```
1 reference
static void AddFile()
{

    Console.Write("Enter file name: ");
    string filename = Console.ReadLine();
    if (filename == "exit")
    {
        return;
    }
    /// Open empty file in console editor
    ConsoleFileEditor editor = new ConsoleFileEditor();
    editor.Edit();

    /// Save data from console editor to file
    IFile file = editor.SaveToFile(filename);
    Console.WriteLine($"{filename} created successfully...");

    /// forward file to service
    proxy.AddFile(file);
}
```

UpdateFile metoda

```
1 reference
static void UpdateFile()
{

    /// Read available files on service
    Console.WriteLine("--------------- Available files ---------------");
    List<string> fileNames = proxy.ReadFiles();
    foreach (var x in fileNames)
    {
        Console.WriteLine($"- {x}");
    }
    string filename = string.Empty;

    Regex r = new Regex(@".+\.[a-zA-Z]+$");
    /// Select file for editing
    do
    {
        Console.Write("\nSelect file: ");
        filename = Console.ReadLine();
        if (filename == "exit")
        {
            return;
        }
        /// Add default extenstion if none is specified
        if (!r.IsMatch(filename))
        {
            filename += ".txt";
        }
    } while (!fileNames.Contains(filename));

    /// Get file from service
    IFile f = proxy.ReadFile(filename);

    /// Read data from file in console editor
    byte[] data = f.File.ToArray();
    ConsoleFileEditor editor = new ConsoleFileEditor(Encoding.UTF8.GetString(data).Split('\n'));
    editor.Edit();

    /// Save updated file
    f = editor.SaveToFile(f.Name);

    /// Forward updated file to service
    proxy.UpdateFile(f);
}
```

UpdateFile metoda za uređivanje datoteke koristi već implementiran **ConsoleEditor** koji očekuje listu stringova.

ConsoleEditor pruža:

- **ConsoleEditor(List<string> lines)** - omogućava uređivanje linija direktno u konzoli
- **ConsoleEditor.Edit()** - startuje uređivanje u konzoli. Automatsko čuvanje pristikom na <Esc>
- **ConsoleEditor.SaveToFile(string fileName)** - Čuvanje ažuriranih linija u **IFile** objekat

Common - pomoćne klase

CustomConsolePrint

```
public enum MessageType
{
    Info,
    Error,
    Warning,
    Success
}

46 references
public static class CustomConsole
{
    46 references
    public static void WriteLine(string message, MessageType messageType)
    {
        ConsoleColor originalColor = Console.ForegroundColor;

        switch (messageType)
        {
            case MessageType.Info:
                Console.ForegroundColor = ConsoleColor.Gray;
                break;
            case MessageType.Warning:
                Console.ForegroundColor = ConsoleColor.Yellow;
                break;
            case MessageType.Error:
                Console.ForegroundColor = ConsoleColor.Red;
                break;
            case MessageType.Success:
                Console.ForegroundColor = ConsoleColor.Green;
                break;
            default:
                Console.ForegroundColor = ConsoleColor.Gray;
                break;
        }

        Console.WriteLine($"{DateTime.Now} - [{messageType}] {message}");

        // Reset the console color to the original color
        Console.ForegroundColor = originalColor;
    }
}
```

Pomoćna klasa za uređeni ispis poruka na osnovu prosleđenog tipa.

CustomException

```
[DataContract]
30 references
public class CustomException
{
    [DataMember]
    13 references
    public string FaultMessage { get; set; }
}
```

Pomoćna klasa za prilagođene poruke očekivane greške.

Intrusion

```
[DataContract]
7 references
public class Intrusion
{
    [DataMember]
    3 references
    public DateTime TimeStamp { get; set; }
    [DataMember]
    4 references
    public string FileName { get; set; }
    [DataMember]
    3 references
    public string Location { get; set; }
    [DataMember]
    4 references
    public CompromiseLevel CompromiseLevel { get; set; }
}
```

Pomoćna klasa za smeštanje podataka o narušavanju integriteta nadgledanih datoteka.

```
[DataContract]
3 references
public enum CompromiseLevel
{
    [EnumMember]
    Info = 1,
    [EnumMember]
    Warning = 2,
    [EnumMember]
    Critical = 3
};
```

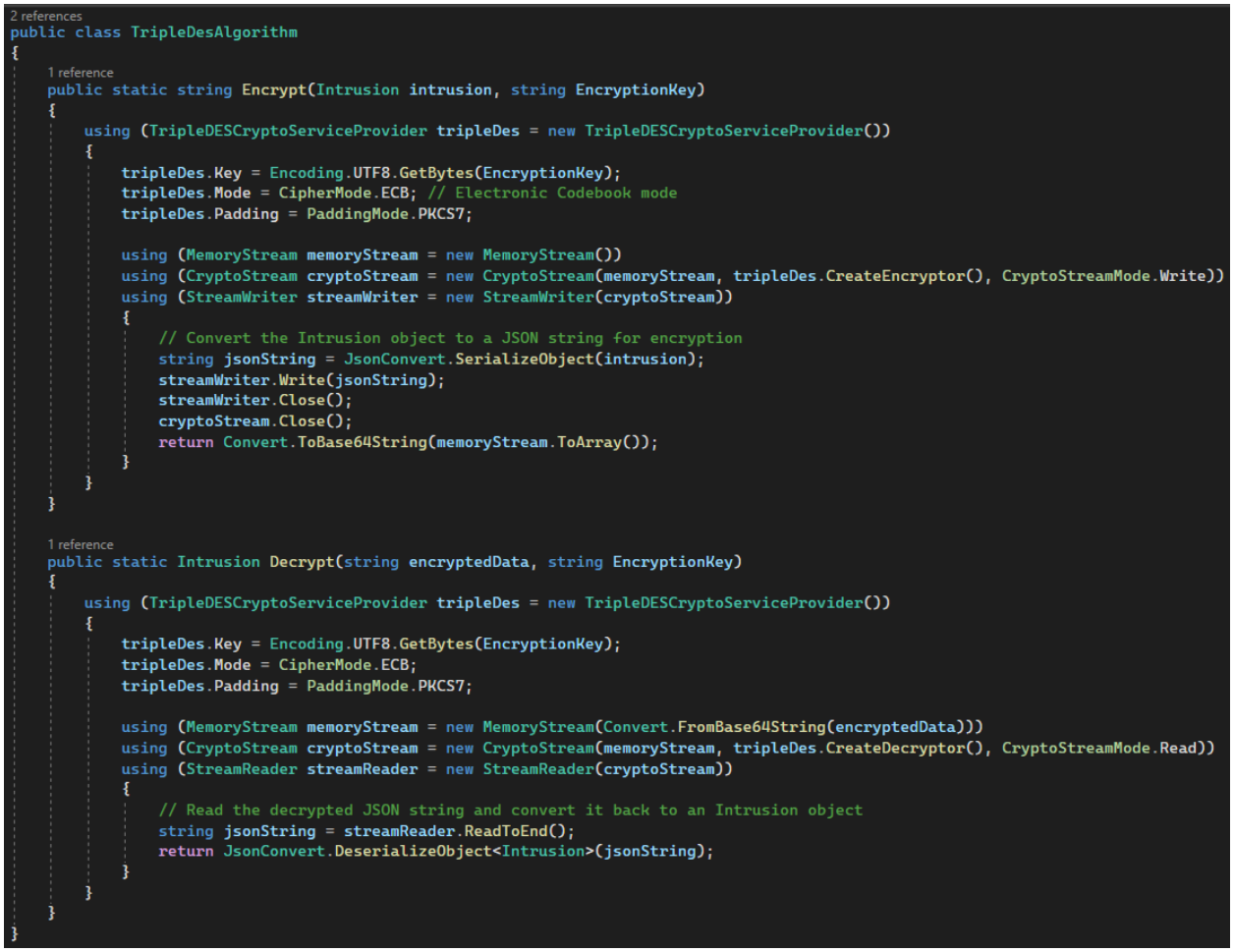
IFile - MonitoredFile

```
public interface IFile: IDisposable
{
    8 references
    MemoryStream File { get; set; }
    15 references
    string Name { get; set; }
    3 references
    string Hash { get; set; }
}
```

```
[DataContract]
9 references
public class MonitoredFile : IFile
{
    private bool disposedValue;
    private MemoryStream file;
    private string name;
    private string hash;

    [DataMember]
    8 references
    public MemoryStream File { get => file; set => file = value ; }
    [DataMember]
    15 references
    public string Name { get => name ; set => name= value; }
    [DataMember]
    3 references
    public string Hash { get => hash; set => hash = value; }


    3 references
    public MonitoredFile()
    {
        file = new MemoryStream();
    }
```

Pomoćna klasa za čuvanje informacija o nadgledanih datotekama koje kruže kroz sistem.

TripleDesAlgorithm



```
2 references
public class TripleDesAlgorithm
{
    1 reference
    public static string Encrypt(Intrusion intrusion, string EncryptionKey)
    {
        using (TripleDESCryptoServiceProvider tripleDes = new TripleDESCryptoServiceProvider())
        {
            tripleDes.Key = Encoding.UTF8.GetBytes(EncryptionKey);
            tripleDes.Mode = CipherMode.ECB; // Electronic Codebook mode
            tripleDes.Padding = PaddingMode.PKCS7;

            using (MemoryStream memoryStream = new MemoryStream())
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, tripleDes.CreateEncryptor(), CryptoStreamMode.Write))
            using (StreamWriter streamWriter = new StreamWriter(cryptoStream))
            {
                // Convert the Intrusion object to a JSON string for encryption
                string jsonString = JsonConvert.SerializeObject(intrusion);
                streamWriter.Write(jsonString);
                streamWriter.Close();
                cryptoStream.Close();
                return Convert.ToBase64String(memoryStream.ToArray());
            }
        }
    }

    1 reference
    public static Intrusion Decrypt(string encryptedData, string EncryptionKey)
    {
        using (TripleDESCryptoServiceProvider tripleDes = new TripleDESCryptoServiceProvider())
        {
            tripleDes.Key = Encoding.UTF8.GetBytes(EncryptionKey);
            tripleDes.Mode = CipherMode.ECB;
            tripleDes.Padding = PaddingMode.PKCS7;

            using (MemoryStream memoryStream = new MemoryStream(Convert.FromBase64String(encryptedData)))
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, tripleDes.CreateDecryptor(), CryptoStreamMode.Read))
            using (StreamReader streamReader = new StreamReader(cryptoStream))
            {
                // Read the decrypted JSON string and convert it back to an Intrusion object
                string jsonString = streamReader.ReadToEnd();
                return JsonConvert.DeserializeObject<Intrusion>(jsonString);
            }
        }
    }
}
```

Pomoćna klasa koja omogućava funkcionalnost enkriptovanja i dekriptovanja **Intrusion** objekta za potrebe komunikacije između **FIM** i **IPS** servisa.