KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Darius Lukšas

Programinio kodo obfuskacija ir įgyvendinimo metodai

Magistro darbas

Vadovas
doc. dr. Eimutis Karčiauskas

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. Eduardas Bareiša

Programinio kodo obfuskacija ir įgyvendinimo metodai

Informatikos magistro baigiamasis darbas

Recenzentas
doc. dr. Aleksas Riškus

Vadovas
doc. dr. Eimutis Karčiauskas

Atliko
IFN 3 gr. stud.
Darius Lukšas

Kaunas, 2006

TURINYS

LENTELĖS

PAVEIKSLAI

1. ĮVADAS

Tyrimo sritis - Microsoft .NET Framework technologijomis paremtų failų obfuskacija.

Temos aktualumas – plintanti Microsoft .NET aplinkos technologija yra nesaugi programos kodo atžvilgiu. Programos kodas, resursai, algoritmai gali būti lengvai dekompiliuojami ir matomi bet kuria .NET paremta aukšto lygio programavimo kalba. Toks kodas gali būti lengvai kopijuojamas savavališkam naudojimui.

Darbo tikslas – aprašyti ir išanalizuoti esamus .NET aplinkos failų obfuskacijos metodus, sukurti sistemą (obfuskatorių) demonstruojančią vieną iš metodų.

Darbo uždaviniai:

1. aprašyti ir palyginti esamus sistemos kūrimo analizės metodus;
2. išanalizuoti tyrimo objektą;
3. aprašyti ir palyginti literatūros šaltiniuose pateiktus problemos sprendimus;
4. apibrėžti kuriamos sistemos kokybės kriterijus;
5. pateikti sistemos projektą;
6. suformuluoti sistemos projekto išvadas.

Tyrimo objektas - Microsoft .NET Framework CLR (Common Language Runtime) failai ir jų obfuskavimo metodai.

Tyrimo metodika:

1. Literatūros analizė ir apibendrinimas.
2. Duomenų sisteminimas, palyginimas ir analizė.

2. ANALIZĖS DALIS

2.1 Analizės metodų ir priemonių pasirinkimas

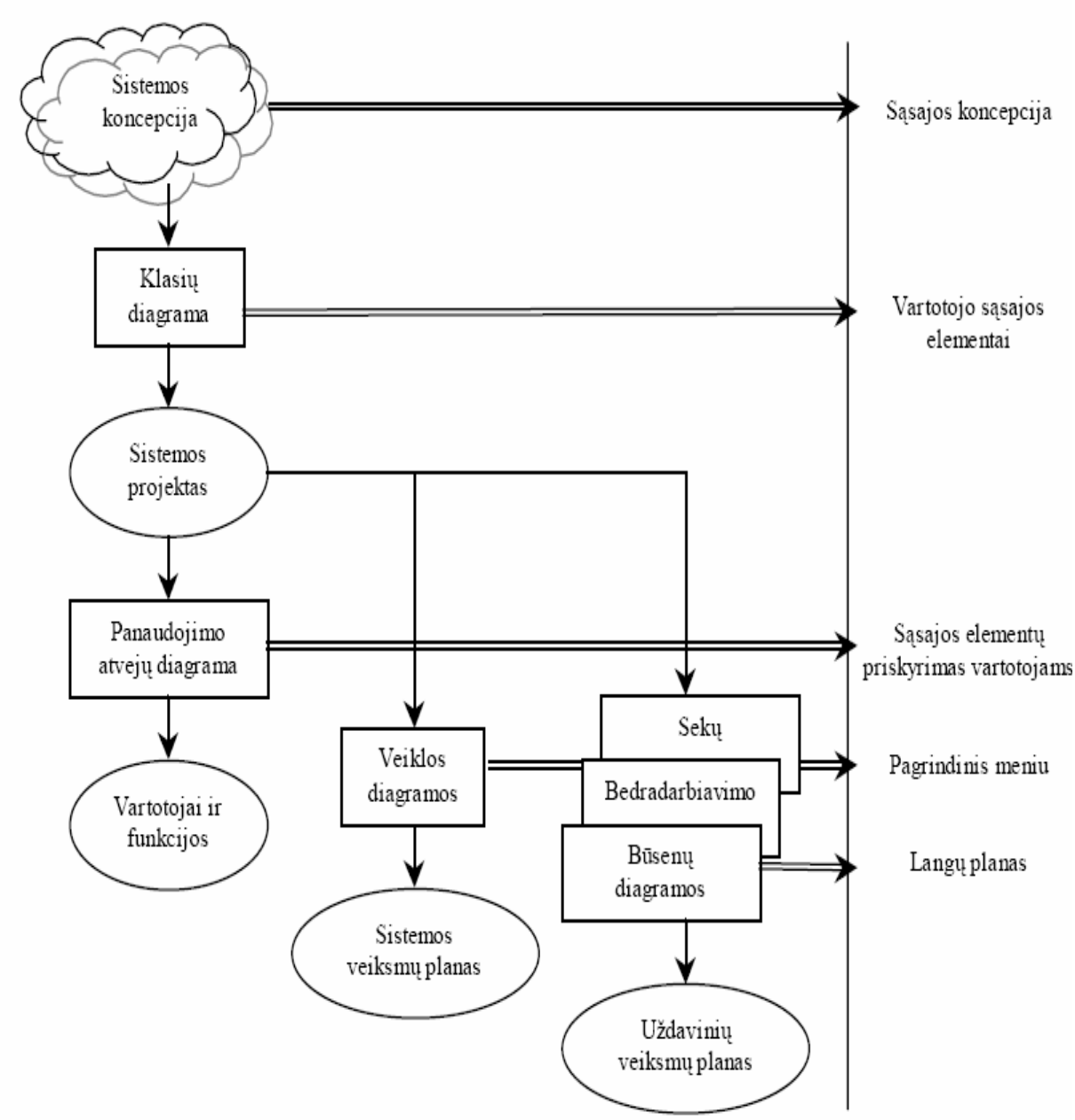
Projektuojant informacijos sistemas vienas iš pagrindinių klausimų, lemiančių projekto kokybę ir jo įdiegimo tempus, yra vartotojo sąsaja. Panagrinėsime taikomųjų programų projektavimo metodikas:

- unifikuota modeliavimo kalba (UML);
- struktūrinis projektavimas.

2.1.1 UML (Unified Modeling Process)

UML - tai formalių uždavinio procesus aprašančių diagramų rinkinys, kuriame galima įžvelgti nuoseklų ėjimą nuo uždavinio koncepcijos iki duomenų bazės schemos ir uždavinio sprendimo algoritmo formalaus aprašymo. 1 paveiksle parodytas galimas šių diagramų apjungimas projektavimo procese ir išskirti sąsajos projektavimo žingsniai, kurie turėtų būti atlikti bendroje sistemos projektavimo veiksmų sekoje.

Klasių diagrama yra pradinė ir pagrindinė, nes jai sudaryti reikia žinoti uždavinio formuluotę ir esminius struktūrinius elementus. Ši diagrama leidžia aiškiai suprasti, ko yra siekiama uždaviniu ir konceptualiai parodyti ryšiai tarp uždavinio struktūrinių elementų (klasių). Klasių diagramą galima laikyti apibendrintu sistemos projektu, todėl ją kuriant paaiškėja atskiri vartotojo sąsajos elementai. Sudarius klasių diagramą, galima vystyti sistemos aprašymą panaudojimo atvejų (angl. *USE CASE*) diagramą, kurioje sistema yra paskirstoma “menamiems” vartotojams, įsivaizduojant jų funkcijas sistemoje. Taigi, šitokia diagrama leidžia įsivaizduoti atskirų vartotojų sąsajas, bet tik apibendrintame lygyje. Sistemos projektas, aprašytas klasių diagramose leidžia detalizuoti sistemą sprendžiamais uždaviniais (veiklos diagrama) ir atlikti šių uždavinių detalizaciją (būsenų, sekų, bendradarbiavimo diagramos). Šios keturios diagramos vadinamos elgsenos diagramomis. Pirmoji leidžia sudaryti pagrindinį sąsajos meniu, o kitos trys – vartotojo sąsajos langų planą.



1 pav. UML ir vartotojo sąsajos projektavimas

2.1.2 Struktūrinis projektavimas

Tai gebėjimas pasirinkti sistemos komponentus ir sąryšius tarp jų. Pagrindinis projektavimo tikslas perkelti sistemos funkcinius reikalavimus į programos bei technikos reikalavimus. Projektavimo rezultatai: programos specifikacija, testavimo planai, projektiniai sprendimai, apmokymo ir instaliavimo planai. Taip pat projektavimas gali įtakoti kitas programos kūrimo proceso dalis.

Struktūrinio projektavimo tiksliai yra paprasti ir aiškūs: sumažinti kūrimo bei prižiūrėjimo laiką ir kainą. Šiuos struktūrinio projektavimo tikslus pirmieji paminėjo Yourdon ir Constantine (1979). Tikslus galima pasiekti sistemoje naudojant kiek galima mažesnius, bet nepriklausomus komponentus. Projektavime ir turi būti nustatyti šie komponentai.

Struktūrinės analizės bei projektavimo tikslai aiškūs, bet nėra griežtai nurodoma, kaip juos pasiekti. Naudojantis šia metodologija norima pasiekti minimalaus programos dalių sukibimo bei maksimalaus rišlumo, tačiau praktiškai šie pasiekimai remiasi pačio žmogaus patirtimi, o ne naudojant formalius metodus. Jeigu sistemoje yra didelis komponentų sukibimas ir mažas rišlumas tai apsunkina sistemos priežiūrą bei testavimą.

2.1.3 Metodų analizės išvados

Lyginant su UML, struktūrinė metodologija nėra pati geriausia, bet ji gali būti sėkmingai naudojama kuriant programas, kurios iš prigimties yra procedūrinės. Nesant formaliai aprašytiems vartotojo sąsajos reikalavimams sistemų projektavimo technologijose, sistemos kokybė nukenčia dėl šių procesų integracijos nebuvimo. Atlikus sistemos projektavimo ir vartotojo sąsajos projektavimo technologijų analizę, išryškėjo galimybės dalinai integruoti šias technologijas. Baigiamieji vartotojo sąsajos projektavimo žingsniai yra subjektyvūs, todėl integruoti jų į sistemos projektavimo technologiją negalima. UML projektavimo technologija, atspindi objektišką programavimo stilių. Struktūrinis projektavimas žiūri į sistemą, kaip į vientisą funkciją, palaipsniui sudalytų į smulkesnes įvairaus lygio funkcijas.

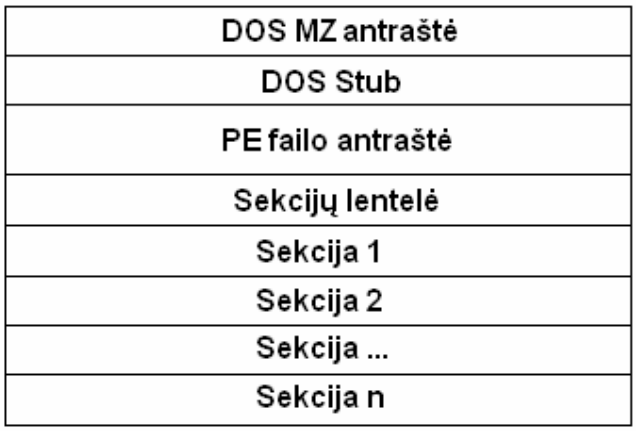
Tolimesnei veiklos analizei ir reikalavimų modeliui buvo pasirinktas struktūrinis projektavimas, nes jis labiausiai atitiko kuriamos sistemos poreikius.

2.2 Veiklos analizė

2.2.1 Objekto charakteristika

Prieš nagrinėjant objektą reikėtų išsiaiškinti jo sandarą ir kai kurias sąvokas.

PE (Portable Executable) – tai failai (exe, dll, ocx, sys ir t.t.) veikiantys Windows šeimos operacinėse sistemose, šis formatas buvo suprojektuotas Microsoft 1993 metais ir standartizuotas Tool Interface Standard Committee (Microsoft, Intel, Borland, Watcom, IBM ir kiti) – daugumoje paremtas Common Object File Format (COFF), kuris naudojamas kai kuriuose UNIX ir VMS operacinėse sistemose. 32 bitų PE failo sandara galime matyti 2 pav.



2 pav. PE failo sandara

DOS MZ antraštė (standartnis dydis 64 baitai) – patys pirmi 2 baitai pagal kuriuos nustatoma ar failas yra MS-DOS paleidžiamasis turi būti raidės „MZ“. Paskutiniai 4 baitai nurodo PE failo antraštės adresą. Kita informacija esanti šioje antraštėje nebus naudojama kuriant sistemą.

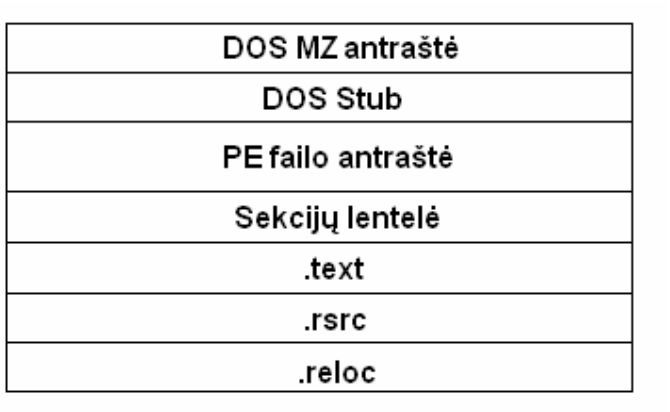
DOS stub (standartnis dydis 64 baitai) – asemblerio instrukcijos užima pirmus 14 baitų, jos išveda toliau sekantį klaidos pranešimą paleidus failą DOS aplinkoje. Jokia informacija esanti šioje dalyje kuriant sistemą nebus naudojama.

PE failo antraštė (PE Header)- pirmi du baitai turi būti raidės „PE“, pagal tai nustatoma ar tai PE failas, visa kita informacija kuriant sistemą nebus naudojama.

Sekcijų lentelė (dydis priklauso nuo sekcijų skaičiaus) – po sekcijos vardo pabaigos sekančių septynių baitų esantys keturi baitai nurodo sekcijos virtualų adresą, toje vietoje prasideda įvardinta sekcija, dar sekantys keturi baitai nurodo sekcijos pradžios realų adresą. Kuriant sistemą ieškosime .text sekcijos virtualaus adreso, visa kita informacija kuriant sistemą nereikalinga.

Sekcijos (dydis priklauso nuo importų, kodo, resursų ir t.t. dydžio) – sekcijų vardai realiai neturi reikšmės (kiekvienas kompiliatorius sukuria savo), dažniausiai sekcijose .text, .code būna sukompiliuotas programos kodas, .rsrc – resursai, .idata, .data – importų lentelės. Kuriant sistemą bus nagrinėjama tik .text sekcija.

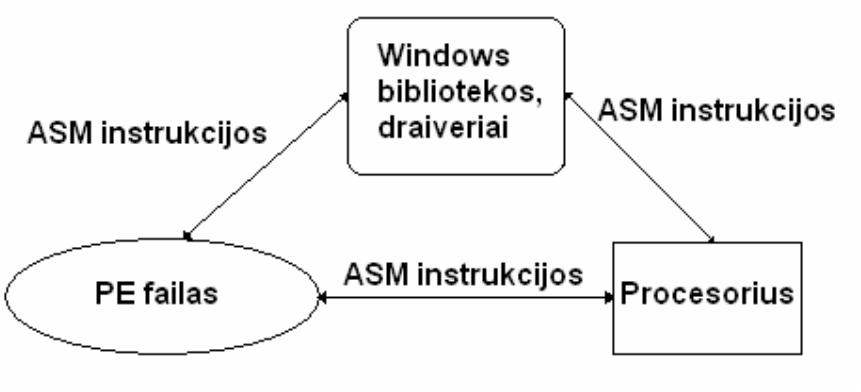
CLR (Common Language Runtime) – tai ganėtinai naujas Microsoft Framework .NET bibliotekomis paremtas PE failo formatas. Naudojama ta pati PE failo struktūra, kold prieinama iki failo pradžios taško (Entry Point). Tada iškviečiamas mscoree.dll, kuris susiranda .text sekcijoje esančias CLR antraštę ir toliau vykdo failą.



3 pav. PE (CLR) failo sruktūra

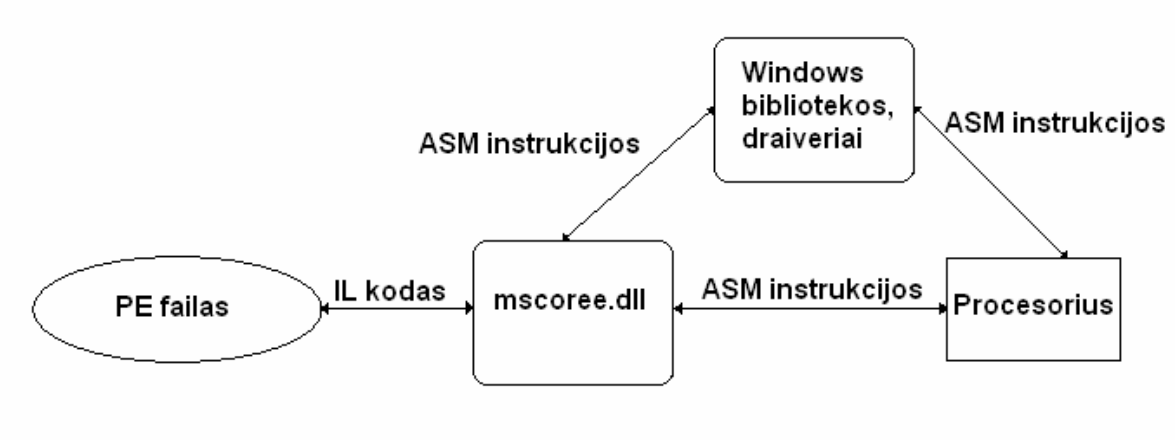
CLR failo struktūra (3 pav.) visada būna tik šitos trys sekcijos, CLR antraštė nuo kurios ir prasideda tokio tipo failo analizė yra .text sekcijos pradžioje. Taigi galima teigti kad CLR failas taspats PE failas .text sekcijoje turintis visa reikiamą informaciją, kad mscoree.dll galėtų jį vykdyti. Plačiau CLR failo sandarą panagrinėsime toliau einančiuose punktuose.

2.2.2 Informacijos srautų analizė



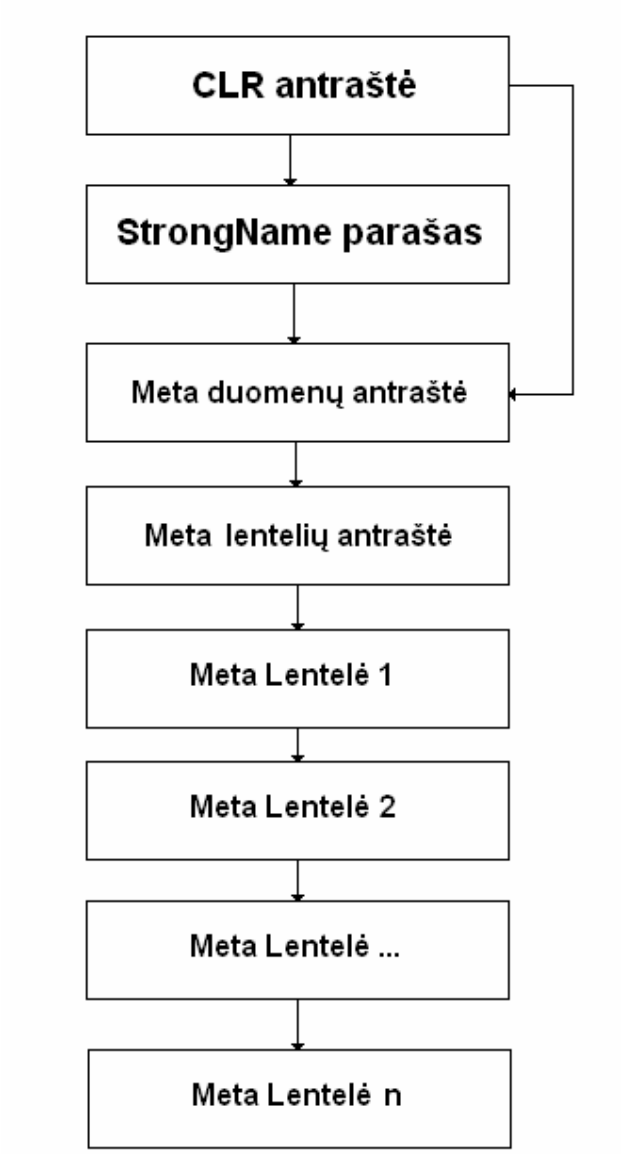
4 pav. PE failo išoriniai srautai

PE failo išoriniai srautai parodyti 4 pav., kur jis tiesiogiai asemblerio instrukcijomis gali kreiptis į procesorių arba į Windows sisteminių bibliotekų funkcijas ar tvarkykles



5 pav. PE (CLR) failo išoriniai srautai

PE (CLR) failo išoriniai srautai parodyti 5 pav., kur IL (Intermediate Language) kodas paimamas ir vykdomas mscoree.dll bibliotekos, priklausomai nuo IL kodo instrukcijų pagrindinė Framework .NET biblioteka kreipiasi į procesorių ar i Windows sistemines bibliotekas ir tvarkykles.



6 pav. CLR failo vidiniai srautai

CLR failo vidiniai srautai pavaizduoti 6 paveikslėlyje. CLR antraštės einama į StrongName parašo tikrinimą, jei StrongName parašo nėra einama tiesiai į meta duomenų antrašte (MetaData Header), toliau kreipiamasi į meta lentelių antraštę (MetaTable Header), kur randama informacija kokios galimos meta lentelės (Meta Tables), jų adresai dydžiai. Išviso galimos 44 lentelės, bet dažniausiai naudojama 12-15 lentlių, jouse saugoma įvairi informacija apie metodus, kintamuosius, adresai nukreipiantys kiekvieną metodą į IL koda ir t.t.

2.2.3 Saugomi duomenys

1 lentelė CLR antraštės duomenys

Nr.	Pavadinimas	Paaiškinimas
1	BytesReserved	Pirmi aštuoni baitai rezervuoti, pirmi keruti iš jų tai tikras virtualus adresas - RVA (Real Virtual Address) į mscoree.dll importą.
2	CLRsize	CLR antraštės dydis baitais
3	MajorRuntimeVersion	
4	MinorRuntimeVersion	
5	MetaDataRVA	Meta duomenų antraštės virtualus adresas
6	Flags	
7	EntryPointToken	
8	Resources	Resursų direktorijos virtualus adresas
9	ResourcesSize	Resursų direktorijos dydis
10	StrongName	StringName parašo virtualus adresas
11	StrongNameSize	StringName parašo ilgis
12	CodeManager	
13	CodeManagerSize	
14	VTableFxiups	
15	VTableFxiupsSize	
16	ExportAdressTable	
17	ExportAdressTableSize	
18	ManagenativeHeader	
19	ManagenativeHeaderSize	

1 lentelėje matome CLR antraštėje saugomus duomenis, kuriant sistemą svarbu rasti Meta duomenų antraštės adresą, kad galėtume toliau nagrinėti failą. Taip pat svarbus StrongName adresas. StrongName – tai 128 baitų, 2.0 .NET Framework versijoje jau ir 256 baitų raktas įterpiamas kompiliuojant failą. Šis parašas skirtas apsaugoti failą nuo modifikavimų ar virusų. Deja jis lengvai gali būti pašalintas prilyginus StrongName ar StrongNameSize nuliui. Kadangi obfuskatorius įneša pakeitimus į failą ir norint po obfuskavimo kad failas veiktų reikia arba pašalinti arba įterpti naują raktą kreipiantis į mscoree.dll StrongNameSignatureGeneration funkciją.

2 lentelė Meta duomenys

Nr.	Pavadinimas	Paaiškinimas
1	MetaSign	4 baitai, keturios raidės „BSJB“ pagal kurias galime atpažinti ar pasitikrinti kad tikrai esame Meta duomenų antraštės pradžioje
2	MajorVersion	
3	MinorVersion	
4	VersionString	.NET Framework versija, kuria buvo sukompiliuotas failas
5	NumOfStreams	srautų skaičius
6	1 srauto antraštė	
7	2 srauto antraštė	
8	... srauto antraštė	
9	n srauto antraštė	

Kiekvieno srauto antraštė turi tokius duomenis:

1. srauto adresas
2. dydis
3. pavadinimas
4. tipas

Teoriškai srautų gali būti daug, bet paprastai būna penki (3 lentelė). Virtualūs adresai į srautų duomenis skaičiuojami prie meta duomenų virtualaus adreso pridedant srauto adresą.

3 lentelė Srautų pavadinimai

Nr.	Pavadinimas	Srauto adreso paaiškinimas
1	~#	nurodo meta lentelių antraštės adresą.
2	#US	(UserStrings) nurodo vartotojo naudojamo teksto adresą
3	#Strings	nurodo metodų, parametrų vardų adresą, kurie saugomi teksto pavidale
4	#GUID	nurodo adresą į GUI interfeisą jei toks yra
5	#Blob	nurodo adresą į virtualų adresą į mscore.dll bibliotekos importą

4 lentelė Meta lentelių antraštės duomenys

Nr.	Pavadinimas	Paaiškinimas
1	Reserved	
2	MajorVersion	
3	MinorVersion	
4	HeapOffsetSizes	
5	RIDPlaceholder	
6	MaskValid	8 baitų skaičius, kuriuos pavertus į 64 bitus skaičiuojant nuo galo galime sužinoti ar meta lentelė egzistuoja , 1 – egzistuoja, 0 - neegzistuoja
7	MaskSorted	
8	Table1NrOfRows	meta lentelės eilučių skaičius
9	Table2NrOfRows	meta lentelės eilučių skaičius
10	Table…NrOfRows	meta lentelės eilučių skaičius
11	Table.n.NrOfRows	meta lentelės eilučių skaičius

Teoriškai maksimaliai galimi 64 lentelių eilučių duomenys. Bet yra tik 44 žinomos lentelės ir paprastai naudojama tik 12-15 lentelių. Jei lentelė neegzistuoja – tai nėra ir jai priklausančių duomenų apie jos eilučių skaičių.

Galimos meta lentelės ir juose saugomi duomenys pavaizduoti 5,6 lentelėse.

5 lentelė Meta lentelės

Nr.	Pavadinimas	Duomenys
1	Module	Generation - Name - Mvid – EncId - EncBaseId
2	TypeRef	ResolutionScope – Naem - NameSpace
3	TypeDef	Flags – Name – NameSpace – Extends –FieldList - MethodList
4	FieldPtr	Field
5	Field	Flags – Name – Signature
6	MethodPtr	Method
7	Method	RVA – ImpFlags – Flags – Name – Signature - ParamList
8	ParamPtr	Param
9	Param	Flags – Sequence - Name
10	InterfaceImpl	Class – Interface
11	MemberRef	Class – Name - Signature
12	Constant	Type – Parent - Value
13	CustomAtribute	Type – Parent - Value
14	FieldMarshal	Parent - Native
15	Permission	Action – Parent - PremissionSet
16	ClassLyout	PackingSize – ClassSize - Parent
17	FieldLayout	Offset - Field
18	StandAloneSig	Signature
19	EventMap	Parent - EventList
20	EventPtr	Event
21	Event	EventFlags – Name - EventType

6 lentelė Meta lentelės

Nr.	Pavadinimas	Duomenys
22	PropertyMap	Parent - PropertyList
23	PropertyPtr	Property
24	Property	PropFlags – Name - Type
25	MethodSemantics	Semantic – Method - Association
26	MethodImpl	Class – MethodBody - MethodDeclaration
27	ModuleRef	Name
28	TypeSpec	Signature
29	ImplMap	MappingFlags – MemberForwarded – ImportName - ImportScope
30	FieldRVA	RVA - Field
31	ENCLog	Token - FuncCode
32	ENCMap	Token
33	Assembly	HashAlg – MajorVersion – MinorVersion – BuildNumber – RevisionNumber – Flags – PublicKey – Name - Locale
34	AssemblyProcessor	Processor
35	AssemblyOS	OSPlatformId – OSMajorVersion - OSMinorVersion
36	AssemblyRef	MajorVersion – MinorVersion – BuildNumber – RevisionNumber – Flags – PublicKeyOrToken – Name – Locale - HashValue
37	AssemblyRefProcessor	Processor - AssemblyRef
38	AssemblyRefOS	OSPlatformId – OSMajorVersion – OSMinorVersion - AssemblyRef
39	File	Flags – Name - HashValue
40	ExportedType	Flags – TypeDefId – Typename – TypeNamespace - TypeImpl
41	ManifestResource	Offset – Flags – Name - Implementation
42	NestedClass	NestedClass - EnclosingClass
43	TypeTyPar	Number – Class – Bound - Name
44	MethodTyPar	Number – Method – Bound - Name

Kuriant obfuskavimo sistemą informacija turėtų būti renkama iš TypeDef, Method, Param lentelių. Todėl jų duomenis panagrinėsime plačiau.

TypeDef – duomenys Name (žr. 5 lentelę, nr.3) saugo adresą į modulių (class) vardus, kuriuos kuriant sistemą galima būtų obfuskuoti. Kuo daugiau modulių tuo daugiau eilučių, eilutės ilgis 14 baitų.

Method – duomenys Name (žr. 5 lentelę, nr.7) saugo adresą į metodų vardus, kuriuos kuriant sistemą galima būtų obfuskuoti. Kuo daugiau metodų tuo daugiau eilučių, eilutės ilgis 14 baitų.

Param - duomenys Name (žr. 5 lentelę, nr.9) saugo adresą į kintamųjų vardus, kuriuos kuriant sistemą galima būtų obfuskuoti. Kuo daugiau kintamųjų tuo daugiau eilučių, eilutės ilgis 6 baitai.

2.2.4 Analizės išvados

Šioje dalyje buvo išanalizuota CLR failo sandaros dalys reikalingos sukurti obfuskacijos sistemai. Pagrindinę failo dalį sudaro meta lentelės (Meta Tables) ir juose laikoma informacija, ką kuriama sistema turi pasiimti iš failo ir analizuoti.

2.3 Pasaulio literatūros šaltiniuose pateiktų sprendimų problemai spręsti lyginamoji analizė

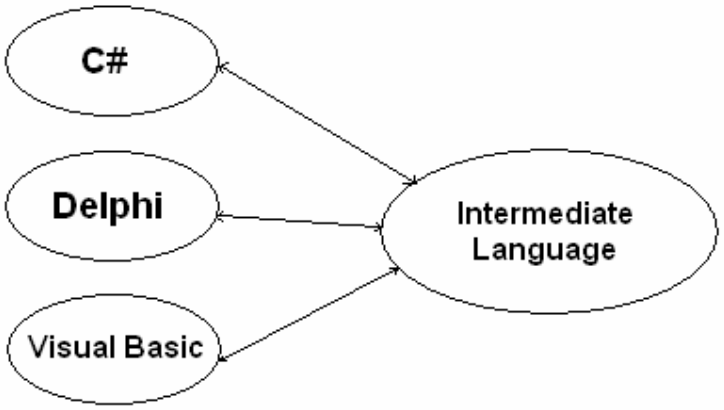
Microsoft .NET Framework technologijomis paremtų failų (CLR failų) atviras kodas, nesaugumas ir atvirkštinės inžinerijos (reverse engineering) galimybė labai paplitusi ir plačiai diskutuojama problema, daugelis žmonių norėtų žinoti kaip apsaugoti savo intelektualią nuosavybę. Vienas iš atsakymų – kodo obfuskavimas.

Smulki sukompiliuoto CLR sandara buvo aprašyta veiklos analizėje, egzistuoja dvi pagrindinės .NET sukompiliuoto failo dalys:

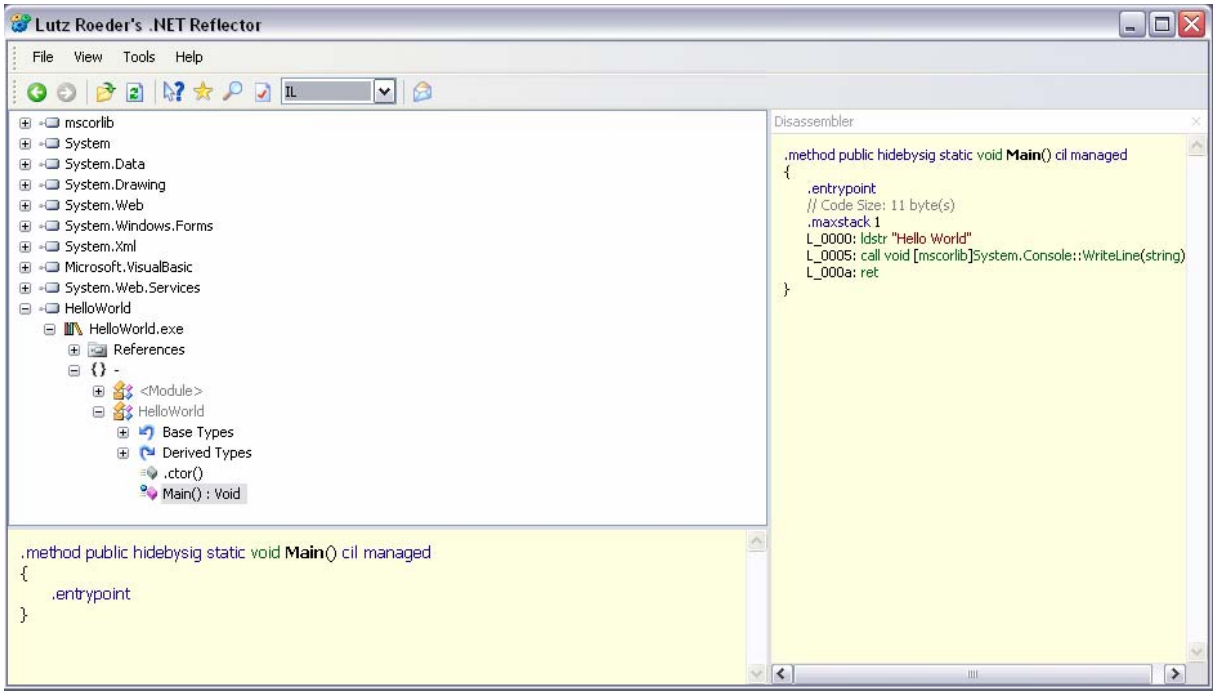
- Meta duomenys (MetaData)
- IL (Intermediate Language) kodas

Kaip jau žinome meta duomenis sudaro vardai, lentelių adresai - tai beveik visa CLR failo struktūra. Šie duomenys palengvina įsilaužėliui orientuotis faile, nes matomi visi klasių, metodų, kintamųjų vardai.

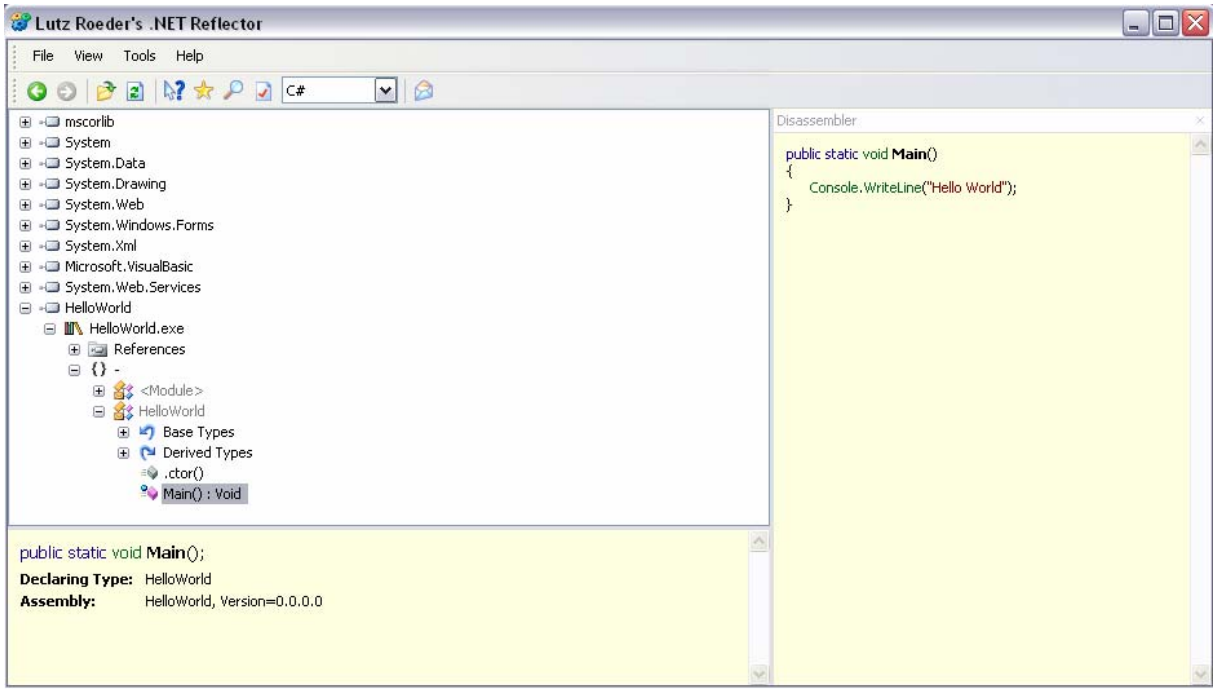
IL (Intermediate Language) – tai paprasta .NET Framework žemo lygio kalba, kuri dekompiliatorių pagalba lengvai gali būti verčiama (7 pav.) į bet kokią aukšto lygio .NET kalbą (C#, Delphi, Visual Basic ir t.t.). „Lutz Roeders .NET Reflector“ – tai dekompiliatorius, kurį nemokamai galima atsisiųsti iš http://www.aisto.com/roeder/dotnet/. Naudosime jį demonstracijai. 8, 9, 10, 11 monitoriaus paveikslėliuose galime matyti sukompiliuoto HellowWorld.exe (Microsoft Visual Studio .NET pavyzdys) iš IL kodo dekompiliuotą į C#, Delphi ir Visual Basic kalbas.



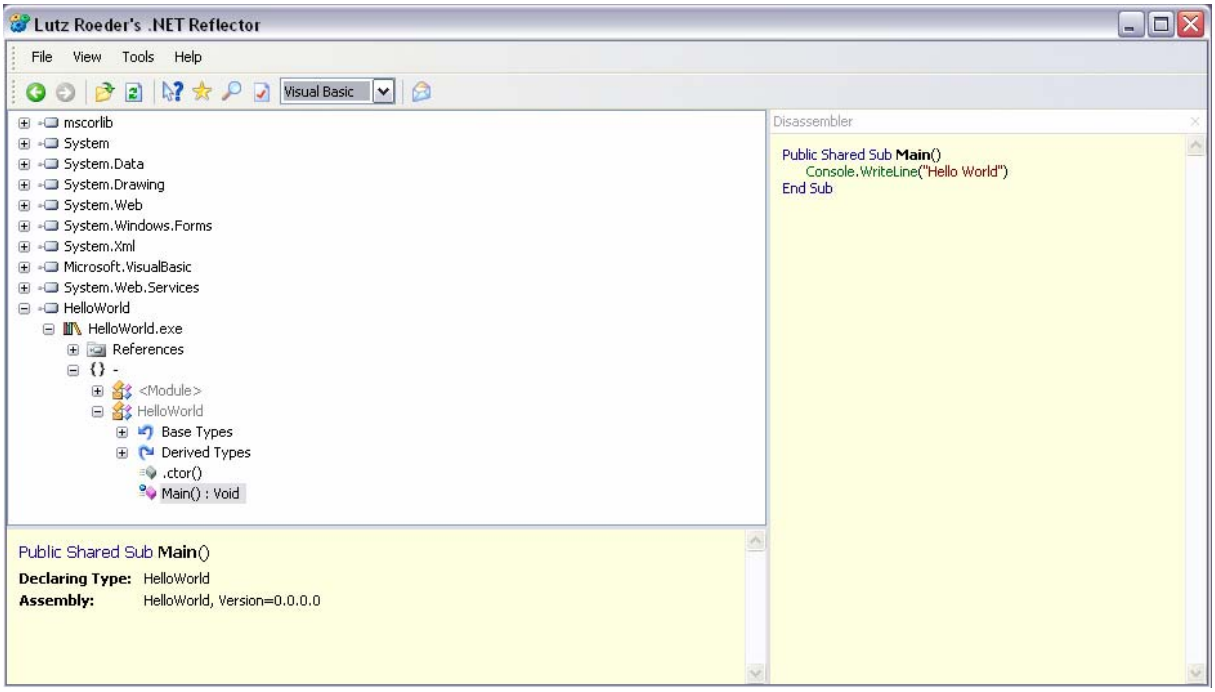
7 pav. Kompiliavimas į IL kodą



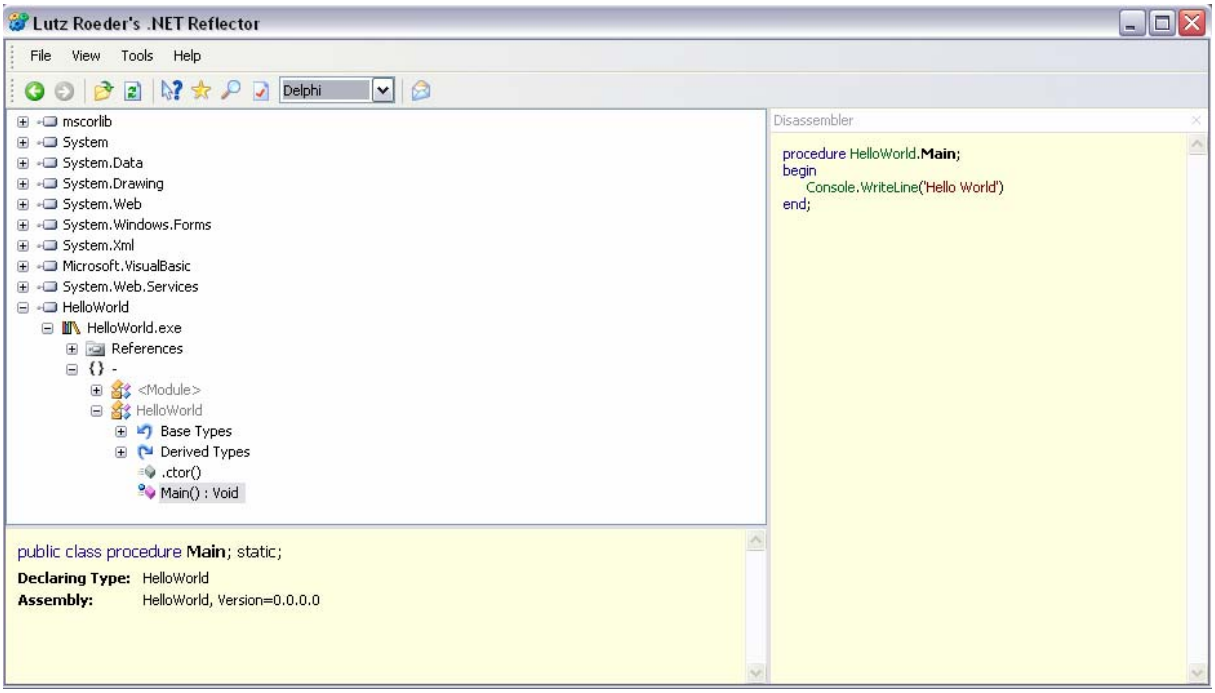
8 pav. IL kodas



9 pav. C# kodas



10 pav. Visual Basic kodas



11 pav. Delphi kodas

Obfuskatorių veikimo principas: siekiama pakeisti meta duomenų ar IL kodo sandarą, kad jie tapti kuo sunkiau nuskaitomi ar suprantami netgi dekompiliavus.

Klasių , metodų, kintamųjų vardų keitimo metodas (7 lentelė, nr. 1): svarbiausias ir patikimiausias būdas keisti metodų, funkcijų, kintamųjų vardus. Vardai tai vienintelis dalykas, kurį galima keisti nekliudant viso kito, t.y išvengiant dekompiliacijos. Svarbu juos pakeisti taip, kad nebūtų įmanoma to atstatyti. Pavyzdžiui jei obfuskatorius paims ir pakeis visu vardų kiekvieną raidę sekančia – visą tai galima bus lengvai išsiaiškinti ir atstatyti. Geresnis obfuskatorius butų naudojantis HASH algoritmus (tokius kaip SHA1 ar MD5) ar generuojantis naują vardą iš seno – bet tai vėlgi nėra visiškai neatstatoma. Geriausias obfuskatorius pašalintų senus vardus ir pakeistų juos naujais visiškai nesuprantamais. Pvz: int foo(int i), string bar(string s) ir int bar(string s) būtų pakeisti int a(int a), string a(string a) ir int a(string a).

Vartotojo teksto (User Strings) šifravimo metodas: būdas reikalaujantis, dekompiliuoti obfuskuojamą failą į IL kodą. Į dekompiliuotą IL kodą reikėtų įvesti funkciją, kuri būtų paleidžiama pradžioje ir iššifruoja vartotojo tekstą, arba po kiekviena ldstr instrukcija, kuri užkrauna tekstą išvedimui, būtų kreipiamasi į iššifravimo funkciją (7 lentelė, nr.2). Deja šis metodas gali būti atstatytas įsilaužėlio, suradus iššifravimo funkciją ir algoritmą.

Failo kodo užšifravimas (7 lentelė, nr.3.): yra keli įgyvendinimo būdai. Pirmas būdas reikalaujantis, dekompiliuoti obfuskuojamą failą į IL kodą. Į dekompiliuotą IL kodą reikėtų įvesti funkciją, kuri iššifruotų jau užšifruotą IL kodą, t.y. yra veiktų kaip paleidėjas (Loader). Antras būdas nereikalauja dekompiliuoti failo į IL kodą. Reikėtų tiesiog padaryti paleidėją ne IL kodui bet visam PE failui, t.y. paleidėjas net nebūtų IL kodas, jis būtų įterptas įvedus naują sekciją į failą ir nukreipus failo pradžios tašką (Entry Point) į užkrovėją, jis iššifruotų CLR failo dalis dar prieš failui kreipiantis į mscoree.dll.

Vis dėl to failo kodo šifravimo metodas nėra toks geras būdas dėl kelių priežasčių. Viskas kas gali būti paleista – gali būti ir nukopijuota. Šifravimo algoritmas gali būti surastas ir failas iššifruotas.

.

7 lentelė Palyginimo kriterijai

Nr.	Metodas	Originalus kodas	Obfuskuotas kodas	Atstatomumas
1	Keičiami metodų, funkcijų, kintamųjų vardai	MyFileReader.FileRead(string fileName)	a.b(string c)	Priklauso nuo pakeitimo būdo.
2	Vartotojo teksto šifravimas (User Strings)	ldstr “Hello World!” call Console::WriteLine(string)	ldstr “AsdfAFGdgzDSf=” call string Utils.Decrypt(string) call Console.WriteLine(string)	Visada galima atstatyti.
3	Kodo šifravimas	-	Netgi dekompiliavus kodą bus matomi nesuprantami baitai.	Visada galima atstatyti.

Egzistuoja ir daugiau metodų bet jie arba neefektyvūs arba netinkami vartotojui. Naudojant visus metodus esančius 7 lentelėje kartu, būtų pasiektas pats geriausias variantas.

Obfuskatoriai veikia failus keliasi būdais:

- obfuskuojamas jau sukompiliuotas failas;
- failas dekompiliuojamas į IL kodą, vykdomi pakeitimai ir sukompiliuojamas jau naujas failas.

Žemiau 8 lentelėje pateikti gamintojų produktai ir jų produktų atliekamų funkcijų palyginimai. Įdomus dalykas, kad beveik kiekvienas gamintojas taip pat gamina ir siūlo vartotojams dekompiliatorius.

8 lentelė Gamintojų esamų sprendimų palyginimas

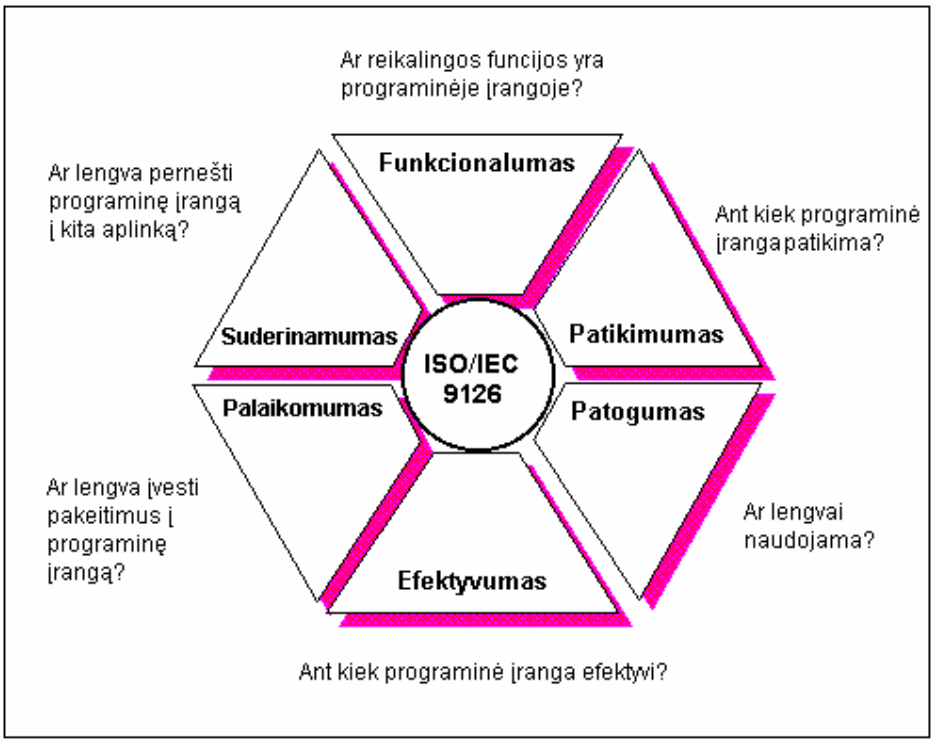
Gamintojas	Keičiami metodų, funkcijų, kintamųjų vardai	Vartotojo teksto šifravimas (User Strings)	Kodo šifravimas	Kiti Metodai
Remotesoft Salamander .NET Obfuscator	yra	nėra	nėra	yra
Deploy.NET	nėra	yra	yra	nėra
Demeanor for .NET	yra	yra	nėra	nėra
Spices .NET Suite	yra	yra	nėra	nėra
IL Obfuscator 2.0	yra	nėra	nėra	nėra
XenoCode .NET	yra	yra	nėra	nėra

2.4 Kokybės kriterijų apibrėžimas

ISO (International Standard Organization) - tai tarptautinės standartų organizacijos kokybės standartai. ISO standartas labiausiai paplitęs Vakarų Europoje, JAV. Kinija vis labiau pereina prie ISO standartų. Šiandien daugelis įmonių Lietuvoje dirba pagal tarptautinius ISO standartų reikalavimus.

ISO/IEC 9126 standarto paskirtis – įvertinti programos kokybę. ISO/IEC 9126 standartas neteikia reikalavimų programos kokybei, o tik apibrėžia kokybės modelį, kuris priimtinas bet kokiai programai. Šio standarto kriterijais bus remiamasi kuriant šiame darbe aprašomą projektą.

Išskiriami šeši produkto kokybę charakterizuojantys veiksniai (12 pav.) – taip pat kiekvienas iš jų išskirstomas detaliau (9 lentelė).



12 pav. ISO/IEC 9126

9 lentelė ISO/IEC 9126 veiksniai

Pagrindinis veiksnys	Detalesnis išskyrimas	Paaiškinimas
Funkcionalumas	Tinkamumas	Tam skirtų užduotis atliekančių funkcijų atitikimas.
	Tikslumas	Žadėtų rezultatų atitikimas.
	Operatyvumas	Sąveika su kitomis sistemomis.
	Teisėtumas	Tam tikrų standartų, teisės aktų, konvencijų ir.t.t. atitikimas.
	Saugumas	Neautorizuoto panaudojimo galimybė.
Patikimumas	Užbaigtumas	Klaidų dažnumo atitikimas.
	Klaidų galimybė	Klaidų tolerancija, leistinas klaidų skaičius.
	Atstatomumas	Prarastų duomenų dėl įvairių klaidų atstatymo galimybė.
Patogumas	Aiškumas	Vartotojo sąsajos logiškumas ir suprantamumas.
	Mokymosi galimybė	Vartotojo mokymo galimybė.
	Operatyvumas	Programos operatyvumas vartotojo atžvilgiu.
Efektyvumas	Greitumas	Funkcijų vykdymo greitis.
	Resursų naudojimo greitumas	Resursų naudojimo kiekio ir ju naudojimo funkcijų vykdymo greitis.
Palaikomumas	Analizės galimybė	Diagnostikos arba klaidos vietos identifikavimo galimybė.
	Pakeitimo galimybė	Reikiamų vietų pakeitimo ar pašalinimo galimybė.
	Stabilumas	Rizikingų vietų ir netikėtų jas įtakojančių veiksnių galimybė.
	Testavimo galimybė	Pakeitimų ar pataisymų testavimo galimybė.
Suderinamumas	Pritaikomumas	Automatinė prisitaikymo prie įvairių sistemos variantų galimybė.
	Diegimo galimybė	Diegimo įvairiuose sistemos variantuose galimybė.
	Atitinkamumas	Įvairių suderinamumo standartų, konvencijų atitikimas.
	Pakeičiamumas	Programinės įrangos pakeitimo galimybė vietoje kitos.

2.5 Analizės išvados.

1. Analizei pasirinktas struktūrinis projektavimas nes jis labiausiai atitinka tyrimo objekto poreikius;
2. Išanalizuotas tyrimo objektas , CLR failo ir meta lentelių sandara.
3. Aprašyti ir palyginti literatūros šaltiniuose pateikti obfuskacijos metodai. Projektuojamai sistemai kurti pasirinktas (9 lentelėje, nr.1) metodas, kai keičiami obfuskuojamo failo metodų, funkcijų, kintamųjų vardai;
4. Apibrėžti kuriamos sistemos kokybės kriterijams buvo pasirinktas ISO/IEC 9126 standartas.

3. PROJEKTO DALIS

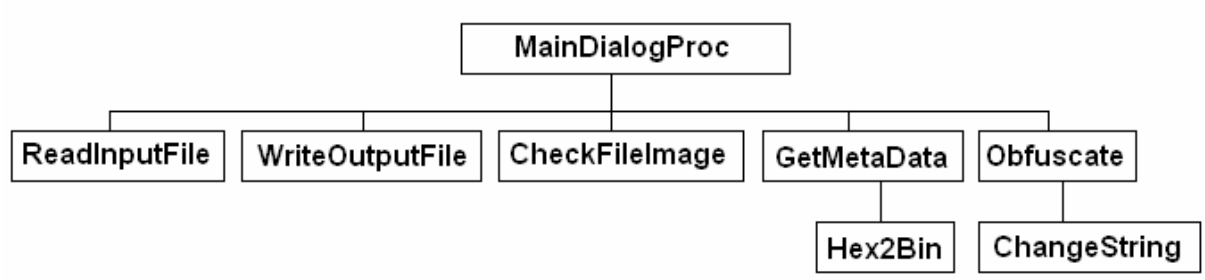
3.1 Techninė užduotis

Projekto užduotis: pasinaudojant struktūrinio projektavimo modeliu sukurti obfuskatorių. Pasirinktai vartotojui įvedus failą išvesti obfuskuotą pagal analitinėje dalyje pasirinktą metodą. Reikia kompiuterizuoti šiuos etapus: failo įvedimas, analizė, obfuskacija, išvedimas. Vartotojai taip pat kelia reikalavimus, kad sistema padarytų atsarginę originalaus failo kopiją.

3.1.1 Reikalavimų modelis

3.1.1.1 Kompiuterizuojamų funkcijų hierarchija

Sistemos kompiuterizuojamų funkcijų hierarchija pavaizduota 13 pav.



13 pav. Funkcijų hierarchija

MainDialogProc – pagrindinė funkcija, dialogo valdymas.

ReadInputFile – nuskaito įvedamą failą į atmintį.

WrueOutPutFile – rašo iš atminties į failą.

CheckFileImage – tikrina ar įvestas failas tinkamas.

GetMetaData – surenka Meta informaciją apie failą.

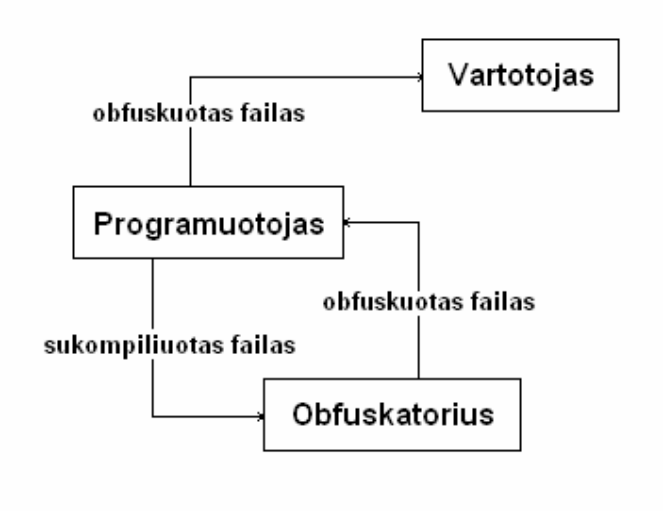
Hex2Bin – verčia iš šešioliktainės sistemos į dvejetainę.

Obfuscate – obfuskuooja failą esantį atmintyje.

ChngeString – pakeičia tekstą į nurodytą.

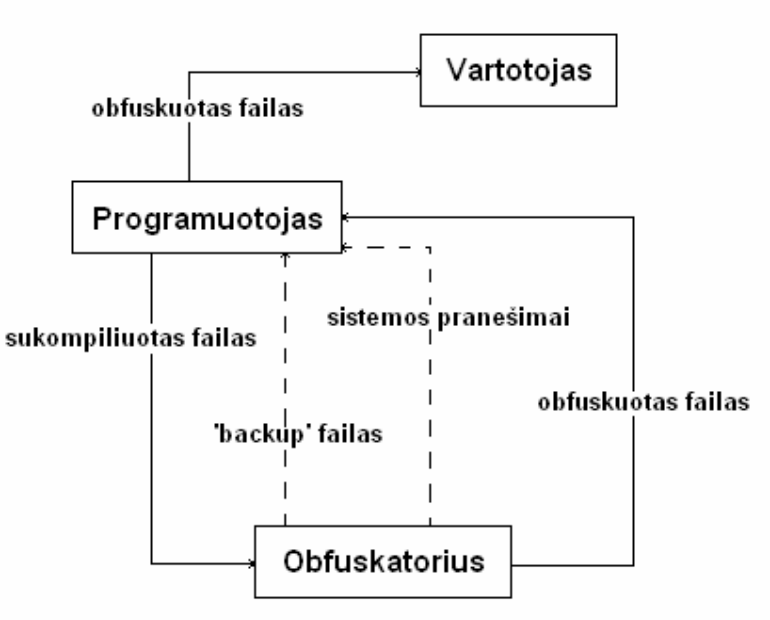
3.1.1.2 Kompiuterizuojamos sistemos duomenų srautų diagrama

Aukščiausio lygmens DSD (14 pav.) vaizduoja obfuskuojamo failo aplinką. Pagal diagramą galime matyti kokie elementai yra sistemos išorėje ir ryšį su jais.



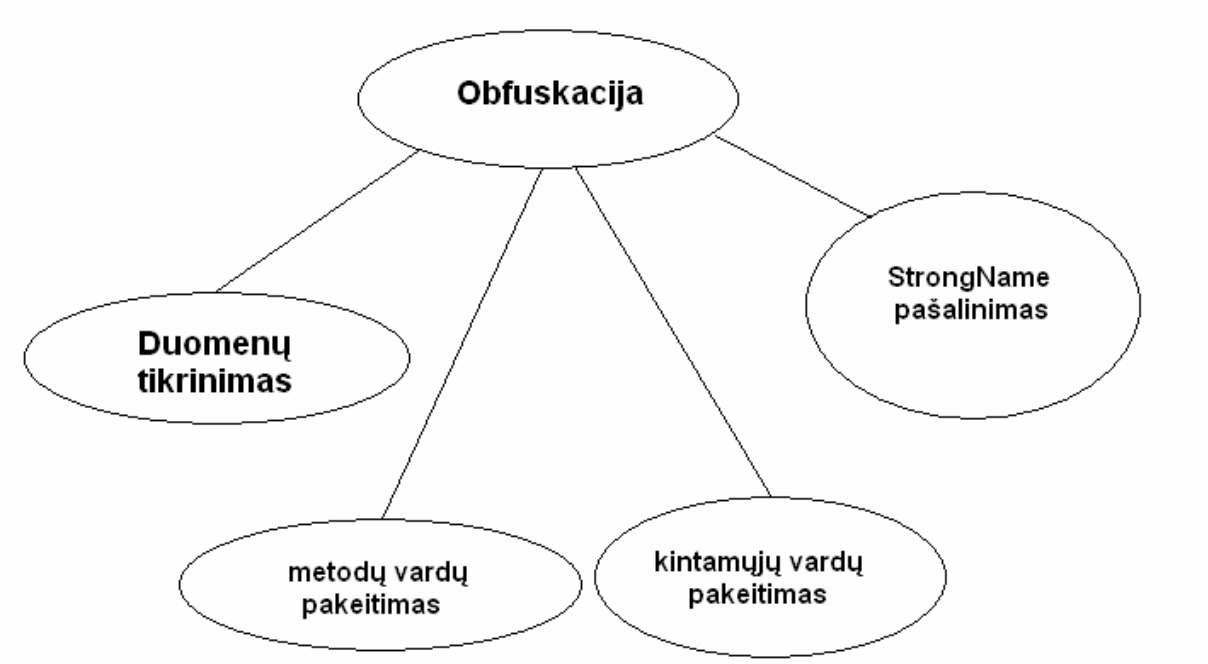
14 pav. Aukščiausio lygmens DSD

Nulinio lygio diagrama (15 pav.) detalizuoja aukščiausio lygmens DSD.



15 pav. Nulinio lygmens DSD

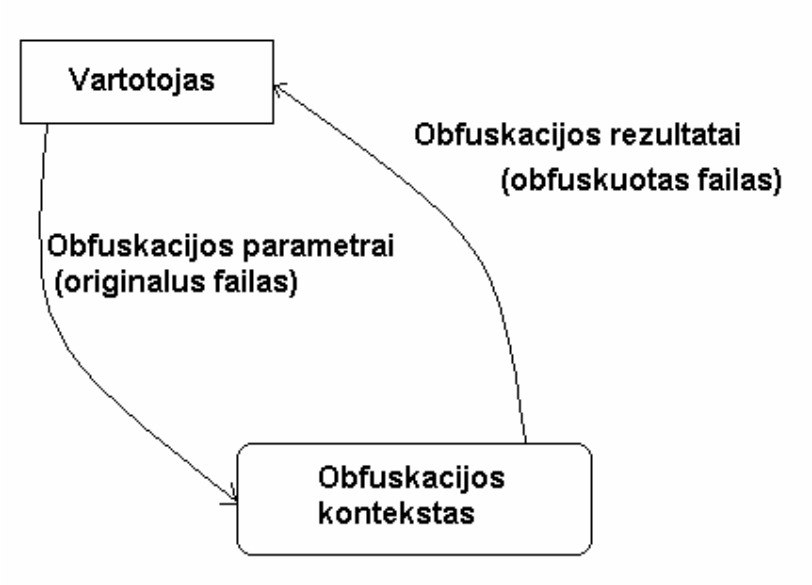
Kiekvieną procesą, pavaizduotą nulinio lygio diagramoje, paprastai sudaro keli subprocesai. Smulkiau detalizuojame obfuskacijos procesą (16 pav.).



16 pav. Pirmo lygmens DSD

3.1.1.3 Vartotojo interfeiso modelis

Kadangi vartotojas bus programuotojas, kuris nori obfuskuoti savo sukompiliuotą failą, sistema turi būti kuo paprastesnė vartotojui ir neapkrauti jo pildomomis programos vartojimo instrukcijomis ar žiniomis. Tiesiog vartotojui nurodžius failą ir patvirtinus vykdymą, failas automatiškai turi būti apdirbtas ir išvestas tolimesniam naudojimui kaip parodyta veiklos konteksto diagramoje (17 pav.). Vidinius sistemos reikalavimus funkcionalumui, nematomus vartotojui, galima būtų pavaizduoti pateiktame veiklos įvykių sąraše 10 lentelėje.



17 pav. Veiklos konteksto diagrama

10 lentelė Veiklos įvykių sąrašas

Eilės Nr.	Įvykio pavadinimas	Įeinantys/išeinantys informacijos srautai
1	Vartotojas paduoda sistemai failą.	Vartotojo failas (in)
2	Sistema formuoja pranešimus vartotojui.	Pranešimai (out)
3	Sistema sukuria atsarginį failą.	Atsarginis failas (out)
4	Sistema atlieka failo tikrinimą.	Pranešimai (out)
5	Sistema atlieka failo ofbuskaciją.	Obfuskuotas failas (out)

Reikalavimai vartotojo sąsajai:

- lengvai skaitoma – vartotojui nereikėtų turėti daug įgūdžių priprantant dirbti su sistema;
- grafinė sąsaja - turi būti GUI o ne Console variantas;
- paprastas panaudojimas – kad nebūtų sudėtinga naudoti;
- profesionali išvaizda – nesudėtinga, akcentuojanti pagrindinius dalykus, kuriuos galima atlikti, neperkrauta spalvom, dalykiška;
- neįkyri sąsaja - nereikalaujanti pastoviai ką nors kelis kartus patvirtinti.

3.1.1.4 Reikalavimai sistemos funkcionalumui

Tinkamumas (specifikuotiems uždaviniams): sistema tinka analizuoti CLR failo struktūrą rinkti meta lentelių duomenis, lengvai gali būti ištobulinta iki CLR failo peržiūrėjimo programos.

Tikslumas (rezultato): rezultato tikslumas labai svarbus, nes dirbama su failais, kuriuos sistemos vartotojas parduos arba platins kitiems vartotojams.

3.1.1.5 Reikalavimai sistemos patikimumui

Užbaigtumas: sistema turi būti pilnai užbaigta, nes dirbs su paleidžiamaisiais failas, kuriuos vartotojas naudos įvairiems poreikiams.

Atstatomumas: prieš darant kokius nors pakeitimus visada padaroma originalaus vartotojo failo kopija, todėl visada yra galimybė atstatyti originalų failą.

Klaidų tolerancija: vartotojo padarytos klaidos turi būti lydimos pranešimais skirtais vartotojui. Kitų klaidų ar netikslumų tolerancija minimali.

3.1.1.6 Reikalavimai sistemos patogumui

Galimybė apsimokyti: vartotojas be jokios pagalbos turėtų lengvai apsimokyti dirbti programa.

Suprantamumas: programos dialogas turi būti lengvai suprantamas.

3.1.1.7 Reikalavimai sistemos efektyvumui

Laiko parametrai: sistema turi veikti greitai ir efektyviai, netoleruojamos pauzės dėl nevykusių algoritmų

Išteklių naudojimas: turi naudoti minimaliai resursų, būti kuo mažiau priklausoma nuo kitų bibliotekų ar veiksnių įtakojančių programos veiklą, kreiptis tik į kelias Windows sistemos bibliotekas, visas duomenų apdorojimas maksimaliai susistemintas greitais algoritmais, kurie nenaudoja daug išorinių išteklių.

3.1.1.8 Reikalavimai sistemos priežiūros savybėms

Stabilumas: svarbus programos patikimumo rodiklis – todėl turi būti stabili.

Pakeitimų galimybės: turi būti lengva įvesti pakeitimus ar pataisymus.

Testavimo savybės: testuoti visose galimose Windows operacinėse sistemose.

3.1.1.9 Kiti reikalavimai sistemos funkcionalumui

Sistemos suderinamumas: sistema turėtų veikti bet kuriame kompiuteryje su Windows šeimos 9x ir naujesnėmis operacinėmis sistemomis, taip pat Framework .NET palaikymu.

Reikalavimai sistemos priežiūrai: sistema turi būti tobulinama išėjus naujoms Framework .NET versijoms ar atsiradus pakitimams CLR struktūroje, taip pat atsiradus naujiems metodams.

Reikalavimai saugumui: reikalavimų saugumui nėra – sistema nekomercinė ir laisvai prieinama.

Kultūriniai-politiniai reikalavimai: kadangi sistema daroma lietuvių kalba, bus palikta galimybe lengvai pakeisti lietuviškus pranešimus į kitą kalbą, nes esant vartotojo sąsajai anglų kalba ji taptų labiau prieinamesnė.

Reikalavimai techninei įrangai: kompiuteris nemažesniu nei Pentium 100 MHz procesoriumi. Kompiuteriams dirbantiems su Windows 98 ne mažiau 64 MB operatyviosios atminties. Kompiuteriams dirbantiems su Windows ME nuo 48 iki 64 MB darbinės atminties. Kompiuteriams dirbantiems su Windows NT 4.0 ne mažiau 64 MB darbinės atminties. Kompiuteriams dirbantiems su Windows 2000 nuo 96 iki 128 MB darbinės atminties, o dirbantiems su Microsoft Windows XP operatyvinė atmintis turėtų būti ne mažesnė už 128 MB.

3.1.2 Sistemos projektas

3.1.2.1 Informacinės įrangos projektas

Klasifikavimo ir kodavimo sistemos aprašymas: buvo pasirinkta Win32ASM (Windows 32 Assembler) programavimo kalba. Ji priskiriama prie žemo lygio programavimo kalbų, senesnė C kalba naudoja tą patį kompiliatorių. Dažniausiai šios kalbos naudojamos tvarkyklių arba šifruotojų („krypterių“) programavimui, kur mažai arba visai nereikalingas objektinis programavimas ir siekiama programos spartos ir optimalaus kodo, nepriklausančio arba mažai priklausančio nuo įvairų bibliotekų funkcijų. 11 lentelėje Win32ASM palyginimas su kitomis kalbomis. Reikia pabrėžti kad tik JAVA pilnai suderinama su kitomis operacinėmis sistemomis, C++ tik tam tikrais atvejais.

11 lentelė Programavimo kalbų palyginimas

Nr.	Kalba	Operacijų vykdymo greitis	Programos kodo optimalumas	Objektinio programavimo efektyvumas	Kodo suderinamumas	Priklausomybė nuo bibliotekų, klasių
1	ASM	MAX	MAX	MIN	Windows	MIN
2	C++	MAX	MID	MID	Windows,UNIX	MIN
3	Delphi	MAX	MID	MID	Windows	MID
4	Visual Basic	MIN	MIN	MAX	Windows	MAX
5	Java	MIN	MIN	MAX	Windows,UNIX	MAX

MAX – maksimaliai;
MID - vidutiniškai;
MIN – minimaliai;

12 lentelė Įėjimo informacijos aprašymas

Nr.	Srauto pavadinimas	Pateikimo forma	Šaltinis	Periodiškumas	Formos pavyzdys
1	Failo pasirinkimas	Elektroninė	Vartotojas	Pagal vartotojo norą	Sukompiliuotas CLR failas
2	Pakeisti metodų pavadinimus	Elektroninė	Vartotojas	Prieš vykdant	Pasirinkti arba ne
3	Pakeisti kintamųjų pavadinimus	Elektroninė	Vartotojas	Prieš vykdant	Pasirinkti arba ne
4	Pašalinti StronName parašą	Elektroninė	Vartotojas	Prieš vykdant	Pasirinkti arba ne
5	Vykdyti	Elektroninė	Vartotojas	Vieną kartą pasirinkus failą	Pasirinkti tik kai failas pasirinktas ir parametrai užduoti
6	Apie	Elektroninė	Vartotojas	Pagal vartotojo norą	Laisva forma

13 lentelė Rezultatinės informacijos aprašymas

Nr.	Srauto pavadinimas	Pateikimo forma	Periodiškumas
1	Failo pasirinkimas	Išvedamas monitoriuje pasirinkimo dialogas	Vienąkart po įvedimo
2	Pakeisti metodų pavadinimus	Pakeitimai vyksta išvedamame faile	Vienąkart po kiekvieno vykdymo
3	Pakeisti kintamųjų pavadinimus	Pakeitimai vyksta išvedamame faile	Vienąkart po kiekvieno vykdymo
4	Pašalinti StronName parašą	Pakeitimai vyksta išvedamame faile	Vienąkart po kiekvieno vykdymo
5	Vykdyti	Minitoriuje, naujo failo išvedimu ir atsarginio failo sukūrimu	Vienąkart po pasirinkimo
6	Apie	Išvedamas monitoriuje	Vienąkart po pasirinkimo

3.1.2.2 Programinės įrangos projektas

Sistemos architektūra

14 lentelė Funkcijos ir jų kintamieji

Nr.	Funkcijos pavadinimas	Kintamieji	Aprašymas
1	MainDialogProc	hWin :dword uMsg :dword wParam :dword lParam :dword local IsMethodFlagChecked :dword local IsParamFlagChecked :dword local IsStrongNameFlagChecked :dword	Pagrindinė funkcija, dialogo valdymas.
2	ReadInputFile	pFileName :dword pMem :dword dFSize :dword local hFile :dword local dBR :dword local dFS :dword local dMM :dword	Nuskaito įvedamą failą į atmintį.
3	WrueOutPutFile	pFileName :dword pMem :dword dFSize :dword local hFile :dword local dBW :dword local dFS :dword local dMM :dword	Rašo iš atminties į failą
4	CheckFileImage	pMem :dword	Tikrina ar įvestas failas tinkamas.
5	GetMetaData	pMem :dword	Surenka meta informaciją apie failą.
6	Hex2Bin	pHex :dword dLenght :dword pBinStrOut :dword	Verčia iš šešioliktainės sistemos į dvejetainę.
7	Obfuscate	pMem :dword dIsMethod :dword dIsParam :dword dIsStrongName :dword local dwSize: dword	Obfuskuooja failą esantį atmintyje.
8	ChngeString	pMem :dword	Pakeičia tekstą į nurodytą.

Programinės įrangos aprašymas

Kad programa veiktų, vartotojo kompiuteryje turi būti instaliuota operacine sistema Windows 98/ME2000/XP. Rekomenduojamos sistemos Windows 2000 arba XP. Microsoft Framework .NET instaliuoti nebūtina, programa įvykdys pakeitimus sėkmingai. Tik kad pratestuoti pakeistą failą turės būti įdiegta Microsoft Framework .NET.

Testavimo duomenų aprašymas

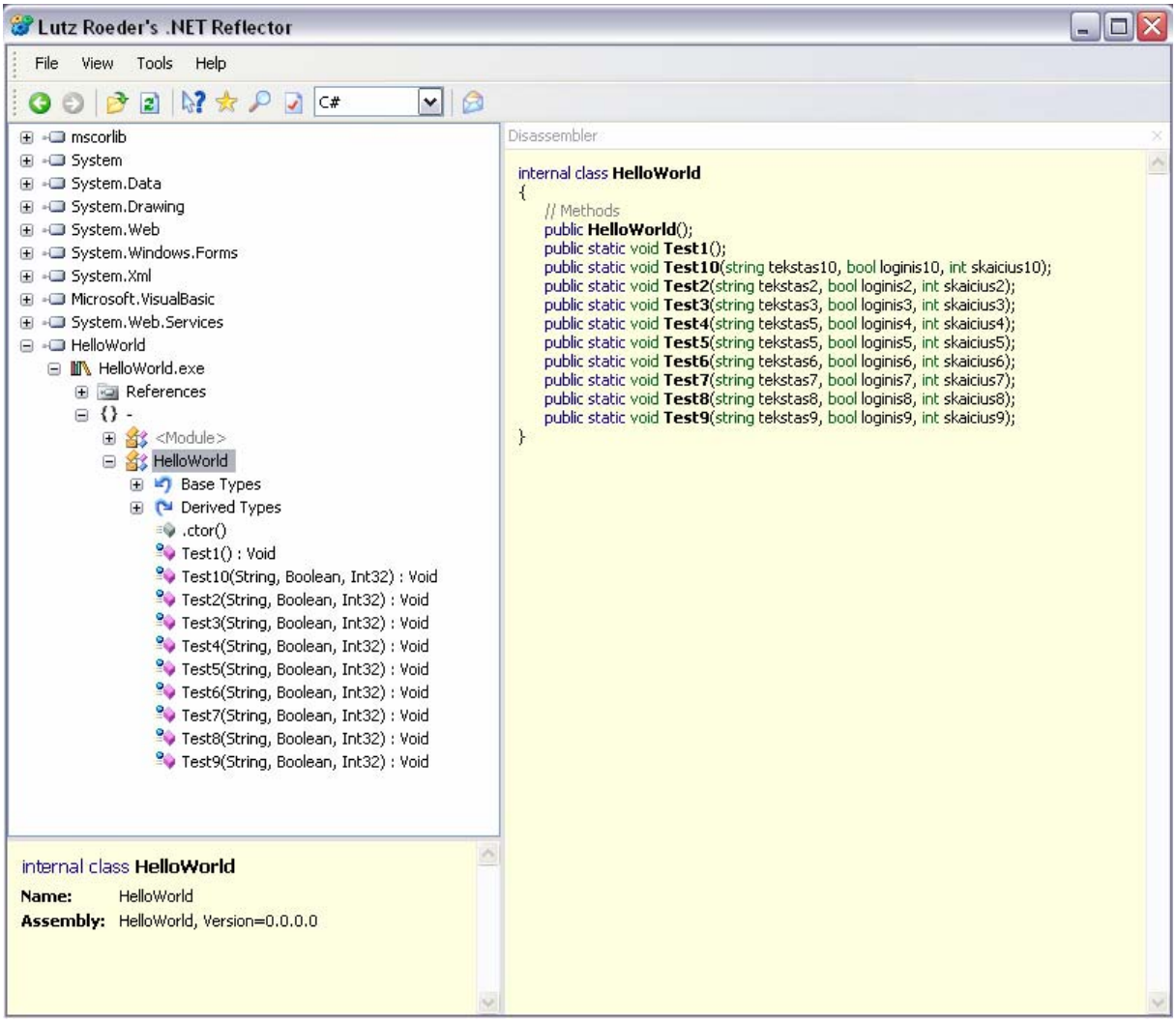
Testavimui buvo sukurtas failas, jo išeities kodas pateiktas žemiau.

HeloWorld.cs

```
class HelloWorld
{
		public static void Test1()
		{
				System.Console.WriteLine("Test String 1");
		}
		public static void Test2(string tekstas2,bool loginis2,int32 skaicius2)
		{
				System.Console.WriteLine("Test String 2");
		}
		public static void Test3(string tekstas3,bool loginis3,int32 skaicius3)
		{
				System.Console.WriteLine("Test String 3");
		}
		public static void Test4(string tekstas4,bool loginis4,int32 skaicius4)
		{
				System.Console.WriteLine("Test String 4");
		}
		public static void Test5(string tekstas5,bool loginis5,int32 skaicius5)
		{
				System.Console.WriteLine("Test String 5");
		}
		public static void Test6(string tekstas6,bool loginis6,int32 skaicius6)
		{
				System.Console.WriteLine("Test String 6");
		}
		public static void Test7(string tekstas7,bool loginis7,int32 skaicius7)
		{
				System.Console.WriteLine("Test String 7");
		}
		public static void Test8(string tekstas8,bool loginis8,int32 skaicius8)
		{
				System.Console.WriteLine("Test String 8");
		}
		public static void Test9(string tekstas9,bool loginis9,int32 skaicius9)
		{
				System.Console.WriteLine("Test String 9");
		}
		public static void Test10(string tekstas10,bool loginis10,int32 skaicius10)
		{
				System.Console.WriteLine("Test String 10");
		}
}
```

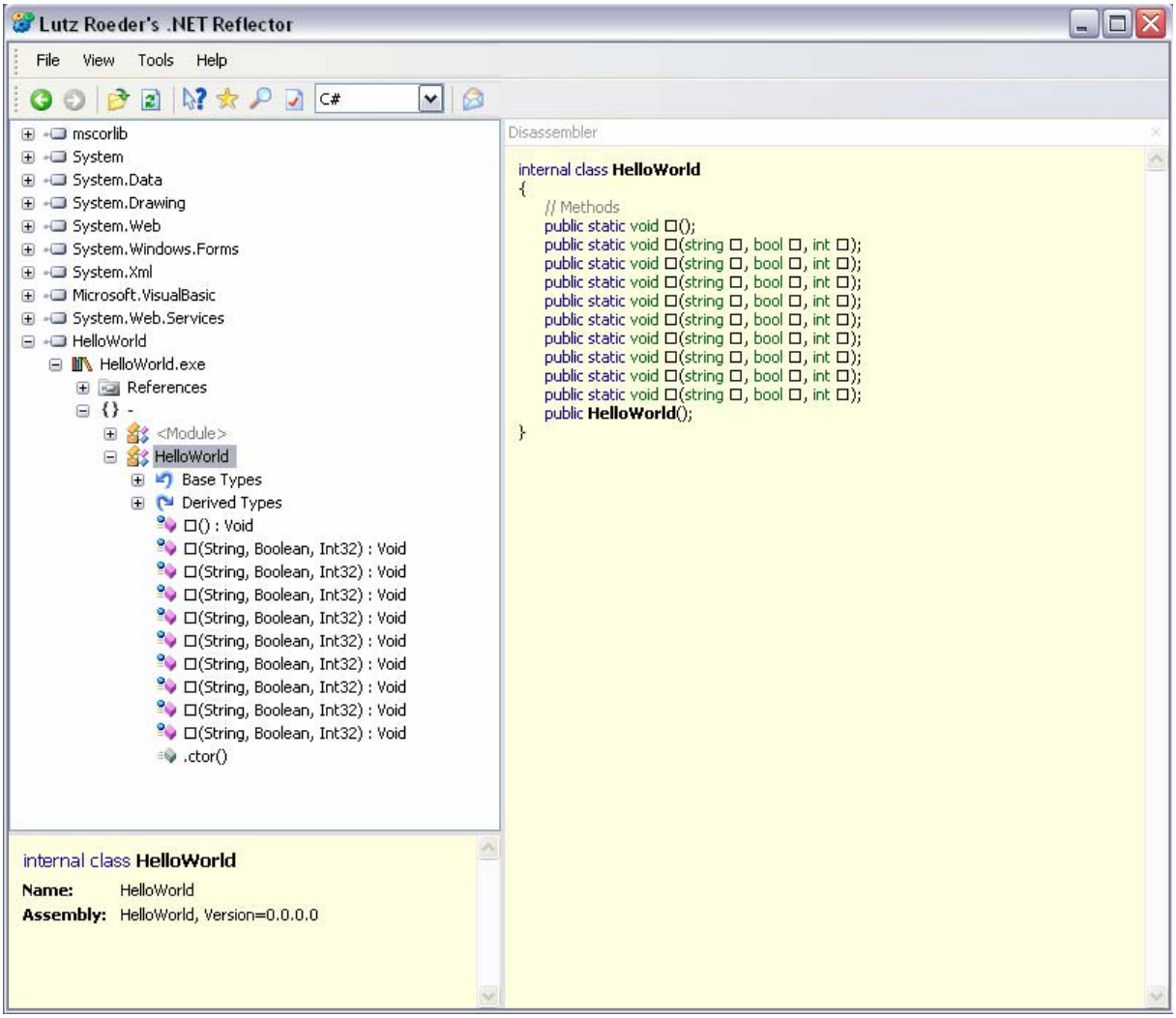
Taigi sukompiliavus HeloWorld.cs gauname HelloWorld.exe, paleidę įsitikiname kad veikia (18 pav.). Atidarome HelloWorld.exe su dekompiliatoriumi (19 pav.) ir aiškiai matome metodų ir kintamųjų vardus: Test1, Test2, Test3 ir t.t.

18 pav. HelloWorld paleistas



19 pav. Originalus HelloWorld

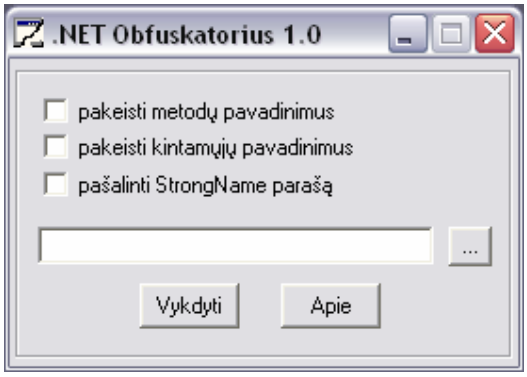
Paleidžiame obfuskatorių pasirenkame failą HellowWorld.exe, uždedame varneles ant pakeisti metodų pavadinimus ir pakeisti kintamųjų pavadinimus ir spaudžiame vykdyti. Programa originalų HellowWorld.exe pervadina į HellowWorld.exe.bak, ir vykdo pakeitimus HellowWorld.exe. Pasileidžiame dekompiliatorių atsidarome obfuskuotą failą (20 pav.) – matome kad visi metodų ir kintamųjų pavadinimai nebesuprantami, obfuskatorius pakeitė jų vardus į baitą 01, tai visiškai nebeatstatoma. Pabandome paleisti HellowWorld.exe ir gauname rezultatą, kaip ir 18 pav., failas obfuskuotas ir veikia.



20 pav. Obfuskuotas HelloWorld

Sistemos naudojimo instrukcija

Norėdami pradėti darbą, spustelkite du kartus kairiu pelės klavišu ant Obfuscator.exe. Programa pasileido ir jūs matote (21 pav.) pagrindinį programos dialogą.



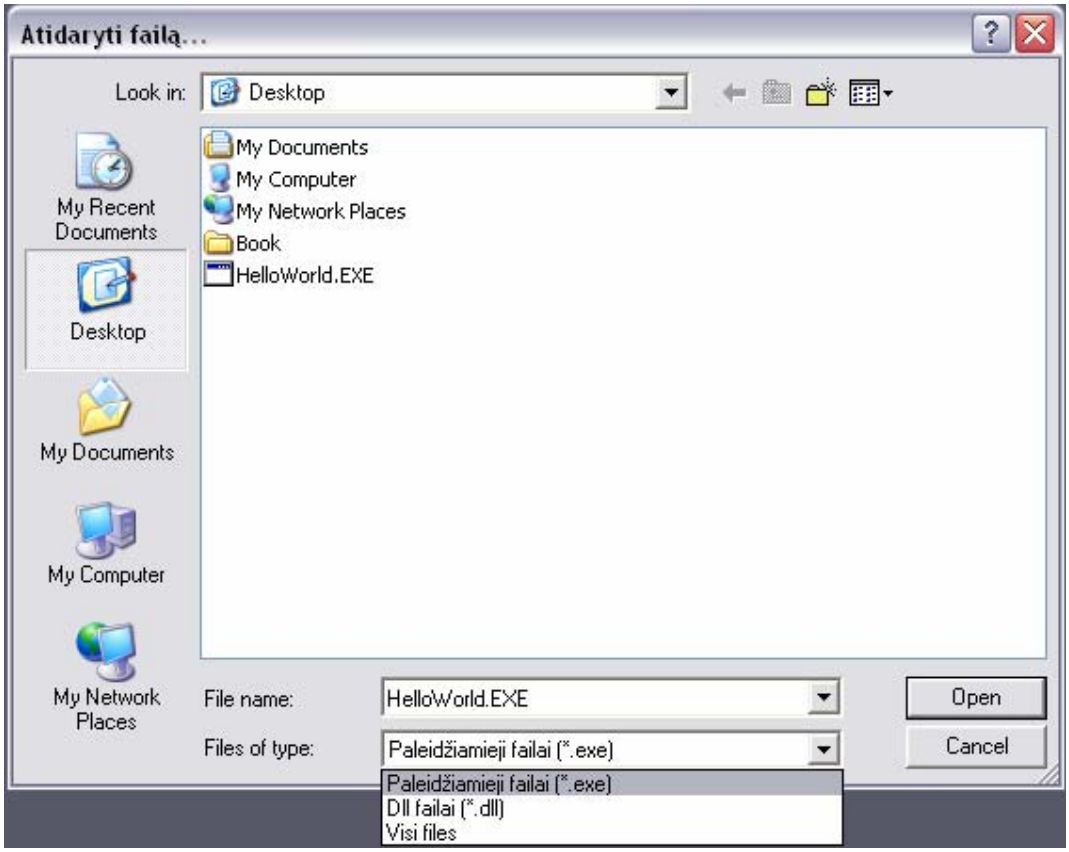
21 pav. Darbo pradžia

Paspauskite mygtuką su daugtaškiu ir jums pasirodys failo pasirinkimo dialogas, prie File Types: galite pasirinkti pagal kokius failus rūšiuoti (22 pav.). Pasirinkite failą kurį norite obfuskuoti ir spauskite Open mygtuką. Kelias į jūsų pasirinktą failą atsiras pagrindinio dialogo (20 pav.) teksto laukelyje.

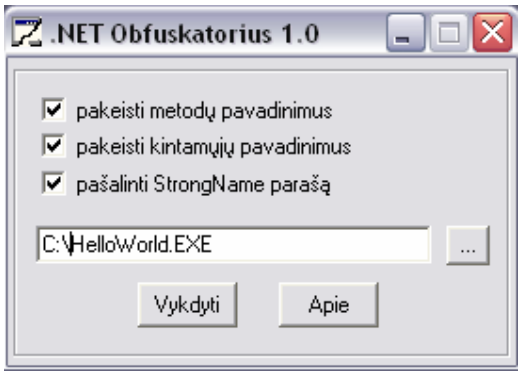
Uždėkite varnelę bent ant vieno iš galimų įvykdyti pakeitimų ir spauskite Vykdyti (22 pav.). Jei failas CLR tipo jūs gausite pranešimą apie sėkmingą pakeitimų įvykdymą (23 pav.), taip pat programa toje pačioje direktorijoje kaip ir jūsų failas sukurs *.bak failą – tai jūsų originalus failas.

Klaidų pranešimai ir jų paaiškinimai pateikti 15 lentelėje.

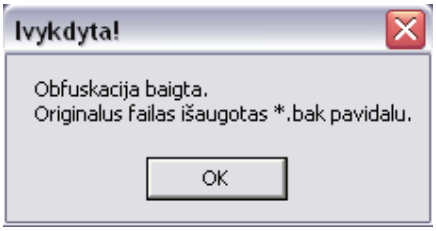
Paspaudę mygtuką Apie išvysite pranešimą apie programos autorių bei programos versiją (25 pav).



22 pav. Failo pasirinkimas



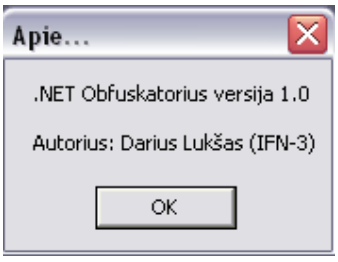
23 pav. Būdų pasirinkimas



24 pav. Sėkmingai įvykdyta

15 lentelė Įmanomi klaidų pranešimai

Klaidos tekstas	Sprendimas/Priežastis
Neįmanoma atidaryti failo.	Reikia patikrinti ar failas nenaudojamas kitų procesų ir ar nėra su READ-ONY atributais.
Trūksta atminties.	Atlaisvinti operatyvinę atmintį, nes failo dydis netelpa į jūsų operatyvinę atmintį.
Nepaleidžiamasis failas.	Įvestas failas nepaleidžiamasis arba neturi jo parašų.
Ne PE failas.	Failas paleidžiamasis bet ne PE formato.
Nesuprantamas PE failas, .text sekcijos neįmanoma rasti.	Failas PE bet neįmanoma rasti .text sekcijos, todėl gali būti arba nežinomas formatas arba failas pažeistas.
Ne CLR failas.	Failas neturi CLR informacijos – todėl greičiausiai tai paprastas PE failas.



25 pav. Apie programą

Techninės įrangos projektas

Minimalūs reikalavimai kompiuterinei techninei įrangai:

- Intel Pentium 100 MHz procesorius;
- 48 MB RAM operatyvinė atmintis;
- 1 Kb laisvos kietojo disko atminties programai saugoti;

Informacinės sistemos diegimas

Informacinė sistema nereikalauja jokių specialių diegimo priemonių – failas tiesiog kopijuojamas į norimą laikmeną ir gali būti paleidžiamas.

3.2 Projekto išvados

Suformuluota sistemos užduotis, kurios tikslas remiantis išanalizuota CLR failo struktūra sukurti obfuskatorių lengvai suprantamą vartotojams. Sistema naudoja metodų, funkcijų, kintamųjų vardų obfuskacijos metodą. Testuojant programą buvo naudojamas .NET aplinkos failų dekompiliatorius. Parašyta sistemos vartotojo instrukcija.

4. EKSPERIMENTINIS TYRIMAS

16 lentelė Sukurtos sistemos kokybės tyrimas

Funkcionalumas	Tinkamumas	visos programos funkcijos skirtos toms užduotims atlikti
	Tikslumas	tiksliai atlieka obfuskavimą ir duoda norimus rezultatus
	Operatyvumas	nepažeidžia jokių teisės aktų ar konvencijų
	Teisėtumas	autorius sutinka su programos laisvu naudojimu
	Saugumas	programa nemokama – todėl į saugumą neatsižvelgta
Patikimumas	Užbaigtumas	programa pilnai užbaigta, atsitiktinių klaidų testavimo metu nepasitaikė
	Klaidų galimybė	klaidos netoleruojamos, nes dirbama su kitomis programomis
	Atstatomumas	daroma originalaus failo kopija
Patogumas	Aiškumas	vartotojo sąsaja lengvai suprantama ir aiški
	Mokymosi galimybė	vartotojas naudojimusi programa gali išmokti labai greitai
	Operatyvumas	operatyviai vykdo instrukcijas ir aiškiai
Efektyvumas	Greitumas	naudojama žemo lygio programavimo kalba todėl programos funkcijos efektyvios ir greitos
	Resursų naudojimo greitumas	išoriniai resursai beveik nenaudojami
Palaikomumas	Analizės galimybė	yra klaidų identifikavimas
	Pakeitimo galimybė	yra funkcijų pakeitimo galimybė
	Stabilumas	dirba stabilia, programos stabilumą gali įtakoti įvestas failas.
	Testavimo galimybė	yra pataisymų galimybė
Suderinamumas	Pritaikomumas	nėra
	Diegimo galimybė	programa kopijuojama
	Atitinkamumas	neprieštarauja jokioms konvencijoms
	Pakeičiamumas	įmanomas pakeitimas kita nauja sistema

Tolimesnės sistemos plėtojimo galimybės – įmanoma įvesti pakompiliavimą į IL kodą ir vykdyti vartotojo teksto (User Strings) bei IL kodo šifravimą.

5. IŠVADOS

Suformuluotos analizės išvados:

1. Palyginti UML ir struktūrinis projektavimas, analizei buvo pasirinktas ir struktūrinis projektavimas nes jis labiausiai atitinka tyrimo objekto poreikius;
2. Išanalizuotas tyrimo objektas, t.y. CLR failas ir meta lentelių sandara, remiantis šia informacija galima buvo tęsti obfuskacijos sistemos kūrimą;
3. Aprašyti ir palyginti įvairuoise šaltiniuose pateikti obfuskacijos metodai. Projektuojamai sistemai kurti pasirinktas (9 lentelėje, nr.1) metodas, kai keičiami CLR failo metodų, funkcijų, kintamųjų vardai;
4. Kuriamos sistemos kokybės kriterijams apibrėžti buvo pasirinktas ISO/IEC 9126 standartas.

Suformuluotos projektinės dalies išvados:

5. Pateiktas kuriamos sistemos projektas, suformuluota sistemos užduotis, jos tikslas remiantis išanalizuota CLR failo struktūra sukurti obfuskatorių lengvai suprantamą vartotojams, kuris naudotų metodų, funkcijų, kintamųjų vardų obfuskacijos metodą;
6. Suformuluoti kuriamos sistemos reikalavimai., kurie per eksperimentinį tyrimą puikiai atitiko analitinėje dalyje užsibrėžtą ISO/IEC 9126 standartą.

6. LITERATŪRA

1. **D. J. Anderson.** User Interface Analysis. Prieiga per Internetą <http://www.uidesign.net/1999/papers/UIA1.html>.

2. **V.Balasubramanian, M.Turoff.** Supporting the User Interface Design Process with Hypertext Functionality. Prieiga per Internetą: <http://eies.njit.edu/~turoff/Papers/uihtf/uihtf.html>.

3. **C. Finkelstein.** An Introduction to Information Engineering From Strategic Planning to Information Systems. *Addison-Wesley Publishing Company,* 1989, 393 p.

4. **A. Granlund, D. Lafreniere.** A Pattern-Suported Approach to the User Interface Design Process. Prieiga per Internetą: <http://www.gespro.com/lafrenid/PSA.pdf>.

5. OMG Unified Modeling Language Specification [interaktyvus]. *Object Management Group, Inc.*1999. Prieiga per Internetą: <http://www.omg.org/technology/documents/index.htm>

6. Remotesoft Salamander .NET Obfuscator. Prieiga per Internetą: < http://www.remotesoft.com/>

7. Prieiga per Internetą: <http://www.dotnetthis.com/Articles/Obfuscation.htm>

8. ISO/IEC 9126 standartas <http://www.issco.unige.ch/ewg95/node13.html>

9. Prieiga per Internetą: < http://www.informit.com/articles/article.asp?p=25350&rl=1>

7. TERMINŲ IR SANTRUMPŲ ŽODYNAS

17 lentelė Terminai

Angliškai	Lietuviškai
console	**komandinė eilutė**
header	**antraštė**
framework	**aplinka**

18 lenlelė Santumpos

Santrumpa	Pilnas pavadinimas	Vertimas
CLR	Common Language Runtime	Bendros kalbos paleidžiamasis
GUI	Graphical User Interface	Grafinė vartotojo sąsaja
PE	Portable Executable	Pernešamas paleidžiamasis
IL	Intermediate Language	Tarpinė žemo lygio kalba
UML	Unified Modeling Language	Unifikuota modeliavimo kalba

8. SUMMARY

Obfuscation and realization methods

The purpose of this project was to analyze and compare Microsoft Framework .NET (CLR type) files and their obfuscation methods, create the obfuscation system, using some of the analyzed methods.

After analyzing different Microsoft Framework .NET obfuscation techniques in this project:

- class, method, parameters name change;
- user string encryption;
- IL code encryption;

Decision was made to use obfuscation method with class, method, parameters name change.

9. PRIEDAI

19 lentelė Kompaktinio disko turinys

Direktorija/Failai	Paaiškinimai
\Sources _const.asm _dataDLG.asm _dataPE.asm ChangeString.asm CheckFileImage.asm GetMetaData.asm Hex2Bin.asm Main.asm Obfuscate.asm ReadInputFile.asm WriteOutputFile.asm	programos išeities kodai
\Program Obfuscator.exe	programos *.exe failas
\Decompiler ReadMe.htm Reflector.exe	Lutz Roeder‘s .NET dekompiliatorius
\TestFile HelloWorld.cs HelloWorld.exe RunView.bat	testinis HelloWorld failas su išeities kodu