KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

TERSİNE MÜHENDİSLİK YÖNTEMLERİNİ KULLANARAK .NET ÇEVRELERİNDE KAYNAK KODA DÖNÜŞÜM VE DOSYA SİSTEMİ İŞLEMLERİNİN KONTROLÜ

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Salih ARAS

TEMMUZ 2008
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

TERSİNE MÜHENDİSLİK YÖNTEMLERİNİ KULLANARAK .NET ÇEVRELERİNDE KAYNAK KODA DÖNÜŞÜM VE DOSYA SİSTEMİ İŞLEMLERİNİN KONTROLÜ

Bilgisayar Mühendisi Salih ARAS

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsünce
"Bilgisayar Yüksek Mühendisi"
Unvanı Verilmesi İçin Kabul Edilen Tezdir.

Tezin Enstitüye Verildiği Tarih : 06.06.2008
Tezin Savunma Tarihi : 01.07.2008

Tez Danışmanı : Yrd. Doç. Dr. Hüseyin PEHLİVAN
Jüri Üyesi : Prof. Dr. Vasif V. NABİYEV
Jüri Üyesi : Yrd. Doç. Dr. Ali GANGAL

Enstitü Müdürü V.: Doç. Dr. Salih TERZİOĞLU

Trabzon 2008

ÖNSÖZ

Son zamanlarda tersine mühendislik çalışmalarının önemi gitgide artmaktadır. İnsanlar kolay yoldan bilgiye erişmek istemektedir. Başkalarının elde ettiği başarıyı daha az sürede nasıl elde edebilirim hesaplarını yapmaktadırlar.

Tersine mühendislik sadece yazılım dünyasında değil, hayatın birçok alanında da karşımıza çıkmaktadır. Örneğin A firması güzel bir makine geliştirdi. Ve bunun için yıllarca uğraştı. Eğer B firması bu makineyı yapmak istiyorsa ya A firmasının geçtiği süreçten geçecek veya makinenin işlevlerini çözerek, benzerini yapacaktır.

Çalışmamda .net ile yazılmış programların kaynak koda dönüştürülmesi ele alınmış ve PE dosya yapısı, CLR yapısı, metadata tabloları, programın hangi sistem kaynaklarını kullandığı, programdaki yapılar incelenmiştir. Daha ayrıntılı dönüşüm için bu tablolar (42 tane metadata tablosu ve import tablosu) çok detaylı bir şekilde incelenmiştir ve tabloların birbirleriyle olan ilişkiler belirlenmiştir. İmaj dosyasındaki dosya işlemlerinin neden yapıldığının analizi gerçekleştirilmiştir ve eğer bir dosya yazılmak için açılmışsa, bu dosyanın yedeklenmesi için gerekli kodlar imaj dosyasına eklenmiştir. Bu şekilde imaj dosyasının yapısı değiştirilerek dosya sistemi üzerindeki işlemlerin etkileri geri alınabilir yapılmıştır.

Bu çalışmamda bana her zaman destek olan çok değerli büyüğüm DSİ 18. Bölge Müdür Yardımcısı Dr. Uğur Şafak ÇAVUŞ'a desteklerinden dolayı teşekkürü bir borç bilirim. Ayrıca bu günlere gelmemde emeği olan KTÜ'deki hocalarıma ve danışmanlığımı üstlenen hocam Yrd. Doc. Dr. Hüseyin PEHLİVAN'a teşekkür ederim. DSİ ailesine ve varlıklarını her zaman yanımda hissettiğim ve varlıklarıyla güç aldığım aileme teşekkür ederim.

Salih ARAS
Trabzon 2008

İÇİNDEKİLER

Sayfa No:

ÖZET

Tersine mühendislik (Reverse Engineering, RE) bir aygıtın, objenin veya sistemin; yapısının, işlevinin ve ya çalışmasının, çıkarımcı bir akıl yürütme analiziyle keşfedilmesi işlemidir. Bu yöntem, genellikle orijinalinden kopyalamadan onunla aynı şeyi yapan yeni bir alet veya yazılım yapmaya çalışır ve sıklıkla bir şeylerin (örneğin; makine veya mekanik alet, elektronik komponent, yazılım programı gibi) parçalarına ayrılması ve çalışma prensiplerinin detaylı şekilde analizini içerir. Tersine mühendislik, sıklıkla, diğer milletlerin teknolojilerini, aletlerini, bilgilerini veya açık arazilerde muvazzaf skerler tarafından toplanan ve ya haberalma operasyonlarıyla toplanan bilgi parçalarını kopyalamak için, ordu tarafından sık sık kullanılır.

Yazılım dünyasında ise tersine mühendisliğin önemi algoritmalarda ve karmaşık sistemlerde ortaya çıkmaktadır. Örneğin, bir problem için güzel bir yöntem geliştirilerek program koduna dönüştürülmüş olabilir. Bu yöntemi elde etmek için genel olarak üç seçenek mevcuttur. Birincisi problemi çözmek ki çok uzun zaman alabilir veya sonuç vermeyebilir. İkincisi programa sürekli veriler vererek çıkışı gözlemlemek ve bu çıkış değerlerine göre problemin çözümü hakkında fikir sahibi olmaktır. Bu yöntem de sonuç vermeyebilir. Üçüncüsü ise programı kaynak koda döküp problemin nasıl çözüldüğünü anlamaktır. İşte tersine mühendisliğin yazılım dünyasında önemi burada ortaya çıkmaktadır.

Anahtar Kelimeler: Tersine Mühendislik, Tersine Derleyici, Hata Ayıklayıcı, Disassembler, Geçiş Dili, PE, COFF, CLR, Metadata Tabloları, İmaj Dosyası, Import Tablosu

SUMMARY

Convert program in writing .net to source code with using reverse engineering methods.

Reverse Engineering (RE) is the process of discovering the technological principles of a device, object or system through analysis of its structure, function and operation. It often involves taking something (e.g. a mechanical device, electronic component, or software program) apart and analyzing its workings in detail, usually to try to make a new device or program that does the same thing without copying anything from the original. Reverse engineering is often used by military in order to copy other nations' technology, devices or information, or parts of which, have been obtained by regular troops in the fields or by intelligence operations.

In software world, the importance of reverse enginneering occurs in algorihtms and complex systems. For example, suppose that a good method is discovered and programmed for a particular problem. In general, there are three alternative ways to acquire the method. One way is to try to solve the problem, which can take long time or can't be solved. The second one is to give various data to program and to observe the output. This output may give an idea in identifying the method. But it also results in no solution all the time. This method also can't solve the problem. The last one is to understand how the problem is solved converting the program to source code. That is why reverse enginnering is very important in the software world.

Key Words: Reverse Engineering, Decompiler, Debugger, Disassembler, IL, PE, COFF, CLR, Metadata Tables, Image File, Import Table

ŞEKİLLER DİZİNİ

Sayfa No:

TABLOLAR DİZİNİ

Sayfa No:

SEMBOLLER DİZİNİ

RE	Tersine Mühendislik (Reverse Engineering)
CLR	Ortak Dil Çalışma Zamanı (Common Language Runtime)
IL	Ara Dil (Intermediate Language)
PE	Portatif İcra Edilebilir (Portable Executable)
COFF	Ortak Obje Dosya Biçimi (Common Object File Format)
DLL	Dinamik Link Kütüphanesi (Dynamic Link Library)
RVA	Bağıl Sanal Adres (Relative Virtual Adresses)
VA	Sanal Adres (Virtual Address)

1. GENEL BİLGİLER

1.1. Tersine Mühendislik

1.1.1. Tersine Mühendislik Nedir

Tersine mühendislik (Reverse Engineering, RE) bir aygıtın, objenin veya sistemin; yapısının, işlevinin veya çalışmasının, çıkarımcı bir akıl yürütme analiziyle keşfedilmesi işlemidir. Bu yöntem, genellikle orijinalinden kopyalamadan onunla aynı şeyi yapan yeni bir alet veya yazılım yapmaya çalışır ve sıklıkla bir şeylerin (örneğin; makine veya mekanik alet, elektronik komponent, yazılım programı gibi) parçalarına ayrılması ve çalışma prensiplerinin detaylı şekilde analizini içerir.

Günümüzde müşteriler daha çok kişisel ürün istemektedirler. Müşteri isteklerindeki bu çeşitlilik işletmeleri değişik pazarlama stratejileri uygulamak zorunluluğunda bırakmaktadır. Bu işletmeler, müşteri isteklerini karşılayabilmek amacıyla geniş bir ürün çeşitliliğiyle pazara hakim olmanın yanı sıra pazara sürekli yeni ürünler sunmaktadırlar. Sunulan bu ürünlerin kaliteli olması ve pazardaki yerlerini en kısa zamanda alması ise işletmelere rekabet açısından büyük avantajlar sağlamaktadır. Bu koşullar altında varlıklarını sürdürmeye çalışan işletmeler, pazara küçük partiler halinde, özelleştirilmiş, çok kaliteli ürünleri düşük maliyetler ile sunmayı hedeflemektedir. Bu hedefi gerçekleştirmek kolay olmadığı gibi, bu iş için işletmelerin kitlesel üretim ve yalın üretimden çok daha güçlü olan çevik, tepkisel ve bileşik üretim/yönetim felsefelerine ihtiyaçları vardır. Bu yüzden son zamanlarda üretim dünyasında, müşteri isteklerine ve önceden kestirilemeyen pazar değişikliklerine çok çabuk uyum sağlayabilecek; çevik, tepkisel ve esnek üretim ve yönetim stratejileri, yöntemleri ve paradigmaları öne çıkmış bulunmaktadır. Tasarımdan üretime ve üretimden de pazarlamaya değin akıp giden tüm süreçlerin her zaman başlangıç noktası olması nedeniyle, "ürün tasarımı ve geliştirilmesi" alt süreci performansının tüm bu modern yöntemlerin başarılarında en büyük rolü oynadığı anlaşılmış bulunmaktadır. Ürün geliştirme zamanının azaltılması; esnekliğin, çabukluğun, çevikliğin ve tepkiselliğin bir ön şartı durumuna gelmiştir.

Tersine Mühendislik yaklaşımı, ürün geliştirme zamanının azaltılması için işletmelere mükemmel bileşik mühendislik fırsatları sunar. Tesine mühendisliğin temel uygulamaları şu şekilde sıralanabilir;

- Yeni bir parçanın tasarımı,
- Var olan bir parçanın kopyalanması,
- Yıpranmış veya hasar görmüş parçaların kurtarılması, düzeltilmesi ve yeniden tasarlanması
- Model hassasiyetinin ve doğruluğunun geliştirilmesi,
- Numerik modellerin denetlenmesi.

Kavramsal tasarım ile başlayan geleneksel mühendislik sürecinin aksine, Tersine Mühendislik sürecinde ürün tasarımına, gerçekte var olan bir modelin şekil bilgisinin elde edilmesi ile başlanır. Serbest ve karmaşık yüzeylere sahip olan gerçek parçaların geometrik bilgisinin elde edilmesi tesine mühendisliğin en önemli aşamalarından biridir. Yeniden yapılandırılacak parça modelinin kalitesi, başlangıç modelinin üzerine ölçülen noktaların sayısına, ölçüm tipine ve doğruluğuna, ve ölçüm tekniğine (cihazın cinsi vb) bağlı olarak değişebilir.

1.1.2. Tersine Mühendisliğin Türleri ve Uygulamaları

"Tersine Mühendislik" terimi kolaylıkla anlaşılan anlamlara sahip değildir ve çoğu kez karıştırılır. Örneğin, donanım tersine mühendisliği, bilgisayar parçalarının de-montaj yapılarak nasıl çalıştığını öğrenilmesi ve aynılarının yapılması amacıyla kullanılmaktadır. Yazılım tersine mühendisliği ise, bir programın kodlarını çözmek ve programın bazı kısımlarını kopyalamak, programın lisans kodlarını kırmak gibi yasal olmayan amaçlarla da kullanılmaktadır. Bu işlemler pek çok ülkede olduğu gibi, ülkemizde de yasal değildir ve "fikir ve sanat eserlerinin korunması" ile ilgili kanunlar uygulamadaki sorunlarıyla birlikte yürürlüktedir.

"Tersine Mühendislik" bir makineyi veya nesneyi, kopyalamak veya geliştirmek amacıyla veya çalışma prensibini belirlemek amacıyla parçalara ayırmak olarak da tarif edilmektedir. Bu tarif, özde yanlış olmamakla birlikte eksiktir. Örneğin otomobil endüstrisindeki bir firmanın, rakip firmanın otomobilini alıp bunu parçalara ayırması, daha sonra her bir parçayı inceleyip test ederek, kendi otomobilini geliştirmek için bu parçalardan faydalanması tersine mühendisliktir ve yasal olabilir. Ancak, parçaların aynı

prensip ve yöntemler kullanılarak taklit edilmesi etik olmadığı gibi, eğer rakip firma tarafından patent ile korunmuş ise hırsızlıkla eş değerdir. Bu nedenle, neden-sonuç ilişkisinin çok iyi kurulması gereklidir.

Vaktiyle ülkemizde takım tezgahları üreten bir fabrika, Uzakdoğu'dan getirdiği bir bilgisayar kontrollü (CNC) torna tezgahını en küçük parçasına kadar sökmüş ve taklit etmeye çalışmıştı. Ancak sonuç hüsran oldu. Taklit etmeye çalıştıkları tezgahı geliştirmek şöyle dursun, taklit bile edememişler ve iflasın eşiğine doğru sürüklenmişlerdi. Bu sonuç, bu yaklaşımın tek başına yeterli olmadığını, modern teknoloji ve bütünleşik imalat felsefesi olmadan başarıya ulaşılamayacağının bir örneği olarak tarih sayfalarındaki yerini almıştır. Siz bir makinenin, tenekelerinin ve dişlilerinin aynısını yapabilirsiniz, ama Murat 124 şasine Mercedes motoru koyamazsınız. Oysa CNC tezgahların mekanik aksamın dışında bir de kontrol üniteleri vardır. Bedenlerin yanında bir de ruhlar vardır. O ruhu veremezseniz, beden hareketsiz bir kütleden ibaret kalır.

Bizim asıl üzerinde durduğumuz "Tersine Mühendislik", var olan bir nesnenin tasarım bilgilerinin bulunmadığı durumlarda, nesneyi yeniden üretebilmek veya geliştirebilmek amacıyla, ürünün üç boyutlu uzayda sayısal tasarım bilgilerinin elde edilmesidir. Bu yönüyle, Tersine Mühendislik uygulamalarının en önemli elemanları şunlardır,

- Sayısallaştırıcı/ tarayıcılar
- Otoinşa (Hızlı prototipleme) makineleri
- Tersine mühendislik yazılımları

Tersine mühendislik, sıklıkla, diğer milletlerin teknolojilerini, aletlerini, bilgilerini ve ya sahada sıradan askerler tarafından toplanan ve ya haberalma operasyonlarıyla toplanan bilgi parçalarını kopyalamak için, ordu tarafından sık sık kullanılır. İkinci Dünya Savaşı'nda ve Soğuk Savaş'ta sıkça kullanılmıştır. İkinci dünya savaşı'ndan çok bilinen örnekler şunlardır:

- Jerry bidonu:İngiliz ve Amerikan kuvvetleri Almanların Jerry can denen ve mükemmel bir tasarıma sahip olan benzin bidonlarına sahip olduklarını fark ettiler. Bu bidonların kopyaları üzerine ters mühendislik uyguladılar. Bu bidonlar popüler olarak "Jerry can" olarak bilinirler.
- Tupolev Tu-4: Japonya görevi sırasında bir çok B-29 Superfortress bombardıman uçağı Sovyetler Birliği'ne (SSCB) inmek zorundaydı. Böyle stratejik bir bombardıman

uçağına sahip olmayan Sovyetler B-29'u kopyalamaya karar verdiler. Birkaç yıl içinde neredeyse mükemmel bir kopyası olan Tu-4'ü geliştirdiler.

• V2 Roketi: V2 ve bağlantılı teknolojilerin teknik dökümanları savaşın sonunda batılı müttefikler tarafından elde edildi. Sovyet ve yakalanmış Alman mühendisler, daha sonra R-7 Semyorka'nın öncüsü ve uzay programının başlangıcı olan, kendi klon roketleri R-1 i yapmak için ele geçirilen dökümanlardan yola çıkarak, yeni teknik dökümanlar ve planlar oluşturdular.

Birlikte işlerlik amacıyla (örneğin; belgesiz dosya formatlarını ve ya donanım ekipmanlarını desteklemek için) yapılmış tersine mühendislik donanım veya yazılımlarının, patent sahipleri çoğunlukla buna karşı çıksa da ve kendi ürünlerinin herhangi bir sebeple ters mühendisliğe uğramasını engellemeye çalışsa da, legal olduğuna inanılır.

Benzer şekilde, yazılım mühendisliğindeki kara kutu testinin ters mühendislikle pek çok ortak noktası vardır. Testi yapan çoğunlukla uygulama programlama arayüzüne sahiptir, fakat amaçları, ürüne dışarıdan saldırarak, hataları ve belgelendirilmemiş özellikleri bulmaktır.

Güvenlik denetimi, kopya korumanın kaldırılması(yazılımı kırma, krekleme), tüketici elektronik eşyalarında bulunan erişim önlemesini giderme, gömülü sistemlerin (motor yönetim sistemleri gibi) uyarlanmasında, ev içi tamirat ve uyarlamalarda, düşük maliyetli arızalı donanımlarda(bazı grafik kartları çipleri gibi) ek özelliklerin sağlanmasın da veya sadece merak gidermek gibi amaçlar, tersine mühendisliğin diğer amaçlarındandır.

Tersine mühendislik aynı zamanda, işletmeler tarafından, kendi ürünlerinin üç boyutlu dijital kayıtlarını yapmak veya rakiplerinin ürünlerine değer biçmek için, var olan fiziksel geometriyi dijital ürün geliştirme ortamlarına aktarmak için kullanılır. Örneğin; bir ürünün nasıl çalıştığını, ne yaptığını hangi bileşenlerden oluştuğunu; analiz etmek için, maliyetini hesaplamak için ve potansiyel patent ihlalini saptamakta kullanılır.

Değer mühendisliği işletmeler tarafından kullanılan benzer bir aktivitedir. Yapı çözümlemeyi ve ürünü analiz etmeyi içerir ancak asıl amacı maliyeti azaltmak için fırsatlar bulmaktır.

Son olarak, tersine mühendislik sıklıkla özel bir aletin dokümanlarının kaybolması (veya hiç yazılmaması) ve onu inşa eden kişinin artık şirket için çalışmaması sebebiyle yapılır. Entegre devreler çoğunlukla modası geçmiş, tescilli sistemler üzerine dizayn

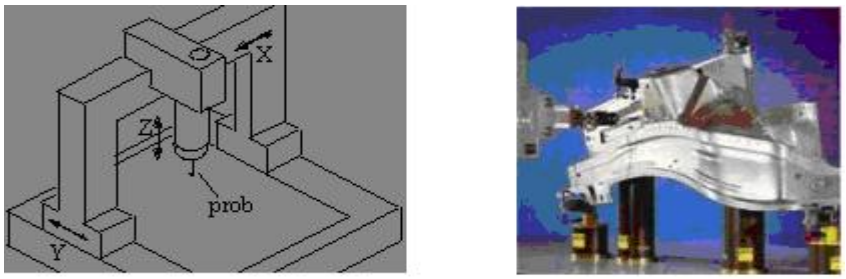
edilir, bu şu anlama gelir; işlevselliği yeni teknoloji ile birleştirmenin tek yolu, var olan çip üzerinde tersine mühendislik yapmak ve onu yeniden dizayn etmektir.

1.1.3. Sayısallaştırma ve Tersine Mühendislik

Nesnelerin üç boyutlu ölçümleri kalite kontrol uygulamaları için vazgeçilmez bir unsurdur. Parça üzerindeki unsurların paralelliği, dikliği ve boyutsal toleranslarının doğruluğunun kontrol edilmesi bu uygulamalar içerisinde yer alır. Bununla birlikte, bu uygulamalar genellikle geleneksel üretim sürecinin bir parçası olarak ortaya çıkar. Tersine Mühendislik ise bunun bir adım ötesidir. Aynı cihazlar üzerinde, sadece ölçüm değil, tarama ve sayısallaştırma da yapılabilir. Tersine Mühendislik'te ölçüm ve sayısallaştırma/tarama uygulamaları içerisine kullanılan cihazları (koordinat ölçme makineleri, sayısallaştırıcı/tarayıcılar vb.) iki ana grupta toplamak mümkündür;

- Temas ederek (Problu) ölçüm ve sayısallaştırma/tarama yapan cihazlar
- Temas etmeden ölçüm ve sayısallaştırma/tarama yapan cihazlar
- Lazerli
- Kameralı (Topometrik Görüş) sistemleri

Problu ölçüm cihazlarında, ölçüm kolunun üzerinde elmas sertliğinde bir küre mevcuttur. Bu küre parçanın yüzeyinde, koordinatları belirlenmesi istenen noktaya değdiği anda, kolun üzerindeki koordinat belirleyici sistemi ile, parçanın o noktadaki konumu, iş parçasının geometrik ve boyutsal verileri üç boyutlu uzayda (x, y, z) elde edilmekte ve cihazın üzerinde bulunan bilgisayara aktarılır. Problu sistemin dezavantajı, ölçüm alınabilmesi için probun yüzeye değme zorunluluğunun olmasıdır. Bu zorunluluk parçanın karmaşık şekilli olması durumunda, istenen değerlerin alınamaması sonucunu doğurabilir.



Şekil 1. Proplu Sistem

Lazerli sistemlerde, ölçüm/sayısallaştırma/tarama bir lazer hüzmesi kullanılarak gerçekleştirilir. Parçanın ölçüm yapılmak istenen bölgelerine yollanan lazer ışını, kaynaktan gidiş ve dönüş zamanının, ışının hızıyla çarpılması sonucu otomatik olarak hesaplanır. Koordinatlar yine kolun üzerindeki bir adım koordinat belirleyici sayesinde alınır. Lazerin doğrusal hareket ettiği dikkate alındığında düz-yüzey tabir edilen yumuşak yüzeyli (Arabaların kaportaları vb.) yüzeyler için oldukça idealdir. Ancak, karmaşık parçalar için, önerilen bir sistem değildir. Bunun nedeni ise lazer ışınının geri dönmesini söz konusu olamayacağı karmaşık şekilli ve içsel (delik içerinde) unsurları bulunan nesnelerin katı modelinin oluşturulması ya da ölçümlerinin yapılmasında neden olduğu zorluktur. Bu sistemde veri toplanması, ilerleyen bir lazer ışınının, kusursuz üçgen tekniği olarak adlandırılan bir yöntem ile geri dönmesi sayesinde sağlanabilir.

Topometrik (Kameralı) ölçüm/sayısallaştırma/tarama sistemlerinde, bir üç-ayağın üzerine sabitlenmiş olan ölçüm/sayısallaştırma/tarama kafası, hedef parçanın yaklaşık 70-100 cm kadar ön tarafında tutulur. Ölçüm/sayısallaştırma/tarama sırasında parçanın yüzeyine kenar oluşumlarının izdüşümlerinin yansıması sağlanır ve bu izdüşümler, ölçüm kafası içerisine sabitlenmiş olan bir kamera tarafından kaydedilir. Dijital görüntü işlemcisinin yardımıyla üç boyutlu koordinatlar yüksek bir hassasiyetle hesaplanır. Nesnenin tamamının sayısallaştırılması/taranması, birçok ayrı ölçümlerin bir araya getirilmesi ile oluşur ve bazen birden fazla görüş açısı veya bir başka deyişle kamera kullanılması gerekebilir. Günümüzde, computer-vision yazılım ve donanım teknolojisinin gelişimi zor (free form veya sculptred surfaces) yüzey ve unsurlara sahip nesnelerin modellerinin oluşturulmasını mümkün kılmaktadır.

Şekil 2. Kameralı Sistem

İş parçasına temas etmeden çalışan algılayıcılarla ölçüm/sayısallaştırma/tarama işlemi uzaktan çok kısa bir sürede tamamlanabildiği halde, mekanik problar gibi iş parçasına temas eden algılayıcılar kullanıldığında işleme çevrimi durdurulup pozisyonlama yapılması gerektiğinden, ihmal edilemeyecek bir zaman kaybına neden olunmaktadır. Fiyat bakımından incelendiğinde, iş parçasına temas etmeyen algılayıcıların diğerlerine göre oldukça ucuz olduğu görülecektir.

Temaslı/temassız sistemlerin hepsi de temelde aynı prensiple çalışırlar. Hedef bir "Nokta Bulutu" elde etmektir. Daha sonra bu nokta bulutu uygun yazılımlar ile birlikte anlamlandırılır, uygun yüzeyler türetilir ve Bilgisayar Destekli Tasarım ve İmalat (BDT/BDÜ) süreçlerinde kullanılabilecek uygun bir formata dönüştürülür. Böylelikle nesnenin model verileri bilgisayar üzerine aktarılmış olur. Elde edilen yüzey veya katı model üzerinde istenilen değişiklik veya geliştirmeler yapılabilir. Model son halini aldıktan sonra, modelin üretimi için gerekli takım yolları ve CNC parça programı elde edilebilir. Ancak, bu son işlemden önce bilgisayar üzerindeki modellerin "OtoİnŞa" (Hızlı Prototipleme) makineleri ile ön-gerçek modellerinin oluşturulması önerilir.

1.1.4. Tersine Mühendislik Araçları

1.1.4.1. Hata Ayıklayıcı (Debugger)

Hata ayıklayıcı diğer yazılımlara bağlanarak hata denetimi yapmayı sağlayan ve onları kontrol etmek için kullanılan bir yazılım programıdır. Bir hata ayıklayıcı kodun tek adımına, hata takibine, breakpoint ayarlamasına ve adım adım icra edilen tarzda bir hedef programın değişkenlerinin ve bellek durumunun gözetlenmesine yardım eder. Hata ayıklayıcı mantıksal program akışının belirlenmesi içinde çok değerlidir.

Hata ayıklayıcılar iki kategoriye ayrılır; kullanıcı modlu ve çekirdek modlu hata ayıklayıcılar. Kullanıcı modlu hata ayıklayıcılar işletim sisteminin kontrolü altında normal programlar gibi koşar ve onlarla aynı kurallara bağlıdır. Böylece kullanıcı modlu hata ayıklayıcılar sadece diğer kullanıcı seviyeli süreçlerin hata ayıklamasını yapabilir. Çekirdek modlu hata ayıklayıcılar işletim sisteminin bir parçasıdır ve aygıt sürücülerinin hata ayıklamasını hatta işletim sisteminin kendisinin hata ayıklamasını gerçekleştirebilir.

1.1.4.2. Hata Enjeksiyon Araçları

Bu araçlar şekil bozukluğu verebilir. Hedef yazılımda hangi tür hatalar olabileceğine karar vermek için program eksiklikleri analiz edilebilir. Bazı hatalar saldırganın bilgisayara veya ağa direk erişimi gibi güvenlik açıkları bulaştırır.

Hata enjeksiyon araçları 2 kategoriye ayrılır. Sunucu veya ağ tabalı. Sunucu tabanlı hata enjektörleri hata ayıklayıcılar gibi çalışır ve sürece iliştirilebilir ve program durumunu değiştirir. Ağ tabanlı hata enjektörleri ağ trafiğini kendi doğrultusunda yönetir ve alıcıdaki etkiye karar verir.

1.1.4.3. Disassembler

Disassembler makine kodunu assembly koduna dönüştüren araçlardır. Assembly kodu makine kodunun insan tarafından okunabilir biçimidir. Disassembler kodda hangi makine kodlarının kullanılmış olduğunu açığa çıkartır. Genellikle makine kodu verilen donanım mimarisine özeldir; örneğin “PowerPC Chip” veya “Intel Pentium Chip” gibi. Bu nedenle disassembler doğrudan hedef donanım mimarisi için yazılır.

1.1.4.4. Tersine Derleyici veya Decompiler

Decompiler assembly veya makine kodunu C gibi yüksek seviyeli dillerin kaynak koduna dönüştürür. Ayrıca decompiler Java byte kodu, “Microsoft Common Runtime Language” gibi ara dilleri Java gibi dillerin kaynak koduna çevirir. Bu araçlar döngüler, if-then durumları, switchler gibi yüksek seviyeli mantığın düşünülmesinde oldukça yararlıdır. Decompilerlar disassemblerlara çok benzer fakat süreçleri bir adım ileriden alır.

1.1.5. Tersine Mühendisliğe Yaklaşım

Tersine mühendislik için 3 yaklaşım vardır; beyaz kutu analizi, siyah kutu analizi ve gri kutu analizi.

1.1.5.1. Beyaz Kutu Analizi

Beyaz kutu analizi kaynak kodun analizini ve anlaşılmasını içerir. Bazen sadece ikili kod kullanılabilirdir; eğer ikili kod decompile edilerek kaynak koda dönüştürülür ve sonra incelenirse beyaz kutu analizi göz önüne alınmalıdır. Beyaz kutu analizi programlama ve uygulama hatalarının belirlenmesinde çok etkilidir.

Beyaz kutu analiz araçlarının iki türü vardır. Bazıları kaynak koda gereksinim duyar. Bazıları da ikili kodu otomatik decompile ederek analiz işlemlerini sürdürür.

1.1.5.2. Siyah Kutu Analizi

Siyah kutu analizleri değişik giriş değerlerine göre koşan bir programın incelenmesiyle ilgilenir. Bu test türü sadece koşan bir programa ihtiyaç duyar, herhangi bir tür kaynak kodu kullanmaz. Güvenlik testinde bir programın kırılabilirliğini sınamak amacıyla programa kötü girdi verisi uygulanabilir. Eğer testin bir bölümünde program kırılırsa bir güvenlik problemi keşfedilmiş olur.

Siyah kutu analizinde ağ üzerinde çalışan programlarında test edilmesi mümkündür. Siyah kutu analizlerinin tümü herhangi bir yerde koşan ve giriş isteyen bir programa gereksinim duyarlar.

Siyah kutu analizi beyaz kutu analizi kadar etkili değildir. Fakat daha az tecrübe ister.

1.1.5.3. Gri Kutu Analizi

Gri kutu analizi beyaz kutu analiziyle siyah kutu giriş testinin kombinasyonundan oluşur. Gri kutu analizi genellikle değişik araçların birlikte kullanılmasına gereksinim duyar. Gri kutu analizine en basit örnek olarak bir debugger programının içinde koşan bir hedef programa değişik girişler verilerek gözlemlenmesidir. Bu yöntemde debugger bir hata bulana kadar program koşmaya ve incelenmeye devam eder.

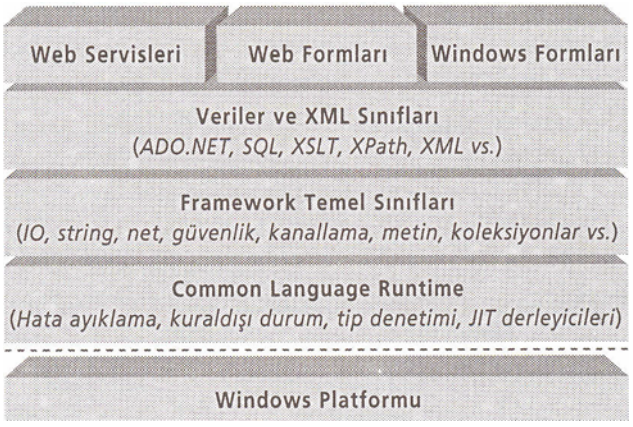
Tablo 1 Siyah Kutu – Beyaz Kutu analizlerinin karşılaştırılması

Siyah Kutu	Beyaz Kutu
Yazılım çalışma zamanı ortamının denetlenmesi • Dış Threadler • Servis Yoksayma • Taşma Hataları • Güvenlik Politikaları ve filtreler • Girişken ağın içinde koşmalar ve ölçeklemeler • Güvenlik ve Sistem Yöneticisi için değerlidir	Yazılım kodunun denetlenmesi • Programlama Hataları • Merkezi Kod Deposu gereklidir • Geliştiriciler ve testçiler için önemlidir.

1.2. Framework Yapısı

1.2.1. Genel Bilgiler

Şekil 3'te görebildiğiniz gibi, .NET Framework farklı Windows sürümleri olabilen ve bazı bileşenler içerebilen (bu bileşenlerden CLR'nin detaylı yapısı ilerleyen bölümlerde ayrıntılı olarak anlatılacaktır) işletim sisteminin en üstünde bulunmaktadır. .NET aslında Windows üzerinde çalışan bir sistem uygulamasıdır.



Şekil 3. .Net Framework yapısı

Bu çatının en önemli bileşeni CLR'dır. Bir Java programcısıysanız, CLR'ı Java Virtual Machine'nin (JVM) eşdeğeri olarak düşünün. Java bilmiyorsanız, CLR'ı .NET mimarisinin kalbi ve ruhu olarak düşünün. Daha üst bir seviyede, CLR nesneleri etkinleştirmekte, onların üzerinde güvenlik kontrolleri gerçekleştirmekte, onları belleğe yerleştirmekte, çalıştırmakta ve onların artıklarını toplamaktadır.

Kavramsal olarak CLR ve JVM, altta yatan platform farklılıklarını soyutlayan çalışma zamanı altyapıları olmaları bakımından benzerdirler. Bununla birlikte, JVM resmi olarak yalnızca Java dilini desteklerken, CLR yalnızca Common Intermediate Language'de (CIL) gösterilebilen tüm dilleri desteklemektedir. JVM bytecode'larını çalıştırır; böylece temel olarak pek çok dili de destekleyebilir. Bununla beraber, Java bytecodu'unun aksine, CIL asla yorumlanamaz. İki farklı altyapı arasındaki bir başka kavramsal fark da, Java bir JVM ile herhangi bir platformda çalışabilirken, .NET kodunun yalnızca CLR'ı destekleyen platformlarda çalışmasıdır. 2003 yılının Nisan ayında, International Organization for Standardization ve International Electrotechnical Committee CLR'ın Common Language Interface (Ortak Dil Arabirimi) adlı işlevsel altkümesini uluslararası bir standart olarak tanıdı. Microsoft tarafından başlatılan ve EC-MA International tarafından geliştirilen bu ilerleme, CLR'ın Linux, Mac OS X gibi platformlarda farklı firmaların kendi sürümlerini uygulamalarının yolunu açtı.

Şekil 1'de CLR'ın üstündeki katman, framework temel sınıfları kümesidir. Bu sınıflar kümesi STL, MFC, ATL ya da Java'da bulunan sınıflar kümesine benzemektedir. Bu sınıflar, diğer işlevlerin yanında, temel giriş ve çıkış işlevselliğini, string işlemeyi, güvenlik yönetimini, ağ iletişimlerini, kanal yönetimini, metin yönetimini, yansıma işlevselliğini ve kolleksiyonlar işlevselliğini desteklemektedir.

Framework temel sınıflarının üstünde, temel sınıflarını, veri yönetimi ve XML işlemeyi destekleyecek şekilde genişleten bir sınıflar kümesi yer alır. ADO.NET adındaki bu sınıflar, kalıcı veri yönetimini desteklemektedir (arka uç veritabanlarında depolanan veriler). Veri sınıflarının yanında, .NET Framework XML verilerini işlemenize ve XML aramaları ve XML çevrimleri yapmanıza olanak veren bazı sınıfları desteklemektedir.

Üç farklı teknolojideki sınıflar (Web servisleri, Web Formları ve Windows Formları) framework temel sınıflarını ve XML sınıflarını genişletmektedir. Web servisleri, firewall'ların önünde ya da NAT yazılımında bile çalışabilen basit dağıtık bileşenler geliştirilmesini destekleyen bazı sınıflar içermektedir. Web servisleri standart

HTTP ve SOAP'ı kullandığından dolayı, bu bileşenler Internet üzerinde tak-ve-çalıştırı desteklemektedir.

1.2.2. Metadata

Metadata bir kaynağa dair, makinenin okuyabileceği bilgidir veya "veriyi tanımlayan veri"dir. Böyle bir bilgi; bir veri kaynağının içerik, format, boyut ya da diğer özellikleri hakkındaki ayrıntıları içerebilir. .NET'te metadata, tip tanımlarını, sürüm bilgilerini, harici assembly başvurularını ve diğer standartlandırılmış bilgileri içerir.

İki sistem, bileşen ya da nesnenin diğeriyle birlikte çalışabilmesi için, en az bir tanesi diğeri hakkında bir şeyler bilmelidir. COM'da, bu "bir şeyler", bir bileşen sağlayıcısı tarafından çalıştırılan ve tüketiciler tarafından kullanılan bir arabirim spesifikasyonudur. Bu arabirim spesifikasyonu tüm parametreler ve geri dönüş tipleri için tip tanımlarını kapsayan tam imzalı metot prototiplerini içermektedir.

Yalnizca C/C++ geliştiricileri kolayca Arabirim Tanımlama Dili (Interface Definition Language, IDL) tip tanımlarını değiştirebilir ya da kullanabilir. Bunu VB geliştiricileri ya da diğer geliştiriciler yapamazlar ve daha da önemlisi diğer araçlar ya da middleware için de yapamaz. Bu yüzden Microsoft herkesin kullanabileceği IDL'den farklı olarak, tip kütüphanesi adında bir şey icat etmiştir. COM'da tip kütüphaneleri bir geliştirme ortamı ya da aracının hedef geliştirici için en uygun ve kullanışlı wrapper (uyumlulaştırıcı) sınıfları okumasına ve yaratmasına imkân verir. Tip kütüphaneleri ayrıca VB, COM, MTS ya da COM+ gibi çalışma zamanı motorlarının tipleri çalışma zamanında incelemelerine ve onları kullanacak uygulamalar için gerekli tesisat ya da ara desteği sunmalarını sağlamaktadır. Örneğin, tip kütüphaneleri dinamik çağrım desteklemekte ve çapraz içerik çağrım için COM çalışma zamanının universal marshaling sağlamasına olanak vermektedir.

1.2.3. Assembly'ler ve Manifestolar

Gördüğümüz gibi, tipler, araçların ve programların kendilerine erişmelerine, servislerinden faydalanmalarına olanak vermek için metadatalarını kullanmak zorundadırlar. Tiplerin metadataları tek başlarına yeterli değildir. Yazılım tak-ve-çalıştırını, yapılandırmasını ya da bileşen ya da yazılım kurulumunu basitleştirrnek için,

tipleri barındıran metadatalara da ihtiyacımız vardır. Şimdi .NET assembly'lerinden (dağıtılabilir birimler) ve manifestolardan (assembly'leri tanımlayan metadatalar) bahsedeceğiz.

Bir assembly CLR'ın çalışma zamanında çalıştırdığı IL kodunu içermektedir. IL kodu genel olarak aynı assembly'nin içinde tanımlanmış tipleri kullanırlar, ama diğer assembly'lerdeki tipleri de kullanabilir ya da onlara başvuru yapabilir. Birincisinden faydalanmak için özel bir şey gerekmezken, assembly ikincisini yapmak için diğer assembly'lere başvurular tanımlamak zorundadır. Bir uyarımız vardır: her bir assembly DllMain ( ), WinMain ( ) ya da Main ( ) gibi en çok bir giriş noktasına sahip olabilir. Bu kurala uymak zorundasınız, çünkü CLR bir assembly yüklediğinde, assembly çalışmasını başlatmak için bu giriş noktalarından birini araştırır.

1.2.3.1. Sürümlendirme

.NET'te dört çeşit assembly vardır:

Durağan assembly'ler: Bunlar derleme zamanında oluşturabileceğiniz .NET PE dosyalarıdır. En sevdiğiniz derleyicinizi kullanarak durağan assembly'ler yaratabilirsiniz: csc, cl, vjc' ya da vbc.

Dinamik assembly'ler: Bunlar SystemReflection.Emit ad uzayındaki sınıfları kullanarak çalışma zamanında dinamik olarak yaratabileceğiniz PE biçimli, bellek assembly'leridir.

Özel assembly'ler: Bunlar belirli bir uygulama tarafından kullanılan durağan assembly'lerdir.

Paylaşılmış assembly'ler: Bunlar belirli bir paylaşılmış adı olmak zorunda olan ve herhangi bir uygulama tarafından kullanılabilir durağan assembly'lerdir.

Bir uygulama özel bir assembly'yi durağan bir yol kullanarak ya da bir XML tabanlı uygulama yapılandırma dosyası aracılığıyla assembly'ye başvuru yaparak kullanmaktadır. CLR - doğru sürümün kullanılıp kullanılmadığını kontrol ederek- özel assembly'ler için sürümlendirme politikalarına zorlamasa da, bir uygulamanın beraber inşa edildiği doğru paylaşılmış assembly'leri kullandığını garanti etmektedir. Böylece, bir uygulama belirli paylaşılmış assembly'ye başvuru yaparak belirli bir paylaşılmış assembly'yi kullanmakta ve CLR çalışma zamanında doğru sürümün yüklendiğini taahhüt etmektedir.

. NET’te, bir assembly bir sürüm numarasını ilişkilendirebileceğiniz en küçük birimdir ve formatı aşağıdaki gibidir:

<major_version>.<minor_version>.<build_number>.<revision>

1.2.3.2. Dağıtma

Bir istemci uygulamanın assembly manifestosu (kısaca bahsetmek gerekirse) assembly adı, uygulamanın kullandığı sürüm gibi bilgileri harici başvurularda tuttuğundan, COM'da olduğu gibi etkinleştirme ve marshaling ipuçlarını depolamak için kaydediciyi kullanmak zorunlu değildir. Uygulamanın manifestosunda kayıtlı olan sürüm ve güvenlik bilgilerini kullanarak, CLR doğru paylaşılmış assembly'i yükleyebilmektir. CLS harici assembly'lerin yüklemesini yapabilmekte ve onların tipleri kullanılmak istendiğinde onları getirmektedir. Bundan dolayı, birçok küçük harici assembly'si olan küçük, indirilebilir uygulamalar oluşturmak mümkündür. Belirli bir harici assembly gerektiği zaman, çalışma zamanı kayıt ya da bilgisayarı yeniden başlatmaya gerek kalmadan onu otomatik olarak indirebilir.

1.2.3.3. Güvenlik

Bir kullanıcı kimliği kavramı tüm geliştirme ve işletim platformlarında karşımıza çıkmaktadır, ama küçük bir kod parçasının bile kimliğinin olduğu ticari yazılım endüstrisinde kod kimliği kavramı yenidir. .NET’te, bir assembly'nin kendisi assembly'nin paylaşılmış adı, sürüm numarası ve kodun nereden geldiği (yerel, Intranet ya da Internet) gibi bilgileri içeren bir kod kimliğine sahiptir. Bu bilgilere ayrıca assembly'nin kanıtı da denir ve kodun, özellikle de taşınabilir kodun, kimliklendirilmesine ve koda izin sağlanmasına yardımcı olur.

Bir kod kimliği kavramıyla uyumlu olmak için, CLR kod erişimi kavramını desteklemektedir. Kod'un kaynaklara erişebilmesi ya da diğer kodları kullanıp kullanamaması tamamen yöneticinin belirlediği ve CLR'ın kullanılmasını zorunlu kıldığı kurallar kümesi olan güvenlik politakasına bağlıdır. CLR assembly'nin kanıtını inceler ve hedef assembly'ye çalışması esnasında incelenecek olan bir izinler kümesi vermek için güvenlik politikasını kullanır. CLR bu izinleri kontrol etmekte ve assembly'nin kaynaklara ya da diğer kodlara erişimi olup olmadığını belirlemektedir. Bir assembly yarattığınızda,

istemci uygulamanın assembly'nizin kullanmak için sahip olması gereken bir izinler kümesini tanımlayabilirsiniz. Çalışma zamanında, istemci uygulama assembly'nize ait kodun erişimine sahipse assembly'nizin nesnelerine çağrılar yapabilir; aksi takdirde bir güvenlik kuraldışı durumu ortaya çıkacaktır. Ayrıca çağrı yığınındaki tüm kodların belirli kaynağa erişmesi için uygun izinlere sahip olmasını da mecburi kılabilirsiniz.

1.2.3.4. Yanyana Çalıştırma

CLR aynı, paylaşılmış DLL'nin tüm sürümlerinin aynı anda, aynı sistemde ve hatta aynı süreçte çalışmasına imkân vermektedir. Bu kavram yanyana çalıştırma olarak bilinir. Microsoft .NET yanyana çalıştırmayı paylaşılmış tüm assembly'lerde bulunan sürümlendirme ve dağıtma özelliklerini kullanarak uygular. Bu kavram aynı, paylaşılmış assembly'nin sürümlendirme çakışmaları ya da DLL Cehennemi söz konusu olmadan aynı makinede herhangi bir versiyonunun kurulmasına olanak verir. Tek uyarımız assembly'lerinizin public ya da paylaştırılmış assembly'ler olmasıdır. Önemli olan .NET'in sürümlendirme ve dağıtma özellikleri tarafından sağlanan bilgilerdir.

Belirli paylaşılmış bir assembly kullanan bir uygulama inşa ettiğiniz zaman, paylaşılmış assembly'nin versiyon bilgilerinin uygulamanızın manifestosuna eklendiğini unutmayın. Ayrıca paylaşılmış assembly'nin 8 bitlik public anahtarı da uygulamanızın manifestosuna eklenmektedir. Bu iki parça bilgiyi kullanarak, CLR uygulamanızın kullandığı paylaşılmış assembly'yi tam olarak bulabilir ve hatta 8 bitlik hash'in gerçekte paylaşılmış assembly'ninkinin eşdeğeri olduğunu da doğrulayabilir.

1.2.3.5. Paylaşım ve Yeniden Kullanım

Assembly'nizi dünyanın geri kalanıyla paylaşmak istediğinizde, assembly'nizin paylaşılmış ya da güçlü bir adı olmalıdır ve ayrıca onu GAC'ye kaydetmelisiniz. Benzer şekilde, belirli bir paylaşılmış assembly tarafından barındırılan belirli bir sınıfı kullanmak ya da genişletmek isterseniz, yalnızca bu sınıfı ithal etmez, aynı zamanda tüm assembly'nizi de uygulamanıza ithal edersiniz. Bu nedenle, tüm assembly bir paylaşım birimidir.

Assembly'ler .NET'te son derece önemli bir özellik olarak karşımıza çıkmaktadır, çünkü çalışma zamanının önemli bir parçasıdırlar. Bir assembly, assembly içinde

tanımlanmış tüm tipleri sarmalamaktadır. Örneğin, iki farklı assembly, Personal ve Company, aynı tipi, Car, tanımlayabilir ve kullanabilir. Siz onu [Personal]Car ya da [Company]Car olarak nitelendirmedikçe bir Car'ın tek başına bir anlamı yoktur. Bunu göz önünde bulundurarak, tüm tipler kendilerini içeren assembly'le sınırlıdırlar ve bu nedenle CLR tipin assembly'sini bilmediği sürece, söz konusu tipten faydalanamaz: Aslında, bir assembly'yi tanımlayan assembly manifestonuz yoksa CLR programınızı çalıştırmayacaktır.

1.2.3.6. Manifestolar: Assembly Metadatası

Bir assembly manifestosu, assembly'ye ait dosyaların bir listesini, harici assembly'lere başvuruları, ihraç edilmiş tipleri, ihraç edilmiş kaynakları ve izin taleplerini de içeren assembly hakkında her şeyi tanımlayan metadatadır. Kısaca, yazılım tak-ve-çalıştırı için gerekli olan tüm ayrıntıları tanımlamaktadır. Bir assembly tüm bu ayrıntıları içerdiğinden, COM dünyasında olduğu gibi bu tip bilgisini kaydedicide tutmak için bir neden yoktur.

COM'da belirli bir COM sınıfını kullandığınızda, COM kütüphanesine bir sınıf tanımlayıcısı vermiş olursunuz. COM kütüphanesi bu sınıfı kullanan, bileşeni yükleyen, bileşene bu sınıfın bir örneğini vermesini söyleyen ve bu örneğe bir başvuru döndüren COM bileşenini bulmak için kaydedicide arama gerçekleştirir. .NETte, registery'ye bakmak yerine, CLR doğrudan assembly'nin manifestasuna bakar, hangi harici assembly'nin gerektiğini belirler, uygulamanızın gereksinim duyduğu doğru assembly'i yükler ve hedef sınıfın bir örneğini yaratır.

```
. assembly extern mscorlib
{
.publickeytoken = (B7 7A 5C 56 19 34 EO 89 )
.ver 1:0:5000:0
.assembly hello {
.hash algorithm Ox00008004 .ver 0:0:0:0
}
.module hello.exe
II MVID: {F828835E-3705-4238-BCD7-637ACDD33B78}                    (1.1)
```

1.1'deki manifestonun harici ya da başvuru yapılmış bir assembly'i tanımlayarak işe başladığını görmek kolaydır. Bu özel uygulamanın başvuru yaptığı assembly'nin adı

mscorlib'dir. ".assembly extern" CLR'a bu uygulamanın mscorlib'i çalıştırmadığını, bunun yerine ondan faydalandığını söylemektedir. Bu harici assembly tüm .NET uygulamalarında kullanacağımız assembly'dir. Bu assembly tanımının içine, derleyicinin mscorlib'in yayımlayıcısı hakkındaki temel bilgileri içeren publickeytoken adında özel bir değer yerleştirdiğine dikkat edin. Derleyici .publickeytoken değerini mscorlib assembly'siyle ilişkilendirilmiş olan public anahtarı ufak parçalara ayırarak üretmektedir. mscorlib bloğunda dikkate değer bir başka şey de mscorlib'in versiyon numarasıdır.

İlk .assembly bloğunu incelediğimize göre, bu özel assembly'i tanımlayan ikincisini inceleyebiliriz. Bunun uygulamamızın assembly'sini tanımlayan bir manifesto bloğu olduğunu söyleyebilirsiniz, çünkü extern anahtar kelimesi bulunmamaktadır. Bu assembly'nin tanımı merhaba şeklindeki okunabilir bir assembly adından, sürüm numarası, 0:0:0:0 ve bizde bulunmayan isteğe bağlı bir kültürden oluşmaktadır. Bu bloğun içinde, ilk satır bu assembly'nin seçilmiş içeriğini parçalara ayırmak için kullanılan hash algoritmasını belirtmektedir. Bu algoritmanın sonucu private anahtar kullanılarak şifreli halde olacaktır. Bununla beraber, bu basit assembly'i paylaşmadığımızdan, hiçbir şifreleme ve .publickey değeri yoktur.

1.1 manifestosu için bahsedilecek son şey sadece bu assembly'nin, çıktı dosyasının adını (merhaba.exe) tanımlayan .module'dür. Bir modülün bir GUID ile ilişkilendirildiğine dikkat edin ve bu da modülü her inşa ettiğinizde farklı bir GUID değeri elde edeceğiniz anlamına gelmektedir. Bunu gözönüne alırsak, tam modül eşdeğeri için temel bir test iki modülün GUID'lerinin karşılaştırılmasıdır.

Bu örnek çok basit olduğundan, manifestomuz için gerekenler bunlarla sınırlıdır. Daha karmaşık bir assembly'de, assembly'nizin bileşimi hakkında çok daha fazla ayrıntı içerecek şekilde, bu kavramların tamamını kullanabilirsiniz.

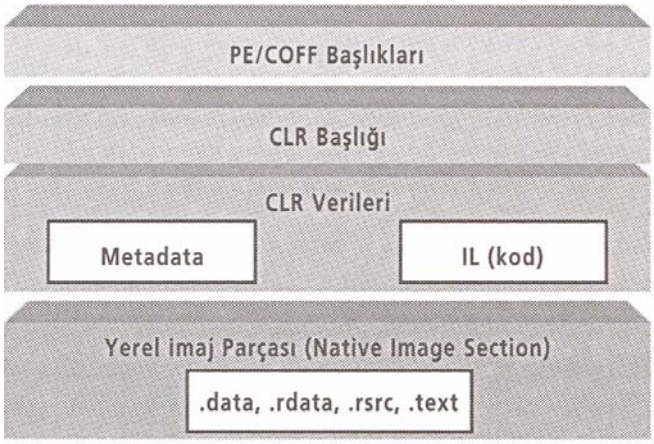
1.2.4. JIT Derleyicileri

JIT derleyicileri .NET platformunda önemli bir rol oynamaktadırlar, çünkü tüm .NET PE dosyaları, yerel kod değil, IL ve metadata içermektedir. JIT derleyicileri hedef işletim sisteminde çalışabilmeleri için IL'ı yerel koda çevirmektedirler. Tip güvenliği doğrulanmış her bir metot için, CLR'de bir JIT derleyicisi metodu derleyecek ve onu yerel koda çevirecektir.

Bir JIT derleyicisinin bir üstünlüğü hedef makine için optimize edilmiş kodu dinamik olarak derleyebilmesidir. Aynı .NET PE dosyalarını tek CPU'lu bir makineden iki CPU'lu bir makineye alırsanız, iki CPU'lu makinedeki JIT derleyicisi ikinci CPU'yu tanıyacak ve ondan faydalanması için yerel kodu açıklayabilecektir. Bir başka belirgin üstünlük, platform CLR'a sahip olduğu sürece, aynı .NET PE dosyasını alıp Windows, Unix ya da başka tamamen farklı herhangi bir platformda çalıştırabilmenizdir.

1.3. Portatif İcra Edilebilir Dosya Formatı

1.3.1. İmaj Dosyasının Yapısı



Şekil 4. PE dosya yapısı

Şekil 4'te bir imaj dosyasının yapısı görülüyor.

Koşma zamanı kodu icra edebilmek için makine tarafından kullanılan değişik veri formatları tanımlar. Bu veri formatlarına örnek olarak, metadata, IL metod gövdesi ve garbarage collection verilebilir. Her veri türü PE dosyasında herhangi bir bölümde (section) olabilir.

EXE ya da DLL olsun, bir Windows çalıştırılabiliri, Ortak Nesne Dosya Formatı'nın (Comman Object File Format - COFF) bir türevi olan PE adındaki dosya formatına uymak zorundadır. Bu formatların her ikisi de tam olarak tanımlı ve herkese açık durumdadır. Windows, DLL ve EXE'leri nasıl yükleyeceğini ve çalıştıracağını bilmektedir; çünkü bir PE dosyasının formatını anlamaktadır. Sonuç olarak Windows

çalıştırılabilirleri üretmek isteyen herhangi bir derleyici PE/COFF spesifikasyonuna uymak zorundadır.

Standart bir Windows PE dosyası bir MS-DOS başlığında başlayan, bir PE başlığıyla devam eden, isteğe bağlı bir başlık tarafından takip edilen ve son olarak NETteki .text, .data, .rdata ve .rsrc ayrımları dahil bazı yerel imaj parçalarına doğru giden bazı parçalara bölünmüştür. Bunlar bir Windows çalıştırılırının standart parçalarıdır, ama Microsoft'un C/C++ derleyicisi, bir #pragma yönergesi kullanarak kendi özel ayrımlarınızı PE dosyasına eklemenize imkân vermektedir. Örneğin, yalnızca kendinizin okuyabileceği şifrelenmiş verileri tutmak üzere kendi veri kısmınızı yaratabilirsiniz. CLR'ı desteklemek için, Microsoft PE/COFF dosya formatını metadata ve IL kodunu kapsayacak şekilde genişletmiştir. CLR sınıfları nasıl yükleyeceğini belirlemek için metadatayı ve çalıştırma işlemi için onu yerel koda çevirmek içinse IL kodunu kullanmaktadır. ŞekiI 4'te gösterildiği gibi, Microsoft'un normal PE formatına eklediği uzantılar CLR başlığını ve CLR verilerini içermektedir. CLR başlığı temel olarak CLR'ın program çalıştırmasına yardımcı olacak kalıcı bilgilerin tutulduğu konumlara giden bağıl sanal adreslerini (relative virtual addresses - RVA) depolamaktadır. CLR veri parçası, her ikisi de programın nasıl çalıştırılacağını belirleyen metadata ve IL kodunu kapsamaktadır. CLR'ı hedef alan derleyiciler hem CLR başlığını hem de veri bilgisini üretilen PE dosyasına koymak zorundadır, aksi takdirde elde edilen PE dosyası CLR altında çalışmayacaktır.

CLR, çalışma zamanı tarafından gereksinim duyulan bazı ilgili ayrıntıları tutmaktadır, örneğin:

Runtime sürümü: Bu programın çalıştırılması için gerekli olan çalışma zamanı sürümünü belirtmektedir.

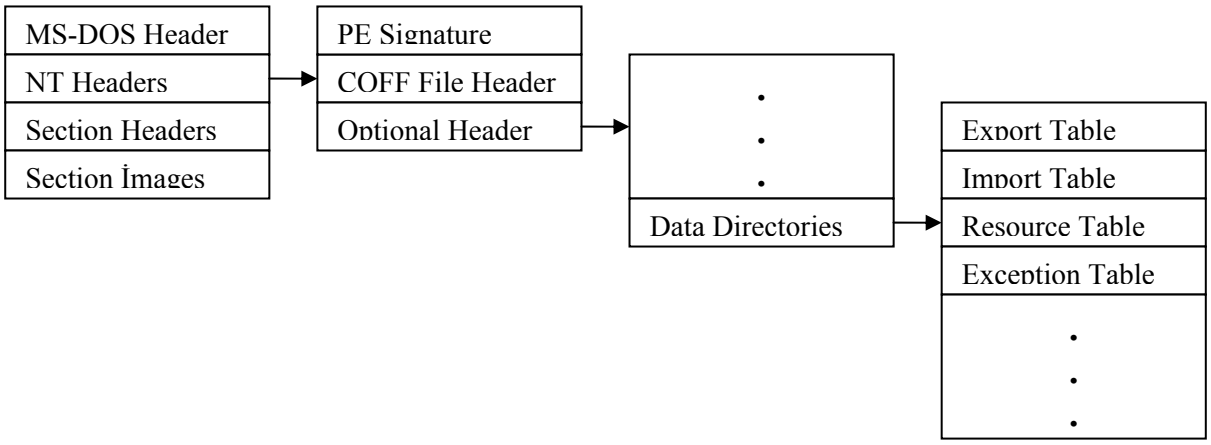
MetaData dizini: Çalışma zamanında CLR tarafından ihtiyaç duyulan metadatanın konumunu belirttiği için önemlidir.

Giriş noktası jetonu: Tek bir dosya assembly'si için Main( ) gibi CLR'ın çalıştırdığı giriş noktasını tanımlayan jeton olduğundan çok daha önemlidir.

Bu format Windows işletim sisteminin kaynak kodu icra etmesi ve koşan bir program için özel bilgileri (sabit veri, değişken veri, kütüphane linkleri ekleme, kaynak veri…) saklaması için en iyi yöntem olarak tanımlanır. Bu dosya MS-DOS dosya bilgisi, Windows NT dosya bilgisi, Bölüm Başlığı ve bölüm resimlerini içerir. Yapısı aşağıdaki gibidir

1.3.2. PE Başlığı

Yapısı Şekil 5'te gösterilmiştir.



Şekil 5. PE Başlığı

1.3.2.1. MSDOS Başlığı

Her PE dosyası MSDOS başlığıyla başlar. Bu Msdos datası içindeki en önemli kısım PE imzasının ofsetidir. Bu data Windows NT datasının pozisyonunu bulmak için kullanılacaktır. Yapısı Ek 1'de gösterilmiştir.

1.3.2.2. Msdos Stub Programı

Bu program msdos altında çalışan bir programdır. Exe imajından önce yer alır. İmaj dos altında çalışırılmak istenirse "bu program dos ortamında çalışamaz" mesajı üretilir ve sonra programı sonlandırır.

Stup programı 0x3c bölgesinde PE imzasına atlamak için gerekli olan ofset değerine sahiptir. Bu imza Windows'un imaj dosyasını çalıştırmasını etkinleştirir.

1.3.2.3. NT Başlığı

Msdos stub programından sonra 0x3c de belirtilen adrese atlanıldığı zaman dosyanın PE dosyası olduğunu belirten 4 byte uzunluğunda bir PE imzasına ulaşılır. Değeri "PE\0\0"'dir. (P ve E karakterlerini takip eden 2 null karakter). Eğer buradaki değer "PE\0\0" değilse PE dosyası olmadığını gösterir.

İmzadan sonra standart bir coff dosya başlığı vardır. Yapısı Ek 2'de gösterilmiştir. Dosya başlığında imaj dosyasının koşabileceği işlemci türleri, imaj dosyasındaki bölüm sayısı, imaj dosyasının oluşturulma tarihi, imaj dosyasının karakteristiği gibi bilgiler tutulur.

Her imaj dosyası yükleyiciye (loader) bilgileri sağlamak için opsiyonel başlıklara sahiptir. İmaj dosyası için bu başlık gereklidir. Obje dosyası da bu başlığa sahip olabilir fakat genelde bu opsiyonel başlık obje dosyasında fonksiyona sahip değildir ve bu yüzden boşuna dosya boyutunu arttırır.

Opsiyonel başlığın boyutu sabit değildir. Yapısı Tablo 2'de gösterildiği gibidir.

Tablo 2. Opsiyonel Başlık

Offset (PE32/PE32+)	**Boyut (PE32/PE32+)**	**Başlık Bölümü**	**Açıklama**
0	28/24	Standard Alanlar	COFF un tüm yürütmelerinde standard olarak tanımlanmıştır, UNIX i içerir.
28/24	68/88	Windows-Özel Alanlar	Windows'un desteklenen belirli özellikleri için ek alanlar (örneğin alt sistemler) içerir.
96/112	Değişken	Veri Dizinleri	İmaj dosyasında bulunan ve işletim sistemi tarafından kullanılan özel tablolar için Adres/boyut çifti(örneğin, import tablo, export tablo… gibi)

Opsiyonel başlığın içindeki ilk sekiz alan COFF'un tüm yürütmelerinde standart olarak tanımlanmıştır. Bu alanlar bir çalıştırılabilir dosyanın yüklenmesi ve icra edilmesi için gerekli olan genel bilgileri içerir. Ek 5'te yapısı verilmiştir.

Standart Alanlardaki magic numarası imaj dosyasının PE32 veya PE32+ executable olup olmadığına karar verir.

Tablo 3. Magic Number

Magic number	PE format
0x10b	PE32
0x20b	PE32+

PE32: 32 bitlik işletim sisteminde çalışan imaj dosyaları

PE32+: 64 bitlik işletim sisteminde çalışan imaj dosyaları. PE32+ imajları 2 GB imaj boyutu sınırına kadar 64 bit adres alanı adreslememize izin verir.

PE32 PE32+ da mevcut olmayan BaseOfCode u takip eden ek alanlar içerir. Ek alanların yapısı Tablo 4'te verilmiştir.

Tablo 4. PE32 ek alanlar

Offset	Boyut	Alan	Açıklama
24	4	BaseOfData	Adres belleğe yüklendiği zaman veri kısmının başlangıcının imajın tabanına göreceli adresi

Standart Alanları takip eden 21 alan COFF opsiyonel başlık formatının uzantısıdır. Windows'taki linker ve loader için gerekli olan ek bilgileri içerirler. Yapısı Ek 6'da verilmiştir.

Opsiyonel Başlığın son bölümü ise veri dizinleridir. Her veri dizini Windows'un kullanması için tablo boyutu ve adresten oluşur. Sistemin bu data dizini kayıtlarına erişebilmesi için bunlar koşma zamanında belleğe yüklenir. Her veri dizini 8 byte uzunluğunda alanlardan oluşur.

İlk alan olan "VirtualAdress" aslında tablonun RVA'sıdır. 2. alan ise boyutu byte türünde verir.

RVA: Relative virtual address. Win32'de çalıştırılabilir ve obje dosyalarının içindeki ofset referanslarına verilen isimdir.

Opsiyonel başlığın son bölümü olan veri dizinleri Ek 9'da listelenmiştir. Veri dizin sayısı sabit değildir. Özel bir veri dizinine erişmek için önce opsiyonel başlıktaki NumberOfRvaAndSizes e bakmak gerekir. Bizim için Opsiyonel başlıktaki en önemli

yapı veri dizinleridir. Çünkü 15. veri dizini CLR başlığının boyutunu ve konumunu gösterir. CLR başlığı çalışma zamanına özgü veri kayıtlarını ve çalışma zamanına özel diğer bilgilerin hepsini içerir.

1.3.2.4. Bölüm Başlığı

Bölüm tablosundaki her satır bölüm başlığının etkisi altındadır. Opsiyonel başlıktan hemen sonra başlar. Dosya başlığından bölüm tablosuna direk gösterge olmadığından konumu gereklidir.

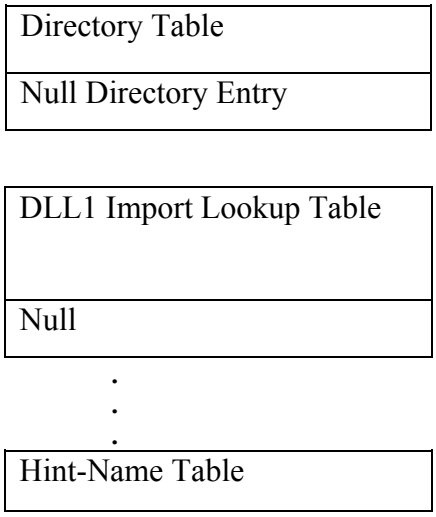
Bölüm tablosundaki giriş sayısı dosya başlığındaki NumberOfSections alanında belirtilir. Bölüm tablosundaki girişler birden başlayarak numaralandırılır. Kod ve veri bellek bölümleri linker tarafından seçilir.

İmaj dosyasında bölüm için VA linker tarafından ayrılır. Böylece bölümler artan sırada ve bitişik bir hal alır. Ve bölümlerin boyutu Opsiyonel başlıktaki SectionAlignment değerinin katı olmak zorundadır.

Bölüm başlığının yapısı Ek 10'da gösterilmiştir.

1.3.2.5. .Idata Bölümü (Import Tablosu)

Hemen hemen tüm exe dosyaları. idata bölümü olan import bölümüne sahiptir. Tipik yapısı Şekil 6'daki gibidir.



Şekil 6. Import tablosunun yapısı

1.3.2.5.1. Import Dizin Tablosu

Import bilgileri import bilgilerinin kalanını tanımlayan import dizin tablosuyla başlar. Adres bilgisini içerir. Her dll için dll giriş noktasının adresini çözmek için bir dizi içerir. Dizin tablosunun sonunu gösteren son dizin girişi null değerle doldurulmuştur.

Her import dizin girişi Tablo 5'te gösterilen formattadır.

Tablo 5. Import Dizin Tablosu

Ofset	Boyut	Alan	Açıklama
0	4	Import Lookup Table RVA (Characteristics)	Arama (lookup) tablosunun RVA'sı. Bu tablo her import için bir isim ve sıra içerir. (karakteristik ismi Winnt.h ta kullanılır, fakat uzun zamandır tanımlanmamıştır.). Hint adresini ve fonksiyon ismini tutar.
4	4	Time/Date Stamp	Bu damga imaj bağlı olana kadar sıfıra setlenir. İmaj bağlandığı zaman dll in tarih/zaman damgasına setlenir
8	4	Forwarder Chain	İlk ileri başvuru indeksini gösterir. API'nin ilk ileri zincirine işaret eder.
12	4	Name RVA	Dll adını içeren ascii stringin adresi. Bu adres imaj tabanına göreceli dir.
16	4	Import Address Table RVA	İmport adres tablosunun RVA sı. Bu tablonun içeriği imaj bağlanıncaya kadar import lookup tablosunun içeriğiyle aynıdır.

1.3.2.5.2. Import Lookup Tablosu

Import lookup tablusu pe32 için 32 bitlik bir diziden veya pe32+ için 64 bitlik bir diziden oluşur. Her bit formatı Tablo 6'da tanımlanmıştır. Bu girişlerin toplamı verilen DLL'in tüm importlarını tanımlar. Son tablonun sonunun geldiğini göstermek için giriş sıfırlanır (null).

Tablo 6. Import Lookup Tablosu

Bit(s)	Boyut	Bit Alanı	Açıklama
31/63	1	Ordinal/Name Flag	Eğer bu bit setlenmişse sırayla import edildiğini gösteriri, diğer durumda isme göre import edildiğini gösterir. PE32 için bit 0x80000000 ile maskelenir, pe32+ için ise bit 0x8000000000000000 ile maskelenir.
15-0	16	Ordinal Number	16 bitlik sıralama numarası. Sadece Ordinal/Name Flag numarası 1 ise kullanılır (sırayla import edilmişse).
30–0	31	Hint/Name Table RVA	Hint/name tablo girişi için 31 bitlik RVA adresi. Eğer Ordinal/Name Flag numarası 0 ise bu alan kullanılır (isme göre import edilmişse). PE32+ için 62–31 bitleri sıfırlanmalıdır.

1.3.2.5.3. Hint/Name Table

BirHint/Name tablosu tüm import bölümleri için yeterlidir. Her Hint/Name tablosundaki girişler Tablo 7'deki formattadır.

Tablo 7. Hint/Name Tablosu

Offset	Boyut	Alan	Açıklama
0	2	Hint	Export isim gösterge tablosu içindeki indeks. Eşleştirme ilk bu değerle denenir. Eğer başarısız olursa DLL in export isim gösterge tablosunda ikili arama yapılır.
2	Değişken	Name	İmportun ismini içeren bir ascii string. Bu string Dll içindeki açık isimle eşleşmiş olmalıdır. Bu string büyük küçük duyarlıdır ve null karakteriyle sonlandırılmıştır.
*	0 veya 1	Pad	Null baytı eklendikten sonra sıfır eki eklenir.

1.3.2.5.4. Import Adres Tablosu

Bu tablonun içeriği ve yapısı imaj bağlanıncaya kadar import lookup tablosunun içeriğiyle aynıdır. Bağlama boyunca import adresindeki girişler import edilmiş 32 bit (PE32 için) veya 64 bit (PE32+ için) sembollerin adresleriyle yeniden yazılır. Bu adresler

sembollerin asıl bellek adresleridir teknik olarak bunlar sanal adresler olarak adlandırılırlar. Genellikle bağlama işini loader (yükleyici) yapar.

1.3.3. CLR

1.3.3.1. Koşma Zamanı Başlığı

IMAGE_DIRECTORY_COR_DESCRIPTOR dizin kaydı (image optional header daki data directorydeki 15. alan) imaj dosyasındaki çalışma zamanı başlığının konumunu belirtir. Bu başlık çalışma zamanına özgü veri kayıtlarını ve çalışma zamanına özel diğer bilgilerin hepsini içerir. Bu başlık imajın sadece okunabilir ve paylaştırılabilir bölümünde konumlandırılmalıdır. Bu başlık Tablo 8'deki gibi tanımlanmıştır:

Tablo 8. Runtime başlığı

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	4	Cb	Başlığın byte olarak boyutu.
4	2	MajorRuntimeVersion	Programın çalışması için gerekli olan çalışma zamanının minimum sürümü. Bu değer çalışma zamanının birlikte derlendiği COR_VERSION_MAJOR makrosuyla eşleşir.
6	2	MinorRuntimeVersion	Versiyonun ikinci parçası. COR_VERSION_MINOR makrosunu kullanır.
8	8	MetaData	Bu imajdaki metadatanın konumu ve boyutu
16	4	Flags	Flaglar (bayraklar) bunun imaj dosyası olduğunu belirtir. Ayrıntılı bilgi için 1.3.3.1.1. "Koşma Zamanı Bayrakları" bölümüne bakınız.
20	4	EntryPointToken	İmajdaki kayıt noktalarının methoddefi için hatırlama noktaları.
24	8	Resources	CLR kaynaklarının konumu. Bu kaynaklar PE dosyasındaki win32 kaynak bölümleriyle aynı değildir.
32	8	StrongNameSignature	Bu PE dosyası için CLR yükleyici tarafından kullanılan bağlama ve versiyonlama için hash datasının konumu.

Tablo 8’ in devamı

Ofset	Boyut	Alan İsmi	Alan Tanımı
40	8	CodeManagerTable	Kod yönetim tablosunun konumu ve boyutu.
48	8	VTableFixups	Dosyadaki fonksiyon pointerlarının dizisini içeren konum ve boyut dizilerinin konumu.
56	8	ExportAddressTableJumps	Jump thunklarının nereye yazıldığını gösteren RVA’nın bir dizisinin konumu ve boyutu.
64	8	EEInfoTable	Ayrılmış Alan.
72	8	HelperTable	Ayrılmış Alan.
80	8	DynamicInfo	Ayrılmış Alan.
88	8	DelayLoadInfo	Ayrılmış Alan.
96	8	ModuleImage	Ayrılmış Alan.
104	8	ExternalFixups	Ayrılmış Alan.
112	8	RidMap	Ayrılmış Alan.
120	8	DebugMap	Ayrılmış Alan.
128	8	IPMap	Eskimiş, kullanılmıyor.

1.3.3.1.1. Koşma Zamanı Bayrakları

Tablo 9’daki bayraklar runtime imajını tanımlar ve loader tarafından kullanılır:

Tablo 9. Runtime bayrakları

Bayrak	Değer	Açıklama
COMIMAGE_FLAGS_ILONLY	0x00000001	Imaj sadece IL kodu içerir. Bu yüzden herhangi özel bir işlemcide koşmasına gerek yoktur.
COMIMAGE_FLAGS_32BITREQUIRED	0x00000002	Imaj sadece 32 bitlik süreç içinde yüklenmiş olmalıdır. Bu bit imajın koşacağı sistemi gösterir. Eğer bit 1’e setlenmişse imaj sadece 32 bitlik makinada koşabilir. Ve 64 bit çalışma zamanı imajı yükleyemez.
COMIMAGE_FLAGS_TRACKDEBUGDATA	0x00010000	Setlendiği zaman çalışma zamanı ve JIT , metodlar hakkındaki debug bilgisi yolu için gereklidir.

1.3.3.1.2. Entry Point Meta Data Token

Entrypoint sembolü daima bir metotdef sembolüdür. Eğer verilmişse daima geleneksel çağrılması yönetilen (managed) yöntemdir.

Bir dll imajı için entrypoint: int DllMain(HINSTANCE, DWORD, void *)

bir exe imajı içinse : void DEFAULT main(Microsoft.Runtime.String[]) gibidir.

PE başlığındaki giriş noktaları (entry point) daima x86 stub (çalışma zamanını yükleyen ve çağıran)'ı veya 0 (runtime windowsun sürümünün farkında olduğunda) olmalıdır. Eğer yönetilen ve yönetilmeyen kod aynı imajda karışmışsa bu durgun durumdur. Bir imaj için yönetilen kodu giriş noktası yapmak için metod için bir pinvoke metoddef tanımlanır ve bu simge kullanılır.

1.3.3.2. Metadata Başlığı

CLR başlığında metadata RVA'sını bulunup metadata başlığına atlanır. Metadata başlığının yapısı Tablo 10'da gösterilmiştir.

Tablo 10. Metadata başlığı

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	0	BSJB	Metadatanın başladığını gösteren imza
4	2	Major	Metadatanın Major Versiyon numarası (1 veya 0 değerini alabilir)
6	2	Minor	Metadatanın Minor numarası (1 veya 0 değerini alabilir)
8	4		Ayrılmış Alan
12	4	Length	Ardından gelen versiyon numarası stringinin boyutunu gösterir.
16	m	Versiyon	.Net framework sürüm numarası
16+m			Burada 32 bitlik bir işlemcide hizalama 4 byte sınırında olduğundan flaga gitmeden 32 bite tamamlıyoruz (bu 32 bitlik işlemciler için geçerli).
X	2	Flag	
X+2	2	Streams	Akım sayısı
X+4		Stream Header	Akım başlığının konumu

1.3.3.2.1. Akım (Stream) Başlığı

Akım başlığının yapısı Tablo 11'de gösterilmiştir.

Tablo 11. Akım başlığı

Offset	Boyut	Alan İsmi	Alan Tanımı
0	4	Ofset	Akımın metadatanın konumuna göre başladığı konumu gösterir.
4	4	Size	Akımın boyutu
8	x	Name	Akımın ismi. Null karakteriyle sonlandırılmış bir stringtir.

Metadatada herşey akım olarak saklanır. Bir CLR başlığında birden fazla akım vardır. Bunların maksimum sayısı beştir. Metadata başlığında akım sayısı önceden alınmıştır. Bir döngü içerisinde her akımın sırasıyla bilgisi alınır.

Akımın boyutu herzaman dördün katı olmalıdır.

Tüm akım isimleri # karakteriyle başlar. Bu akımların isimleri:

#~ : metadata akımı

#string : string akımı

#guid : guid akımı

#blob : blob akımı

#us : us akımı

String Akımı: dosyadaki stringlerin tutulduğu yerdir. Örneğin, method isimleri, method reference isimleri, field reference isimleri, field isimleri hep burada tutulurlar.

Blob Akımı: Bloblar komplike data yapılarıdır. Burada bir byte[] şeklinde bulunan imzalar çözülerek kullanıldıkları yere göre değişik anlamlar içeren veriler barındırırlar. Örneğin, bir method'a ait referans için o method'un tokeni yada signature'ı gibi.

GUID Akımı: Programda kullandığınız GUID'leri barındırır.

Us Akımı: User String'leri yani kullanıcı tanımlı mesaj ya da benzeri stringleri içinde barındırır.

Metadata Akımı: MetaData Tablolarını barındırır.

1.3.3.2.2. Metadata Tablosu

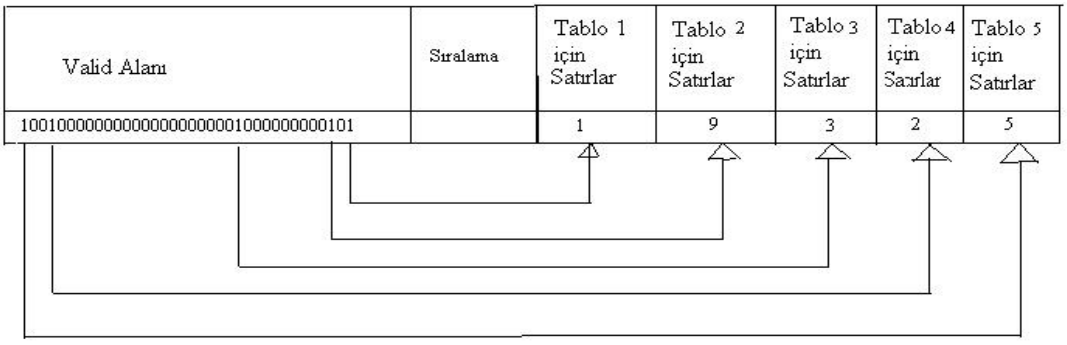
Tablo 12 Metadata tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	4	Reserved	Gelecekte kullanılmak için ayrılmış alan.
4	1	Major Version	Şema tablosunun birincil versiyon numarası.
5	1	Minor Versiyon	Şema tablosunun ikincil versiyon numarası.
6	1	Heapsize	Kümenin boyutu.
7	1	Reserved	Gelecekte kullnılmak için ayrılmış alan.
8	8	Valid	Metadatadaki hangi tabloların kulanıldığını gösteren değerler. Metadatada 64 tane tablo çeşidi vardır. Valid değerinin bitlerinden hangisi 1 ise o tablolar kullanılmıştır.
16	8	Sorted	
24	4*n	Rows	N geçerli tablo sayısıdır. Burada tabloların kaç satırdan oluştuğu bulunur.
24+4*n		Tables	Metadada tabloları

Tüm metadatalar tablo biçiminde saklanır. Metadata'da 43 tablo çeşidi vardır. Tüm class ve türler bir tablo içerisinde saklanır. Tablo isimleri bir string dizisinde tutulur.

Metadatada ilk tablonun offset değeri 24 tür. Bu değerden sonra her tablo için 4 byte eklenerek ilgili tablonun satır sayısını alacağımız offset değeri bulunabilir. Örneğin ilk tablonun satır sayısını alacağımız offset değerini okuyabilmek için Metadata'da 24. byte gidilir. 24. byte'da 4 bytelik satır sayısı değeri okunur. 2. tablonun satır sayısını hesaplamak için 28. byte gidilir ve buradan 4 byte'lik veri okunur.

Bu yapı Şekil 7'de gösterilmiştir.



Şekil 7. Metadata tablo satırları

String akımı null ile sonlandırılmış stringlerden oluşur. Aşağıda kod (1.3)'te gösterildiği gibi string akımındaki tüm stringler ekrana basılır.

```
for(int k=0;k<sisze[1];k++)
{
Console.Write((char)strings[k]);
if(strings[k]==0)
Console.WriteLine();                                    (1.3)
}
```

1.3.3.2.3. Metadata Tablo Türleri

Her tablo eşsiz bir değere sahiptir. Metadata tablo türleri ve boyutları Ek Tablo 11'de gösterilmiştir:

Sırayla bu tablo türlerinden bazılarını inceleyelim:

1.3.3.2.3.1. Modül Tablosu

Valid değerinin sıfırıncı biti ile gösterilen birinci tablo modül tablosunu tanımlar. Tek satırlık basit bir yapısı vardır. Yapısı Tablo 13'teki gibidir:

Tablo 13. Modül tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Generation	Ayrılmış alan, 0 olmalıdır
2	2	Name	Modülün ismini belirten , string tablosuna bir indeks değeridir.
4	2	MVid	GUID kümesine bir indeks değeri.
6	2	Encld	GUID kümesine bir indeks değeri.
8	2	EncBaseld	GUID kümesine bir indeks değeri.

Her tablo kendi özel yapısına sahiptir. Modül tablosunun ilk 2 byte'ı ayrılmış kuşak alanıdır ve değeri daima sıfıra setlenir. İkinci alan ise Name (isim) alanıdır. String tablosuna bir indeks değeri içerir. Bu değerle string akımından isim alınır.

Mvid Guid heap (kümesi) içine bir indekstir. Eğer program yeniden derlenirse bu değerin değiştiğini göreceksiniz. Derleyici her yeni bir exe dosyası oluşturduğunda yeni bir guid üretir. Bu modülün iki değişik versiyonunu ayırt edebilmemizi sağlar.

Sonraki iki alan guid içine bir indekstir ve değerleri 0 olarak ayrılmıştır.

1.3.3.2.3.2. TypeRef Tablosu

Bu tablo valid sayısındaki ikinci bit ile belirtilir. Fakat bu tablonun içeriğini almak için ofsetinin kaç olacağı hesaplanmak zorundadır. Örnek olarak ilk tablo olabilirde olmayabilir de, eğer mevcutsa ilk tablonun birden fazla satırı olabilir. Her bir durumda bu şu anki tablonun ofset değeri önceki tabloların mevcut olup olmamasına ve önceki tabloların satır sayılarına bağlıdır.

Her tablonun tek satırı için gerekli olan boyut sabit değerdedir ve bir dizide tutulur.

TypeRef tablosu 6 byte'dan oluşur ve 3 alanı vardır. Her alandaki değer string akımı içine bir indeks değeridir. Bu tablo programda kullanılan tüm classları ve türleri belirtir. Yapısı Tablo 14'te gösterildiği gibidir:

Tablo 14. TypeRef tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	2	Resolution Scope	Takip eden 4 değerden birini belirtir: Module, ModuleRef, AssemblyRef veya TypeRef. Ilk 2 bit tabloyu belirtir, kalan 6 bit ise tabloya olan indeks değerini belirtir.
2	2	Name	Class adını belirtir. Maksimum boyutu MAX_CLASS_NAME ile belirtilmiştir.
4	2	Namespace	Namespace'i belirtir

Resolution scope değeri belirtilen 4 tablodan birine olan index değeridir. Bu değer namespacenin alanını belirtir. İlk 2 biti alanı belirtir. (Module, ModuleRef, AssemblyRef veya TypeRef.) . Kalan 6 bit ise ilk 2 bit ile belirlenen tabloya olan indeks değeridir.

1.3.3.2.3.3. TypeDef Tablosu

TypeDef tablosu Assemmbly'de oluşturulan tüm sınıf ve türleri saklar. Bir tür bir sınıf, interface, structure, enum ve benzeri bir yapı olabilir. Dosyaya eklenen her yeni bir tür için yeni bir satır eklenir. Bu tablonun yapısı aşağıdaki gibidir:

Tablo 15. TypeDef tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	4	Flags	Türün niteliklerini belirler.
4	2	Name	Türün ismi
6	2	NameSpace	NameSpace
8	2	Extends	TypeDef, TypeRef veya TypeSpec değerlerinin varsayılanına bir kod indeksidir.
10	2	FieldList	Field tablosuna indeks değeri.
12	2	MethodList	Method tablosuna indeks değeri.

Extends değerine göre TypeDef, TypeRef veya TypeSpec tablolarından birine indeksi gösterir. Eğer değeri 0 ise geçersiz indekstir. Örneğin module herhangi bir classtan genişlemez, bu yüzden buradaki değeri 0'dır.

1.3.3.2.3.4. Metod Tablosu

Bu tabloda modülde oluşturulan her metod için bir tane satır vardır. Yapısı Tablo 16'da gösterildiği gibidir:

Tablo 16. Metod tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	4	RVA	Kodun başlama noktasının RVA sı.
4	2	IMPL Flags	Metoda uygulanan nitelikleri gösteren bayrak.
6	2	Flags	Metodun özellikleri gösteren bayrak. Her bit bazı özellikleri ifade eder. (static, private … gibi)
8	2	Name	Metodun ismini belirtmek için string tablosuna bir indeks değeri.
10	2	Signature	Blob kümesinde saklanan imza için indeks değeri. Fonksiyon hakkındaki tüm bilgileri belirtir.
12	2	ParamList	Metodun Parametre Listesi için Param Tablosuna bir indeks değeridir.

IMPL Flags : Metoda uygulanan en temel 2 nitelik vardır. Bunlar managed ve unmanaged'tir. Unmanaged metotlar C#'ta pointer ile kullanılırlar.

Eğer ilk bit sıfır ise bu metodun cil ve managed olduğunu gösterir, bir ise metodun native olduğunu gösterir. 0x20 metodun tek bir thread yada senkron olduğunu gösterir. C#'taki lock ifadesi bu durum için kullanılır.

Signature: Bu değer blog kümesindeki indeksi gösterir. Blog kümesinde bu indekste okunan değer metodun gerçek imzasını takip eden kaç byte olduğunu gösterir. Yani a metodu için bulunan imza sayısı 3 ise bu metodun imzası 3 byte alan kaplıyor anlamına gelir. Alabileceği değerler Ek 12'de belirtilmiştir.

1.3.3.2.3.5. MemberRef (MethodRef) Tablosu

Bu tablonun olup olmadığı valid değerinin 10. bitiyle kontrol edilir. Gerçek boyutu 6 byte'tır. Yapısı Tablo 17'de gösterilmiştir.

Tablo 17. MemberRef tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Class	Metodun hangi classa ve namespace ye ait olduğunu belirtir.
2	2	Name	Modulde kullanılan metodun ismi
4	2	Signature	

Metodun hangi classa ait olduğunu belirtmek için class byte'ındaki ilk 3 bite bakılır. Bu 3 bitin değerine göre TypeRef, ModuleRef, Method, TypeSpec veya TypeDef tablolarından birine başvurulur. Ilk 3 bitten sonraki bitler ise ilgili tabloya olan indeks değeridir.

Tablo 18. Class'ın ilk 3 bitine göre sonuç değerleri

Değer	TabloAdı
0	Kullanılmıyor
1	TypeRef
2	ModuleRef
3	ModuleDef
4	TypeSpec

1.3.3.2.3.6. Custom Attribute Tablosu

Attribute'ler compiler tarafından eklenir. Yapısı aşağıdaki gibidir:

Tablo 19. Custom Attribute Tablosu

Ofset	Boyut	Alan İsmi
0	2	Parent
2	2	Type
4	2	Value

İlk alan parent olarak adlandırılır ve HasCustomAttribute kod indeksine sahiptir. Parent alanındaki ilk 5 bit tabloyu bulmak için kullanılır. Olası değerler 0 ile 18 arasıdır. Bu değerler Ek 14'te gösterilmiştir.

İkinci alan CustomAttributeType kodlama tablosuna olan indekstir. Bunun da ilk 3 bitine bakılarak ilgili tabloya gidilir. Olası 5 değer vardır ve bunlar aşağıdaki gibidir.

Tablo 20. Type alanının ilk 3 bitinin alabileceği değerler

CustomAttributeType: 3 bits to encode tag	**Tag**
Kullanılmıyor	0
Kullanılmıyor	1
MethodDef	2
MemberRef	3
Kullanılmıyor	4

Son iki byte (value) blob kümesine olan indeks değeridir. CustomAttribute blob kümesindeki dataları içerir.

1.3.3.2.3.7. Assembly Tablosu

Bu tablonun olup olmadığı valid değerindeki 32. bit ile kontrol edilir. Bir assembly'nin detaylarını saklar. Bir assembly birçok modülden oluşur. Assembly tablosu ya 0 yada 1 satır içerir. Yapısı Tablo 21'de gösterilmiştir.

Tablo 21. Assembly tablosu

Ofset	**Boyut**	**Alan İsmi**	**Alan Tanımı**
0	4	HashAlgID	System.Configuration.Assemblies te belirtilen hash algoritmalarından birini belirten bir id.
4	2	Major	Assembly'nin asıl sürüm numarası.
6	2	Minor	Assembly'nin ikincil sürüm numarası.
8	2	Build	Assembly'nin build sürüm numarası.
10	2	Revision	Assembly'nin düzeltme sürüm numarası.
12	4	Flags	Flag'ı gösteren bir enum değer.

Tablo 21. Assembly tablosu'nun devamı

Ofset	Boyut	Alan İsmi	Alan Tanımı
16	2	Public Key	Blob kümesine bir index değeri. 0 ise geçersizdir.
18	2	Name	String kümesine bir indeks değeri. Assembly'nin ismini belirtir.
20	2	Culture	String kümesine bir indeks değeri. Assembly'nin dil bilgilerini gösterir.

3 tane mümkün hash değeri olduğu varsayılır. Bunlar Tablo 22'de gösterilmiştir.

Tablo 22. Olası Hash değerleri

Algoritma	Değer
Yok	0x0000
Ayrılmış (MD3)	0x8003
SHA1	0x8004

1.3.3.2.3.8. AssemblyRef Tablosu

İmaj dosyasında başvurulan tüm assemblyleri saklar. Valid değerinin 25. bitine bakılarak kullanılıp kullanılmadığı anlaşılır. Her assembly için bir satırdan oluşur. Yapısı Tablo 23'te gösterilmiştir.

Tablo 23. AssemblyRef tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	2	Major	Assembly'nin asıl sürüm numarası.
2	2	Minor	Assembly'nin ikincil sürüm numarası.
4	2	Build	Assembly'nin build sürüm numarası.
6	2	Revision	Assembly'nin düzeltme sürüm numarası.
8	4	Flags	Flag'ı gösteren bir enum değer.
12	2	PublicKeyOrToken	Blob kümesine bir index değeri. 0 ise geçersizdir.
14	2	Name	String kümesine bir indeks değeri. Assembly'nin ismini belirtir.

Tablo 23'ün devamı

Ofset	Boyut	Alan İsmi	Alan Tanımı
16	2	Culture	String kümesine bir indeks değeri. Assembly'nin dil bilgilerini gösterir.
18	2	HashValue	Blob kümesine bir index değeri.

1.3.3.2.3.9. Fields Tablosu

Bir örnek değişken field olarak da adlandırılır. Bu tabloda bu örnek değişkenler saklanır. Örneğin bir instance fonksiyon veya değişken gibi. Yapısı Tablo 24'te gösterilmiştir.

Tablo 24. Alanlar tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	2	Flags	FieldAttributes flaglarını gösterir.
2	2	Name	Fieldin ismini belirtmek için string kümesine bir indeks değeri.
4	2	Signature	Son alan ise Blob kümesine bir indekstir. Burada alınan değere göre bir tür dönderilir. (string,int … gibi). İlk byte daima 0x06 olmalıdır.

Eğer aynı değişken farklı sınıflarda kullanılmışsa ikisi de fields tablosunda ayrı birer satırda gösterilir.

1.3.3.2.3.10. Sabitler Tablosu

Modülde oluşturulan sabitleri saklar. Yapısı Tablo 25'te gösterildiği gibidir.

Tablo 25. Sabitler tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	1	type	Sabitin veri türünü gösterir.
1	1	Padded Byte	Değeri 0'dır ve ekleme yapmak için kullanılır.

Tablo 25'in devamı

Ofset	Boyut	Alan İsmi	Alan Tanımı
2	2	Parent	HasConst kodlama indeksini gösterir. İlk 2 bit tabloyu gösterir. Kalan 6 bit ise ilgili tablodaki indeks değerini gösterir.
4	2	Value	Sabitin değerini gösterir. Blob kümesinde değeri saklanır. İlk byte boyutunu gösterir. Diğerleri ise sabitin değerini belirtir.

Parent'ın ilk iki bitine göre Field, Param veya Property tablosu olabilir. Bu değerler aşağıda Tablo 26'da gösterilmiştir.

Tablo 26. Parent alanının ilk 2 bitinin olası değerleri

HasConstant: 2 bits to encode tag	Tag
FieldDef	0
ParamDef	1
Property	2

1.3.3.2.3.11. Nested Tablosu

Her iç içe class için Nested Class tablosuna bir tane satır eklenir. İki alandan oluşur. Her ikiside typedef tablosuna bir indeks değeridir. Yapısı Tablo 27'de gösterildiği gibidir.

Tablo 27. Nested tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Nested Class	İçerdeki sınıfı belirtir. Typedef Tablosuna bir indekstir.
2	2	Enclosing Class	Kapsayan Sınıfı belirtir. Typedef tablosuna bir indekstir.

1.3.3.2.3.12. Param Tablosu

Metodlara ayrılmış parametreler bu tabloda saklanır. Valid alanının 8. biti kontrol edilerek var olup olmadığı bulunur. Yapısı Tablo 28'de gösterilmiştir.

Tablo 28. Param tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Flags	Fonksiyon parametresinin niteliklerini gösterir. Alabileceği değerler Tablo 29'da gösterilmiştir.
2	2	Sequence	Parametrenin Sırası
4	2	Name	Parametresinin ismi

Tablo 29. Flag alanının değerine göre nitelikler

pdIn	0x0001
pdOut	0x0002
pdOptional	0x0010
pdReservedMask	0xf000
pdHasDefault	0x1000
pdHasFieldMarshal	0x2000
pdUnused	0xcfe0

1.3.3.2.3.13. Propertymap Tablosu

Property map tablosu valid değerinin 21. bitiyle kontrol edilir. 2 alanı vardır. Yapısı Tablo 30'de gösterildiği gibidir:

Tablo 30. Propertymap tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Parent	TypeDef tablosuna bir index
2	2	Properties Map	Properties tablosuna bir index

Parent farklı sınıflarda aynı isimde property varsa bunları ayırt etmemizi sağlar. Yani propertynin hangi sınıfa ait olduğunu kontrol eder.

1.3.3.2.3.14. Property Tablosu

Properties tablosu valid alanının 23. bitiyle kontrol edilir. Property tablosunun yapısı ise Tablo 31'te gösterildiği gibidir:

Tablo 31. Property tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Parent	TypeDef tablosuna bir index
2	2	Properties Map	Properties tablosuna bir index

Her property için bu tabloya bir satır eklenir.

1.3.3.2.3.15. MethodSematics Tablosu

Yapısı Tablo 32'de gösterildiği gibidir:

Tablo 32. MethodSematics tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Semantics	Semantics Nitelikleri. Olası değerleri Tablo 33'de gösterilmiştir.
2	2	Method	Metodun ismi (Metod Tablosuna bir indeks)
4	2	Assocation	Properties tablosuna bir ilişim

Tablo 33. Semantics olası değerleri

msSetter	0x0001
msGetter	0x0002
msOther	0x0004
msAddOn	0x0008
msRemoveOn	0x0010
msFire	0x0020

1.3.3.2.3.16. FieldLayout Tablosu

Yapısı Tablo 34'da gösterildiği gibidir:

Tablo 34. FieldLayout tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	4	offset	Field Tablosuna indeks
4	2	Fieldindex	Metodun ismi (Metod Tablosuna bir indeks)

1.3.3.2.3.17. Pinvoke Tablosu

Dll lerden (.net ile yazılanlar değil) fonksiyonlara ihtiyaç duyulduğunda kullanılır. Yapısı Tablo 35'de gösterildiği gibidir:

Tablo 35. Pinvoke tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	2	Mapping Flags	Nitelikler
2	2	Member Forwarded	Bir kod indeksidir. İlk 2 bitine bakarak hangi tabloya gideceğimixi belirleriz. Diğer bitler ise ilgili tablodaki ofset değeridir.
4	2	Import Name	Importun ismini belirler. (string tablosuna bir index)
6	2	Import Scope	ModuleRef tablosuna bir indeks. Metodun dll'i belirler.

Mapping Flags alacağı değerler aşağıda Ek Tablo 13'te gösterilmiştir.

Tablo 36. Member forwarder

MemberForwarded: 1 bit to encode tag	Tag
FieldDef	0
MethodDef	1

1.3.3.2.3.18. Interface Tablosu

Valid değerinin 9. biti ile var olup olmadığı kontrol edilir. Yapısı Tablo 37'de gösterildiği gibidir:

Tablo 37. Interface tablosu

Offset	Boyut	Alan İsmi	Alan Tanımı
0	2	Class	Typedef tablosuna bir indeks değeri.
2	2	Interface	TypeDefOrRef tablosuna bir indeks değeri. Ilk 2 bit hangi tablo olduğunu kontrol eder. Kalan bitler ise indeks değerini belirler.

1.3.3.2.4. Metod Kod Bloğu

Bu bölüm imajdaki il metotlarının nasıl formatlandığını ve verildiğini gösterir. Bir metod COR_ILMETHOD metod yapısı ve metot için komutlardan oluşur. Bu metod yapısı için çağrı tanımlayıcı RVA içerir (metot tablosundaki RVA). Bu bölüm için herhangi bir hizalama gerekli değildir. COR_ILMETHOD metod hakkında tüm bilgileri tanımlar. Bu yapı aşağıdakileri içerir:

İşlenen yığın için gerekli olan kaynakların miktarı

Yerel değişkenler için kaynakların miktarı

Locspace komutları için kaynakların miktarı

İstisna tutma bilgileri

1.3.3.2.4.1. Yerel Değişken Planı

COR_ILMETHOD yapısının bir fonksiyonu yerel değişkenlerin planını belirtir. Bu iki nedenden dolayı yapılır.

GC zamanında garbarage collection kümesi içindeki tüm pointerları bulabilmek

Yerel framelerin hesaplanabilmesi için yerel değişkenlerin boyutunu göstermek için

1.3.3.2.4.2. Dosya Formatı Yapısının Tanımlaması

Bu bölüm IL metodu ve onun istisnalarını tanımlamak için kullanılan veri yapılarının yapısını içerir.

1.3.3.2.4.2.1. Metot Gövdesi

Metodun gövdesi genel türde tanımlanır. İstisna tutma verisi metod gövdesinden sonra verilir. Eğer istisna tutma verisi mevcutsa CorILMethod_MoreSects metod başlığında belirtilmelidir.

1.3.3.2.4.2.1.1. Metot Başlığı Tür Değişkenleri

Metod başlığındaki ilk byte'ın en anlamlı 3 biti ne tür başlığın mevcut olduğunu belirtir. Bu 3 bit Tablo 38'de belirtilen türlerin sadece birini gösterir.

Tablo 38. Metod başlığı türleri

İsim	Değer	Açıklama
CorILMethod_TinyFormat	2	Metod başlığı IMAGE_COR_ILMETHOD_TINY yapısı tarafından tanımlanır ve kodun boyutu çift değerdedir.
CorILMethod_TinyFormat1	6	Metod başlığı IMAGE_COR_ILMETHOD_TINY yapısı tarafından tanımlanır ve kodun boyutu tek değerdedir. Tiny biti için 2 değer kullanılması boyut için extra bir bite izin içindir.
CorILMethod_FatFormat	3	Metod başlığı IMAGE_COR_ILMETHOD_FAT yapısı tarafından tanımlanır.

1.3.3.2.4.2.1.2. Tiny Format

IMAGE_COR_ILMETHOD_TINY yapısı 5 veya 6 bit uzunluğunda iki şekilde gelir. Aşağıdakiler tüm Tiny formatlar için uygundur.

İzin verilen yerel değişken yoktur

İstisnalar yoktur

Ekstra veri bölümleri yoktur

İşlem yığını gereksinimi 8 bytedan daha fazlasına gereksinim duymaz.

Birinci kodlama Tablo 39'da gösterilen şekilde olacaktır:

Tablo 39. Tiny Format 1. kodlama türü

Başlama Biti	**Bit Sayısı**	**Açıklama**
0	2	CorILMethod_TinyFormat = 0x2
2	6	Metod gövdesinin boyutu bu başlığı hemen takip eder. Metodun boyutu çift ve 2^6 'nın altındaysa kullanılır.

İkinci kodlama ise Tablo 40'ta gösterilen şekilde olacaktır.

Tablo 40. Tiny Format 2. Kodlama Türü

Başlama Biti	**Bit Sayısı**	**Açıklama**
0	3	CorILMethod_TinyFormat1 = 0x6
3	5	Metod gövdesinin boyutu bu başlığı hemen takip eder. Metodun boyutu tek ve 2^5 nın altındaysa kullanılır.

1.3.3.2.4.2.1.3. Fat Format

Tiny formatın çalışmayacağı işlerde fat format kullanılır. Bu aşağıdaki nedenlerden en az biri için geçerlidir:

Metod boyutu kodlanmak için çok büyüktür.

İstisnalar vardır.

Extra veri bölümleri vardır.

Yerel değişkenler vardır.

İşlem yığını 8 bytedan daha fazlasına gereksinim duyar.

Başlığın ilk bytenın kodlanması bu yüzden Tablo 41'de gösterildiği gibidir.

Tablo 41. Fat Format Kodlama Türü

Başlama Biti	Bit Sayısı	Açıklama
0	2	CorILMethod_Fat = 0x3
2	1	Ayrılmış
3	1	CorILMethod_MoreSects = 0x8 veya 0 ise daha fazla bölüm olmadığını gösterir.
4	1	CorILMethod_InitLocals = 0x10 veya 0 ise yerel değişken initinin olmadığını gösterir.

1.3.3.2.4.2.1.4. Image_Cor_Ilmethod

Bu yapının bir örneği için metadata noktalarıdır. Metot başlığı DWORD olarak hizalanmış olmalıdır. Tiny veya fat olarak iki mümkün format yayımlamakta kullanılır. Bu yapı bu iki yapının birleşiminden oluşur. İlk byte hangi türün mevcut olduğunu gösterir. Daha fazla bilgi için metot başlığı tür değerlerine bakınız.

```
typedef union
{
   COR_ILMETHOD_TINY      Tiny;
   COR_ILMETHOD_FAT       Fat;
} COR_ILMETHOD;                                    (1.4)
```

1.3.3.2.4.2.1.5. Image_Cor_Ilmethod_Tıny

Bu yapı DWORD hizalanmış olmalıdır. Yapısı (1.5)'teki gibidir.

```
typedef struct IMAGE_COR_ILMETHOD_TINY
{
   BYTE Flags_CodeSize;
} IMAGE_COR_ILMETHOD_TINY;                         (1.5)
```

1.3.3.2.4.2.1.6. Image_Cor_Ilmethod_Fat

Bu yapı daima DWORD hizalanmış olmalıdır. Yapısı Tablo 42'de gösterildiği gibidir.

Tablo 42. IMAGE_COR_ILMETHOD_FAT

Ofset	**Boyut**	**Alan**	**Açıklama**
0	12 bit	Flags	Bayraklar.
12	4 bit	Size	DWORD larla doldurulmuş gibi ifade edilen bu başlığın boyutu.
16	16 bit	MaxStack	İşlem yığınındaki maksimum öğe sayısı
4	4	CodeSize	Asıl metod gövdesinin byte olarak boyutu
8	4	LocalVar SigTok	Metod için yerel değişkenlerin planını tanımlayan imza için metadata tokeni. 0 ın anlamı herhangi bir yerel değişken mevcut değildir.

1.4. Dosya İşlemlerinin Algılanması ve Metoda Yedekleme Kodu Eklenmesi

Bu bölümde program dosyalarına nasıl müdahale edeceğimizi, programın akışını nasıl değiştireceğimizi inceleyeceğiz.

Örnek olarak ise program içerisinde dosya işlemlerini analiz edecek bir program geliştirilmiştir. Bu program imaj dosyasındaki dosya işlemlerini aramaktadır ve eğer dosya yazılmak için açılmışsa bir uyarı vermektedir. Eğer bu dosya program çalıştığında yedeklemek isteniyorsa programa yedekleme kodunu eklenir. Anlaşılacağı üzere işlemler iki aşamadan oluşmaktadır: birinci aşama dosya işleminin algılanması, ikinci aşama ise exe dosyasına, yedekleme işlemini yapabilecek yeni kodlar ekleme.

1.4.1. Dosya İşlemlerinin Algılanması

Bu işlem için öncelikle aşağıdaki verilen C# ve bunun karşılığı olan il kodunu inceleyelim:

```
static void Main(string[] args)
{
string dosya = "deneme.txt";
FileStream fs = new FileStream(dosya, FileMode.Open, FileAccess.ReadWrite);
fs.Close();
}                                                                    (1.6)
```

(1.6)'daki kodda da görüldüğü gibi deneme.txt isimli bir dosya açılıyor Bu kodun karşılığı olan IL koduna bakalım şimdide:

```
.method /*06000001*/ private hidebysig static
      void  Main(string[] args) cil managed
// SIG: 00 01 01 1D 0E
{
 // Method 0x2050 RVA da başlıyor
 // Kod Boyutu       24 (0x18)
 .maxstack  4
 .locals /*11000001*/ init (string V_0,
 class [mscorlib/* 23000001 */]System.IO.FileStream/* 01000012 */ V_1)
 IL_0000:  /* 00   |                */ nop
 IL_0001:  /* 72   | (70)000001       */ ldstr      "deneme.txt" /* 70000001} */
 IL_0006:  /* 0A   |                */ stloc.0
 IL_0007:  /* 06   |                */ ldloc.0
 IL_0008:  /* 19   |                */ ldc.i4.3
 IL_0009:  /* 19   |                */ ldc.i4.3
 IL_000a:  /* 73   |    newobj     instance void System.IO.FileStreamctor(string,
System.IO.FileMode,System.IO.FileAccess)
 IL_000f:  /* 0B   |                */ stloc.1
 IL_0010:  /* 07   |                */ ldloc.1
 IL_0011:  /* 6F   |    callvirt   instance voidSystem.IO.Stream::Close()
 IL_0016:  /* 00   |                */ nop
 IL_0017:  /* 2A   |                */ ret
} // Metod Sonu Program::Main                                                    (1.7)
```

(1.7)'daki IL kodunda da görüldüğü gibi dosya işlemi newobj komutuyla başlıyor, ve sonraki aşamada ise :

instance void [mscorlib/* 23000001 */]System.IO.FileStream/* 01000017 */::.ctor(string, System.IO.FileMode, System.IO.FileAccess) parametresini alıyor.

Bölüm Image_Cor_Ilmethod_Fat'tada belirtildiği gibi metod tablosundaki RVA adresine atlanarak fat format yada tiny formata göre metot başlığı verileri alınır. Metod

başlığı bilgileri alındıktan sonra kod boyutunda belirtildiği byte kadar metod gövdesi kodu okunur. Son aşamada bu byte dizisindeki kodlar çözülür.

Örneğin 0 değeri nop operandını gösterir. Bu emrin anlamı herhangi bir işlem olmadığıdır ve emir boyutu sadece tek byte'dır.

114 değeri ldstr komutunu gösterir. Bu komutun türü OperandType.InlineString'tir. Bu operand türü kendinden sonra kod dizisinde 4 byte'lık bir değer alır. Bu 4 byte'lık değerin en anlamsız 3 byte'ı, us (user string) akımından kaç byte'lık değer alınacağını ve başlangıç indeksini gösterir. Önceki bölümlerde de açıklandığı gibi user string akımı kullanıcı tanımlı stringleri tutar.

10 değeri stloc.0 komutunu gösterir. Bu komutun türü OperandType.InlineNone'dır. Bu operand türü kendinden sonra herhangi bir değer almaz. Ve işlem yığınındaki en üstteki değeri stacktaki 0 adresine atar. Diğer bir ifadeyle değişken tanımlamaları için kullanılır. Herhangi bir değişken tanımlanırsa stloc emri ile stağa atılır ve çağrılacağı zaman ldloc emriyle çağrılır.

6 değeri ldloc.0 komutunu gösterir. Bu komutun türü OperandType.InlineNone'dır. Bu komutta kendinden sonra herhangi bir değer almaz. Ve stackta ki sıfırıncı elemanı işlem yığınına yükler (ldloc.1 olsaydı birinci elemanı yükleyecekti).

26 değeri ldc.I4.4 komutunu gösterir. Bu komutun türü OperandType.None'dır. Bu komut kendinden sonra bir değer almaz. Ve işlem yığınına 4 sabitini iter.

25 emri ldc.I4.3 komutunu gösterir. Bu komutun türü OperandType.None'dır. Bu komut kendinden sonra bir değer almaz. Ve işlem yığınına 3 sabitini iter.

115 değeri newobj komutunu gösterir. Bu komutun türü OperandType.InlineMethod'tur. Bu komut kendinden sonra 4 byte değer okur. İlk byte hangi tablonun kullanılacağını gösterir. Kalan 3 byte ise token'i gösterir. Örnek programda newobj den sonra System.IO.FileStream::.ctor fonksiyonu çağrılmıştır. Bu fonksiyon MemberRef (MethodRef) tablosunda saklanır. Tablonunda sırası 10'dur. Bu yüzden dosya işlem analizinin yapılacağı programlar debug edilince tablo değeri 10 çıkar. (Eğer tablo değeri 1 ise TypeRef tablosuna, 4 ise FieldStruct tablosuna gidilir.). MemberRef tablosunun içeriği daha önceki bölümlerde açıklanmıştır. Ama yine bahsedilecek olursa, tablodaki ilk 2 byte referans alınacak metodun hangi class'a ve namespace'e ait olduğunu belirtir. Yine daha önceden açıklandığı gibi class değerinin ilk 3 biti hangi tablodan değer alınacağını gösterir. Dosya işlemi algılanacak programlarda bir tür refreransı alındığı için class alanının ilk 3 bitinin değeri 1 olarak elde edildi.

(System.IO.FileStream türünden bir değer alındığı için). Class alanının kalan bitleri ise bulunan tabloda hangi satırdan değer alınacağını gösterir. Örnek program için elde edilen değer 23'tür. TypeRef tablosunun 23. satırına gidildiğinde alınan sonuç ise System.IO.FileStream'tir. Sonraki aşamada ise hangi fonksiyonun referans olarak kullanıldığı bulunmalıdır. Bu işlem için daha önceden bulunan memberRef tablosuna gidilir ve token ile belirtilen satırındaki name ismi string akımından alınır. Örnek fonksiyon için bu sonuç değeri .ctor dur.

Yukarıdaki programda kullanılan bir referans metodun nasıl çözüldüğü gösterildi. Dosyaya yazıldığını gösteren geri kalan son aşama ise elde edilen metodun parametrelerinin bulunmasıdır. Bu parametreler blob akımından alınır. Blob akımı metadata başlığı ilk okunduğunda string dizisine dönüştürülür. Bu diziden token'inci eleman alınarak metodun kullandığı parametrelere ulaşılır. Parametrelerden FileAccess kontrol edilir. Eğer Write, veya ReadWrite değerlerinden birini almışsa dosya yazma için açılmış olarak yorumlanır ve istenirse sonraki aşamada yedekleme kodu eklenir.

Bu parametrelerin değerleri ise daha önceden işlem yığınına yüklenmişti. Örnek program için bu parametreler sırasıyla string, FileMode ve FileAccess'tir. Bunları daha önce anlatıldığı gibi işlem yığınına sırasıyla ldstr,stloc.0,ldloc.0, ldc.I4.4, ldc.I4.3 emirleriyle atılmıştı. Burada ldc.I4.3 FileAcess değerini gösterir. Örnek program için bu ReadWrite'dır.

Bu değerlendirmeler imaj dosyasındaki tüm metotlar için yapılır. Analiz işlemi tamamlandıktan sonra hangi metodun hangi emrinde dosya yazma işlemi içerdiğine dair veriler elde edilir.

Tüm bu değerler imaj dosyasındaki tüm metotlar için yapılır. Analiz işlemi bittikten sonra elimizde hangi metodun hangi emrinde dosya yazma işlemi var olduğuna dair değerler elde edilir.

1.4.2. Dosyaya Yedekleme Kodunun Eklenmesi

İlk bölümde dosya işlemlerinin nasıl algılandığı anlatıldı. Bu bölümde ise exeye nasıl yeni kod ekleneceği gösterilmiştir. Aşağıda yedekleme kodu eklenmiş fonksiyonun C# ve il kodları görünmektedir.

```
static void Main(string[] args)
{
   string dosya = "deneme.txt";
   FileStream fs = new FileStream(dosya, FileMode.Open, FileAccess.ReadWrite);
   File.Copy(fs.Name,"deneme2.txt");
   fs.Close();
}
```
(1.8)

(1.8)'de belirtilen fonksiyonda görüldüğü gibi fonksiyon koduna yeni bir fonksiyon çağrısı eklendi. File.Copy(fs.Name,"deneme2.txt"); fs ile açılan dosya deneme2.txt (dosya adı değişken olarak ekleniyor) dosyasına kopyalanmaktadır. Aşağıda ise yukarıdaki fonksiyonun il kodu görülmektedir.

```
.method /*06000001*/ private hidebysig static
        void  Main(string[] args) cil managed
  // SIG: 00 01 01 1D 0E
  {
   // Method  0x2050 RVA da başlıyor
   // Kod Boyutu      41 (0x29)
   .maxstack  4
   .locals /*11000001*/ init (string V_0,
           class [mscorlib/* 23000001 */]System.IO.FileStream/* 01000012 */ V_1)
   IL_0000:  /* 00   |                */ nop
   IL_0001:  /* 72   | (70)000001       */ ldstr      "deneme.txt" /* 70000001} */
   IL_0006:  /* 0A   |                */ stloc.0
   IL_0007:  /* 06   |                */ ldloc.0
   IL_0008:  /* 19   |                */ ldc.i4.3
   IL_0009:  /* 19   |                */ ldc.i4.3
   IL_000a:  /* 73   |    IL_000a:  /* 73   | (0A)000010       */ newobj     instance void
[mscorlib/* 23000001 */]System.IO.FileStream/* 01000012 */::.ctor(string,

valuetype [mscorlib/* 23000001 */]System.IO.FileMode/* 01000013 */,

valuetype [mscorlib/* 23000001 */]System.IO.FileAccess/* 01000014 */) /* 0A000010 */
   IL_000f:  /* 0B   |                */ stloc.1
   IL_0010:  /* 07   |                */ ldloc.1
   IL_0011:  /* 6F   |    IL_0011:  /* 6F   | (0A)000011       */ callvirt   instance
string [mscorlib/* 23000001 */]System.IO.FileStream/* 01000012 */::get_Name() /*
0A000011 */
   IL_0016:  /* 72   | (70)000017       */ ldstr      "deneme2.txt" /* 70000017} */
   IL_001b:  /* 28   |    IL_001b:  /* 28   | (0A)000012       */ call       void
[mscorlib/* 23000001 */]System.IO.File/* 01000015 */::Copy(string, string)
   IL_0020:  /* 00   |                */ nop
```

```
        IL_0021:  /* 07   |                */ ldloc.1
        IL_0022:  /* 6F   |    IL_0022:  /* 6F   | (0A)000013       */ callvirt   instance void
[mscorlib/* 23000001 */]System.IO.Stream/* 01000016 */::Close() /* 0A000013 */
        IL_0027:  /* 00   |                */ nop
        IL_0028:  /* 2A   |                */ ret
      } // Metod Sonu Program::Main                                                    (1.9)
```

(1.9)'deki fonksiyonda da görüldüğü gibi fonksiyona newobj emrinden sonra sırasıyla ldloc.1 , callvirt, ldstr,call emirleri eklenmiştir.

Bu emirler aşağıda belirtilen işlemleri yapar.

Ldloc.1 emri, stağa stloc.1 emriyle yüklenen değeri başa alır. (ldloc.1 ile daha önceden de hatırlanacağı gibi stağa newobj System.IO.FileStream::.ctor atılmıştı). İkinci olarak callvirt emriyle yeni stloc.1 deki FileStream'ın ismini get_name fonksiyonuyla alarak yığının tepesine koyar. Üçüncü fonksiyonda ise ldstr emriyle yığına bir string atılır. Son emir call ile System.IO.File::Copy(string,string) fonksiyonu çağrılır. Bu fonksiyon stağın tepesindeki ilk iki değeri alır ve ilk değerdeki stringle belirtilen dosyayı ikinci değerdeki stringle belirtilen fonksiyona kopyalar.

Sırasıyla bu emirlerin türlerini ve imaj dosyasına eklenmesi için neler yapılması gerektiğini inceleyelim:

İlk emir ldloc.1 komutu OperandType.InlineNone türündendir. Kendinden sonra bir değer almaz ve 1 byte'lık alan kaplar. Metod gövdesine eklendiğinde metodun kod boyutu sadece 1 artar. Anlaşıldığı gibi bu komut metoda eklendiğinde metot başlığındaki kod boyutu alanındaki değer bir arttırılır.

2. emir callvirt komutu OperandType.InlineMethod türündedir. Bu emir kendinden sonra 4 byte'lık değer alır. Bu 4 byte'ın ilk byte'ı get_Name() metodunun olduğu tablo olan MemberRef (MethodRef) tablosunu işaret eder. Kalan 3 byte ise token'i gösterir. MemberRef tablosunun token'inci satırına gidildiğinde class alanına bakılır. Bu alan metodun sınıfını ya da namespace'ini gösterir. Class ın ilk 3 biti ilgili tabloyu gösterir. Örnek fonksiyon için bu tablo TypeRef tablosudur. Kalan bitler ise bu tablodaki indeks değeridir. Sonuç olarak "System.IO.FileStream" değeri elde edilir. MemberRef tablosunun token'inci satırının name alanında ise get_name değeri elde edilir. Bu da System.IO.FileStream namespace'indeki get_Name() fonksiyonuna işaret eder.

Yukarıda da anlatıldığı gibi bu emrin boyutu 5 byte'dır. Bu emir metod gövdesine eklenirken, öncelikle MemberRef tablosuna bir satır eklenir. Bu satır get_Name()

fonksiyonunu tanımlar. Ayrıca metod başlığındaki kod alanı 5 arttırılır. Metod gövdesinde ise ilgili emir sırasına ilk önce 111 eklenir (111 calvirt emrini tanımlar). Sonraki aşamada ise bu emrin alacağı 4 byte'lık değer eklenir. Bu 4 byte'ın ilk byte'nın değeri 10'dur. 10 MemberRef tablosunu gösterir. Kalan 3 byte ise metodun memberRef tablosundaki satır numarasını gösterir. Elde edilen tablo ve indeks değeri birleştirilerek 4 byte olarak dosyaya yazılır.

Sıradaki emir ise ldstr emridir. Bu emirde kendinden sonra 4 byte'lık değer alır. Son 3 byte us tablosundan stringi almak için gerekli olan değerdir. Bu emir için önce us akımına ilgili string eklenir. Sonra ise 114 (ldstr) değeri metod gövdesine yazılır. Son olarak ise 4 byte stringi alabilmemiz için gerekli olan değer metod gövdesine eklenir.

Sonraki son emir ise call emridir. Bu emrin de türü OperandType.InlineMethod 'tur ve kendinden sonra 4 byte'lık değer alır. İlk byte tabloyu gösterir. Kalan 3 byte ise ilgili tablodaki satır değerini gösterir. Yine burada da bir fonksiyon çağırıldığı için tablonun değeri 10'dur ve metotRef tablosunu gösterir. get_name() fonksiyonuyla benzer bir yapısı vardır.

Yukarıda sadece metodun gövdesinin nasıl değiştirildiği ve tablolara yeni satırların neden eklendiği anlatıldı. Sonraki aşamada ise tüm metotların RVA larının yeniden hesaplanması ve metod tablosunda ilgili RVA değerlerinin yeniden yazılması gelmektedir. Bunun nedeni ise tablolara yeni satırlar eklenmesi, us akımına yeni değer eklenmesi ve metod gövdelerine yeni kodlar eklenmesidir. Yeni RVA değerleri her fonksiyon için yeniden hesaplanır.

Son olarak ise imaj başlığındaki section başlıklarının RVA'ları yeniden hesaplanır. Ayrıca dataDirectorydeki veri dizinlerinin de RVA'ları yeniden hesaplanır.

Tüm bu elde edilen değerlerin sonuçları dosyaya yeniden yazılarak, exe dosyasında değişiklik yapılmış olur.

2. YAPILAN ÇALIŞMALAR VE BULGULAR

2.1. Giriş

Çalışmada .NET ile yazılmış program exelerinin ve dll'lerin metotlarının, metot parametrelerinin, değişkenlerinin, sabitlerinin, metod içeriklerinin, sınıflarının, namespace'lerinin, import tablosunda kullanılan dll'lerin, bu dll'lerde hangi fonksiyonların kullanıldığının, dosyanın başlık bilgilerinin (COFF , CLR), metadata tablo bilgilerinin elde edilmesi gerçekleştirilmeye çalışılmıştır. Ayrıca imaj dosyasına dosya işlemleri için dosya işlemlerinin analizini ve yedeklemesini gerçekleştirebilecek yapılar eklenmiştir.

2.2. Başlık Bilgilerinin Elde Edilmesi

2.2.1. DOS Başlığı Bilgileri

.NET dosyaları birer PE (Portable Executable) dosyalarıdır. Bu dosyalar ilk iki byte'ında 0x5a4d "MZ" değerini taşımak zorundadırlar. Daha sonra gelen 0x3c offsetindeki 4 byte ise bu dosyanın PE flagının yerini belirtir. Buradan okunan 4 byte'ın verdiği integer değerdeki offset'den 4 byte okunduğunda 'PE\0\0' yani 0x00004550 değerinin alınması gerekir.

```
file_name = opf.FileName;
stream = new FileStream(opf.FileName, FileMode.Open, FileAccess.ReadWrite);
mbinaryreader = new BinaryReader(stream);
IMAGE_DOS_HEADER dos_header = new IMAGE_DOS_HEADER(stream);
if (dos_header.e_magic != (int)HeaderSignatures.IMAGE_DOS_SIGNATURE)
        {
           MessageBox.Show("Uygun Formatta Bir Dosya Değil");
           return false;
        }
        stream.Seek(dos_header.e_lfanew, SeekOrigin.Begin);          (2.1)
```

(2.1)'deki kodda ilk önce FileStream ile bir exe dosyası açılıyor. dos_header yapısı ile bu exe imajının dos başlığı okunuyor. Daha sonra dos başlığındaki e_magic numarasına bakılıp bu dosyanın exe dosyası olup olmadığı kontrol edilir. Son adımda ise dos başlığındaki e_lfanew değerindeki offsete atlanır.

Tablo 43. Örnek Dos başlığı

dos_header	
e_cblp	144
e_cp	3
e_cparhdr	4
e_crlc	0
e_cs	0
e_csum	0
e_ip	0
e_lfanew	128
e_lfarlc	64
e_magic	23117
e_maxalloc	65535
e_minalloc	0
e_oemid	0
e_oeminfo	0
e_ovno	0
e_res	0x0398e374
e_res2	0x0398e380
e_sp	184
e_ss	0

Tablo 43'te Ek 14'te verilen örnek bir programdan alınan dos başlığında e_magic değeri 23117 short integer değerine eşittir. Byte olarak incelendiğinde "MZ" değerine karşılık gelen bu değer ilgili dosyanın dos başlığına sahip olduğunu gösterir. "lfanew" alanında ise 128 değeri olduğu görülüyor. Dosya PE dosyası ise bu değerin işaret ettiği adreste PE imzası bulunur.

2.2.2. NT Başlığı Bilgileri

Dos başlığının e_lfanew değerindeki offset'ten nt_header okunur. Dosyanın PE dosyası olduğunu belirten *nt_header.Signature* değerinin "PE\0\0" olup olmadığı kontrol edilir. Bu kontroller yapıldıktan sonra dosyanın PE dosyası olup olmadığı anlaşılır.

NT başlığı kendi içinde üç bölüme ayrılır. İlk bölüm yukarıda anlatılan PE imzasıdır. 2. bölüm ise bölüm 1.3.2.4'te ayrıntılı olarak anlatılan dosya başlığıdır. Son bölüm ise bölüm 1.3.2.5'te ayrıntılı olarak anlatılan opsiyonel başlıktır.

Tablo 44'te Ek 14'te verilen örnek bir exe dosyasından alınan dosya başlığı (FileHeader) görülüyor. Burada opsiyonel başlığın boyutu 224 byte'dır.

Tablo 44. Örnek bir File Header

FileHeader	
Characteristics	270
Machine	332
NumberOfSections	3
NumberOfSymbols	0
PointerToSymbolTable	0
SizeOfOptionalHeader	224
TimeDateStamp	1170227195

Tablo 44'te Machine alanındaki 332 değeri "0000014C"'a karşılık gelir. 1.3.2.4.1'de anlatılan *Makine Türleri* tablosunda bu değerin anlamı intel 386 veya daha sonra çıkan veya uyumlu olan işlemcilerdir.

Charecteristics alanında bulunan 270 değeri ise "0000010E"'a karşılık gelir. 1.3.2.4.2'de anlatılan karakteristik tablosunda bu değerin karşılığında Windows 2000 ve sonrası için önemsenmez ve 0'a setlenir değeri vardır.

NumberOfSections ise 3'tür. Bu değer dosyada 3 tane bölüm olduğunu gösterir. Tablo 45'te örnek programdan alınmış bölüm başlıkları görülmektedir.

Opsiyonel başlığın boyutunu gösteren SizeOfOptionalHeader alanının değerinde ise 224 vardır.

Tablo 45. Örnek bir bölüm başlığı

Section[0]	
Characteristics	1610612768
Misc	5428
Name	.text (Executable Code)
NumberOfLinenumbers	0
NumberOfRelocations	0
PointerToLinenumbers	0
PointerToRawData	4096
PointerToRelocations	0
SizeOfRawData	8192
VirtualAddress	8192
Section[1]	
Characteristics	1073741888
Misc	912
Name	.rsrc (Resource Directory)
NumberOfLinenumbers	0
NumberOfRelocations	0
PointerToLinenumbers	0
PointerToRawData	12288
PointerToRelocations	0
SizeOfRawData	4096
VirtualAddress	16384
Section[2]	
Characteristics	1107296320
Misc	12
Name	.reloc (Image Relocation)
NumberOfLinenumbers	0
NumberOfRelocations	0
PointerToLinenumbers	0
PointerToRawData	16384
PointerToRelocations	0
SizeOfRawData	4096
VirtualAddress	24576

Tablo 46'da Ek 14'te verilen örnek bir exe dosyasından alınan opsiyonel başlık bilgileri görülmektedir. Bu başlıkta bizim için en önemli alan VirtualAddress alanıdır. VirtualAddress bölümün sanal adrsini gösterir. Bu adrese atlanıldığında bölüm bilgileri elde edilir.

Bu başlıklardaki alanlar 1.3.2.5'te ayrıntılı olarak anlatıldığı için bu bölümde yeniden ayrıntısına girilmemiştir.

Tablo 46. Örnek bir opsiyonel başlık

AddressOfEntryPoint	13614
BaseOfCode	8192
BaseOfData	16384
CheckSum	0
VirtualAddress	0
DataDirectory13	
Size	8
VirtualAddress	8192
DataDirectory14	
Size	0
VirtualAddress	0
DataDirectory15	
Size	72
VirtualAddress	8200
DataDirectory2	
Size	83
VirtualAddress	13528
DataDirectory3	
DllCharacteristics	1024
FileAlignment	4096
ImageBase	4194304
LoaderFlags	0
Magic	267
MajorImageVersion	0
MajorLinkerVersion	8
MajorOperatingSystemVersion	4
MajorSubsystemVersion	4
MinorImageVersion	0
MinorLinkerVersion	0
MinorOperatingSystemVersion	0
MinorSubsystemVersion	0
NumberOfRvaAndSizes	16
SectionAlignment	8192
SizeOfCode	8192
SizeOfHeaders	4096
SizeOfHeapCommit	4096
SizeOfHeapReserve	1048576
SizeOfImage	32768
SizeOfInitializedData	8192
SizeOfStackCommit	4096
SizeOfStackReserve	1048576
SizeOfUninitializedData	0
Subsystem	2
Win32VersionValue	0

AddressOfEntryPoint değerinde başlangıç fonksiyonunun adresi (örneklerimizde Main fonksiyonunun adresi) vardır. Magic alanında bulunan 267 değeri "0000010B" verisine karşılık gelir ve normal icra edilebilir dosya olduğunu gösterir.

Subsystem alanında 2 değeri vardır. Bu imaj dosyasının Windows grafiksel kullanıcı ara yüzü alt sistemi (GUI)'ni kullandığını gösterir.

DLL karakteristiği alanındaki 1024 değeri "00000400" verisine karşılık gelir ve DLL yükleme zamanında yerleştirilebilir anlamını taşır.

Magic değeri imaj dosyasının durumunu tanımlar ve 0x10B hexal karşılığına gelen 267 değerindedir. Ek Tablo 5'te açıklandığı gibi 0x10B değeri normal çalıştırılabilir dosyayı gösterir.

2.3. CLR Bölümü Verileri

CLIHeader DataDirectory 15 ile gösterilir. Eğer 15. veri dizininin sanal adresi varsa, bu sanal adresin gösterdiği RVA değeri hesaplanarak dosya konumlandırıcı buraya getirilir.

```
if (nt_header.OptionalHeader.DataDirectory15.Size == 0)
        {
          MessageBox.Show("Uygun Formatta Bir Dosya Değil");
          return false;
        }
    dwclrDirectory = RVA2Offset(stream,
Convert.ToInt32(nt_header.OptionalHeader.DataDirectory15.VirtualAddress));
        if (dwclrDirectory == 0)
        {
          MessageBox.Show("Uygun Formatta Bir Dosya Değil");
          return false;
        }
        stream.Seek(dwclrDirectory, SeekOrigin.Begin);                    (2.2)
```

(2.2)'deki kodda görüldüğü gibi opsiyonel başlıktaki 15. datadirectory'e bakılır. Eğer 0 ise bu dosya uygun formatta değildir. Diğer durumda RVA'sı hesaplanır ve bu RVA değerinin 0 dan farklı olup olmadığı kontrol edilir. Eğer 0'dan farklıysa dosya konumlandırıcı bu RVA değerine atlatılarak CLR başlığına gelinir. Sonraki aşamada ise

CLR başlığı okunarak metadata bilgilerinin alınması için metadata bölümüne atlanır. Örnek kod (2.3)'te verildiği gibidir:

```
clrheaders=new IMAGE_COR20_HEADER(fs);
this.startofmetadata = RVA2Offset(fs,
Convert.ToInt32(clrheaders.MetaData.VirtualAddress));
 fs.Seek(startofmetadata, SeekOrigin.Begin);                    (2.3)
```

Tablo 47'de Ek 14'te verilen örnek bir exe dosyasından alınan CLR başlığı değerleri görülmektedir:

Tablo 47. Örnek bir CLR başlığı

cb	72
CodeManagerTable	
Size	0
VirtualAddress	0
DebugMap	
Size	143379
VirtualAddress	704643078
DelayLoadInfo	
Size	4904
VirtualAddress	100663298
DynamicInfo	
Size	1929382400
VirtualAddress	1189910
EEInfoTable	
Size	0
VirtualAddress	0
EntryPointToken	100663297
ExportAddressTableJumps	
Size	0
VirtualAddress	0
ExternalFixups	
Size	5160
VirtualAddress	33816576
Flags	1
HelperTable	
Size	655360
VirtualAddress	287834218
IPMap	
Size	285212673

Tablo 47. Örnek bir CLR başlığı'nın devamı

VirtualAddress	43
MajorRuntimeVersion	2
MetaData	
Size	4188
VirtualAddress	9160
MinorRuntimeVersion	5
ModuleImage	
Size	24974338
VirtualAddress	1646919690
Resources	
Size	184
VirtualAddress	8976
RidMap	
Size	1064
VirtualAddress	33554442
StrongNameSignature	
Size	0
VirtualAddress	0
VTableFixups	
Size	0
VirtualAddress	0

Burada cb değeri başlığın byte olarak boyutunu gösterir ve değeri 72 byte'dır.

MajorRuntimeVersion alanındaki değer 2'dir. MinorRuntimeVersion alanındaki değer 5'tir. Bu programın çalışması için gerekli runtime versiyon numarasını gösterir.

Metadata alanındaki *size* değeri metadatanın boyutunu gösterir. Virtual adres değeri ise metadatanın CLR başlığına göre konumunu gösterir. Aynı metadata Resource kaynak bölümünün CLR başlığına göre konumunu ve boyutunu, EntryPointToken ise başlangıç fonksiyonunun (bizim örneklerimiz için main fonksiyonu) adresini gösterir.

Flags alanındaki 1 değerinin anlamı, imaj sadece IL kodu içerir. Bu yüzden herhangi özel bir işlemcide koşmasına gerek yoktur.

CLR başlığındaki alanların detaylı anlatımı 1.3.3.1 Koşma Zamanı Başlığı adlı bölümde yapılmıştır.

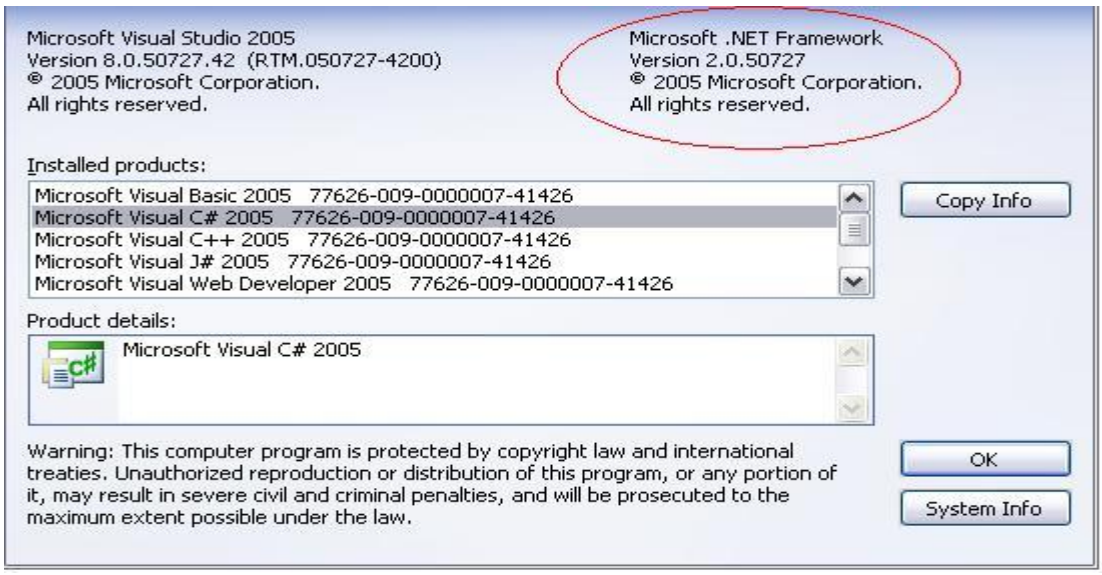
2.3.1. MetaData Bölümü Verileri

CLR başlığın sonra metadata başlığı okunur. Tablo 48'de Ek 14'te verilen örnek bir program için metadata başlık bilgileri verilmiştir:

Tablo 48. Metadata başlık bilgileri

bscb	1112167234
MetadataMajorVersion	1
MetadataMinorVersion	1
versiyon	{Dimensions:[12]} (v2.0.50727)
padding	0
flag	0
stream	5

Tablo 48'de de görüldüğü gibi örnek program 5 akımdan oluşuyor (daha önceden de bahsedildiği gibi akım sayısı sabit değildir, maksimum 5 tane olabilir). Versiyon'da ise .NET framework'un versiyon numarası yer alır. Örneğimiz için bu değer v2.0.50727'dir. Aşağıdaki şekilde ise bir bilgisayarda kurulu olan Visual Studio.net'in versiyon numarası görülüyor.



Şekil 8. Framework versiyon numarası

Şekil 8'de de görüldüğü gibi Tablo 48'teki imaj dosyasında okuduğumuz değer buradaki değere eşit oldu.

(2.4)'te verilen kod ile hangi metadata tablolarının var olduğu ve var olan metadata tablolarının satır sayıları bulunur.

```
valid = BitConverter.ToInt64(metadata, 8);
        for (int k = 0; k <= 63; k++)
        {
           int tablepresent = (int)(valid >> k) & 1;
           if (tablepresent == 1)
           {
              rows[k] = BitConverter.ToInt32(metadata, tableoffset);
              tableoffset += 4;
           }
        }                                                              (2.4)
```

(2.4)'teki kodda metadata başlığındaki valid alanının bitleri tek tek kontrol edilir. Her bit bir metadata tablosunu temsil etmektedir. Eğer ilgili bit bire eşitse, o bitle temsil edilen metadata tablosu vardır anlamındadır.

Metadata başlığı okunduktan sonra ise akım başlığı bölümüne gelinir. Tablo 49'da Ek 14'te verilen örnek program için akım başlıklarının aldığı değerler vardır.

Tablo 49. Metadata akımları ve verileri

Akımlar	
[0]	
Name	#~
offset	108
Size	1484
[1]	
Name	#Strings
offset	1592
Size	1788
[2]	
Name	#US
offset	3380
Size	136
[3]	
Name	#GUID
offset	3516

Tablo 49'un devamı

Size	16
[4]	
Name	#Blob
offset	3532
Size	656

Tablo 49'da görüldüğü gibi örnek program 5 akımdan oluşuyor. Tabloda sırasıyla her akımın ismi, offset değeri ve boyutu vardır. Akım bilgilerini aşağıdaki kod yardımıyla alabiliriz.

```
for (int i = 0; i < stream; i++)
{
   if (streams[i].Name == "#~" || streams[i].Name == "#-")
   {   metadata = new byte[streams[i].Size];
      fs.Seek(startofmetadata + streams[i].offset, SeekOrigin.Begin);
      for (int k = 0; k < streams[i].Size; k++)
         metadata[k] = mbinaryreader.ReadByte();
   }
   if (streams[i].Name == "#Strings")
   {   strings = new byte[streams[i].Size];
      fs.Seek(startofmetadata + streams[i].offset, SeekOrigin.Begin);
      for (int k = 0; k < streams[i].Size; k++)
         strings[k] = mbinaryreader.ReadByte();
   }
   if (streams[i].Name == "#US")
   {   us = new byte[streams[i].Size];
      fs.Seek(startofmetadata + streams[i].offset, SeekOrigin.Begin);
      for (int k = 0; k < streams[i].Size; k++)
         us[k] = mbinaryreader.ReadByte();
   }
   if (streams[i].Name == "#GUID")
   {  guid = new byte[streams[i].Size];
      fs.Seek(startofmetadata + streams[i].offset, SeekOrigin.Begin);
      for (int k = 0; k < streams[i].Size; k++)
         guid[k] = mbinaryreader.ReadByte();
   }
   if (streams[i].Name == "#Blob")
   {   blob = new byte[streams[i].Size];
      fs.Seek(startofmetadata + streams[i].offset, SeekOrigin.Begin);
      for (int k = 0; k < streams[i].Size; k++)
         blob[k] = mbinaryreader.ReadByte();
   }

}                                                                    (2.5)
```

Başlık bilgileri okunduktan sonra sırasıyla her başlığın offset değerine atlanarak akım boyutları kadar byte değeri dosyadan okunarak akım dizilerine aktarılır. Akım bilgileri okunduktan sonra son olarak metadata akımından (#~) metadata tablolarının bilgileri çıkartılır ve daha önceden anlatıldığı gibi ilgili tablolar doldurulur (Ayrıntılı bilgi için bölüm 1.3.3.2.3. Metadata Tablo Türleri bölümüne bakınız).

2.3.2. MetaData Tabloları

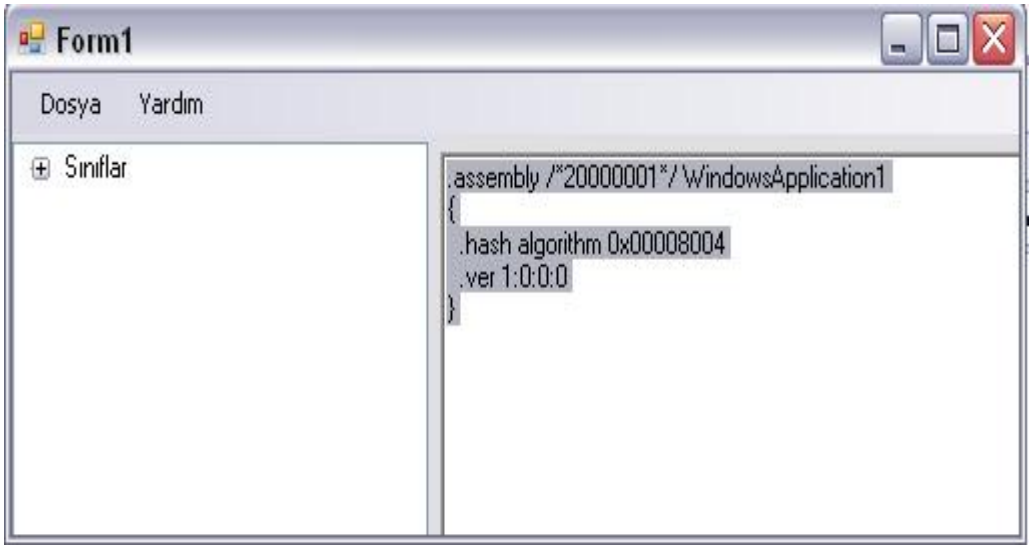
2.3.2.1. Assembly Tablosu Verileri

Tablo 50. Assembly tablosu sonucu

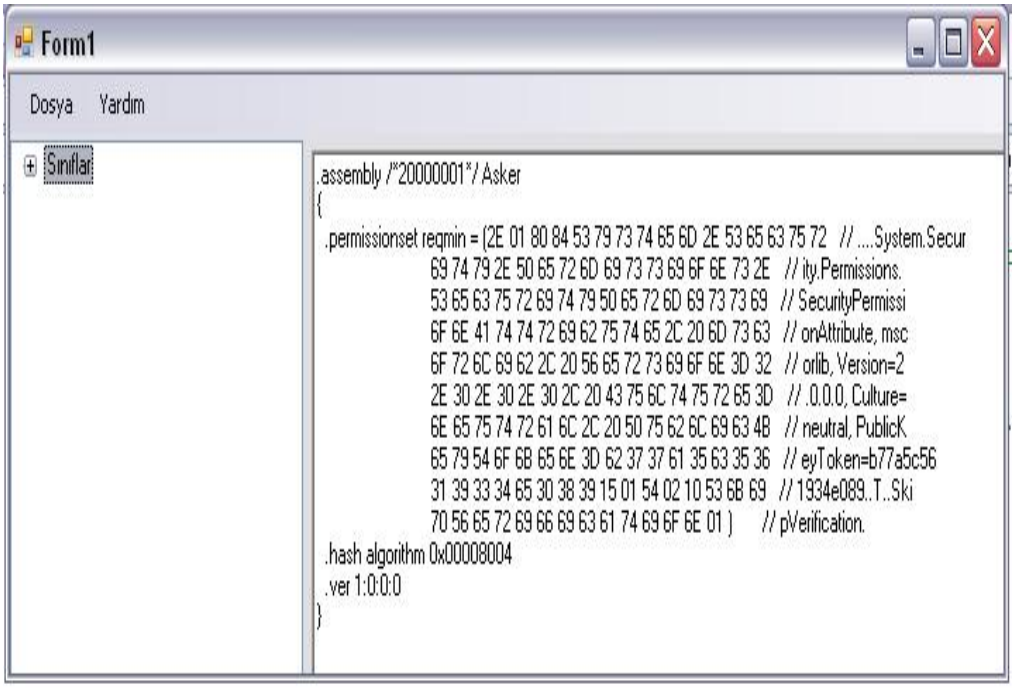
Assembly Tablosu	
build	0
culture	0
flags	0
HashAlgId	32772
major	1
minor	0
name	42 (WindowsApplication1)
publickey	0
revision	0

Major alanında assembly'nin asıl sürüm numarası vardır. name (isim) alanında assembly'nin ismi için string tablosuna bir indeks değeri vardır ve örnek program için bu değer 42'dir. String tablosunda ise elde edilen değer WindowsApplication1'dir. Tablodaki alanlar için ayrıntılı bilgi bölüm 1.3.3.2.3.7.'de anlatılmıştır.

Aşağıdaki şekillerde 2 ayrı programdan elde edilen assembly tablosu sonucu gösterilmiştir. Şekil 9 Tablo 50'de gösterilen programın assembly tablosu sonucuna aittir.



Şekil 9. Assembly tablosu 1. program sonucu



Şekil 10. Assembly tablosu 2. program sonucu

2.3.2.2. TypeDef Tablosu Verileri

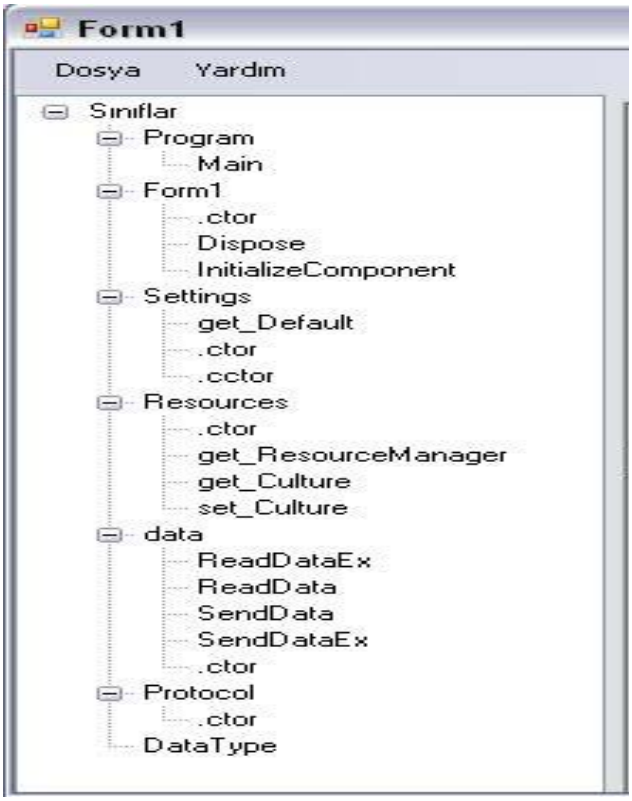
Aşağıda Tablo 51'de örnek program için typedef tablosundan alınan sonuç değerleri gösterilmiştir.

Tablo 51. TypeDef tablosu verileri

TypeDef Tablosu	{Dimensions:[9]}
[1]	{dosya_isimleri.TypeDefTable}
cindex	0
findex	1
flags	0
mindex	1 (Main, .ctor…)
name	1 (<Module>)
nspace	0
[2]	{dosya_isimleri.TypeDefTable}
cindex	5
findex	1 (components)
flags	1048960
mindex	1 (Main, .ctor…)
name	34 (Program)
nspace	42 (WindowsApplication1)
. . .	

Burada cindex değeri eğer sınıf türetilmiş bir sınıf ise hangi sınıftan türediğini gösterir. Mindex değeri sınıftaki metotları göstermek için metod tablosuna bir indeks değeridir. Nspace metodun hangi namespace'e ait olduğunu gösterir. *name* değeri türün adını göstermek için string kümesine bir indeks değeri içerir.

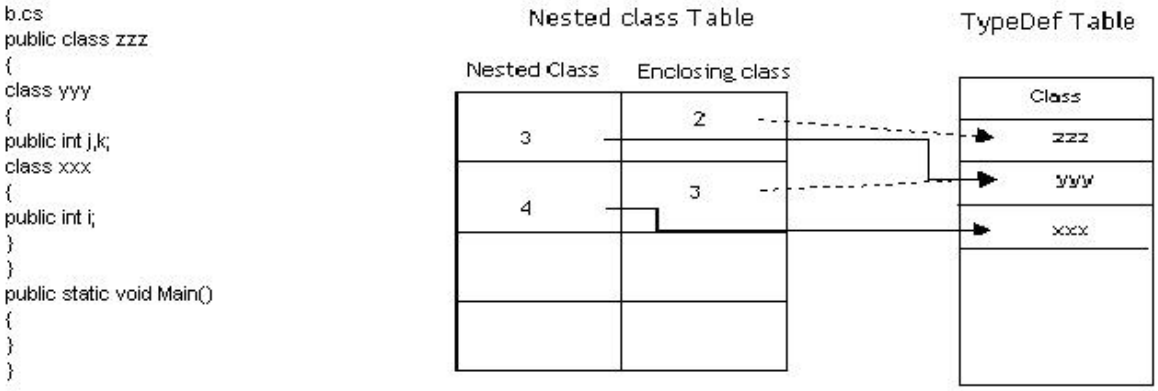
Şekil 11'de Ek 14'te verilen örnek program için elde edilen türler ve türler içindeki metotlar gösterilmiştir.



Şekil 11. Örnek programdaki sınıflar ve bu sınıflara ait olan fonksiyonlar

C# derleyicisiyle sınıf içerisinde sınıf oluşturulabilir. Bu durumda nestedClass tablosuna derleyici gerekli değerleri ekler.

TypeDef tablosuyla Nested Class tablosu arasındaki ilişki Şekil 12'da verilmiştir.



Şekil 12. NestedClass tablosuyla TypeDef tablosu arasındaki ilişki

2.3.2.3. Sabitler Tablosu Verileri

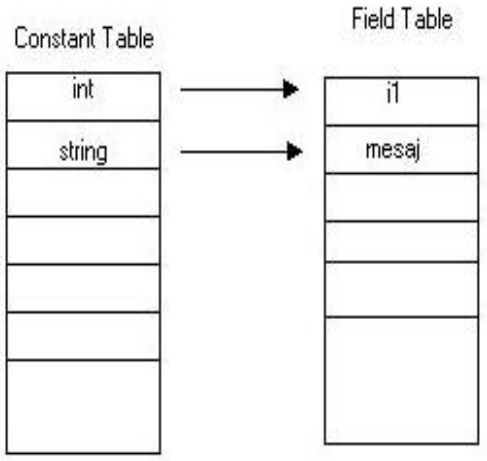
Tablo 52'de Ek 14'te verilen örnek program için sabitler tablosundan elde edilen değerlerin sonucu gösterilmiştir.

Tablo 52. Sabitler tablosu verileri

ConstantsStruct	
[1]	
dtype	8
parent	24
value	124
[2]	
dtype	8
parent	28
value	129
[3]	
dtype	8
parent	32
value	134
[4]	
dtype	8
parent	36
value	139
[5]	
dtype	8
parent	40
value	144
[6]	
dtype	8
parent	44
value	149
[7]	
dtype	8
parent	48
value	154

Constant aynı zamanda bir fieldtir. Uyan kayıt field tablosuna eklenir. Şekil 13'te örnek bir program için Canstant tablosuyla Field tablosu arasındaki ilişki verilmiştir.

```
b.cs
public class zzz
{
const int i1  = 20;
const string mesaj= "merhaba";
public static void Main()
{
}
}
```

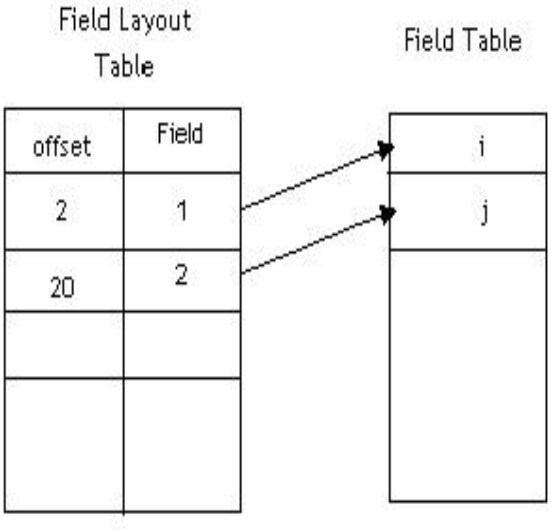


Şekil 13. Sabitler tablosu ile Alanlar tablosu arasındaki ilişki

2.3.2.4. Field Layout Tablosu Verileri

Şekil 14'te bölüm 1.3.3.2.3.16'da anlatılan FieldLayout tablosu için örnek bir program ve programın field layout tablosu ile Field tablosu arasındaki ilişki var.

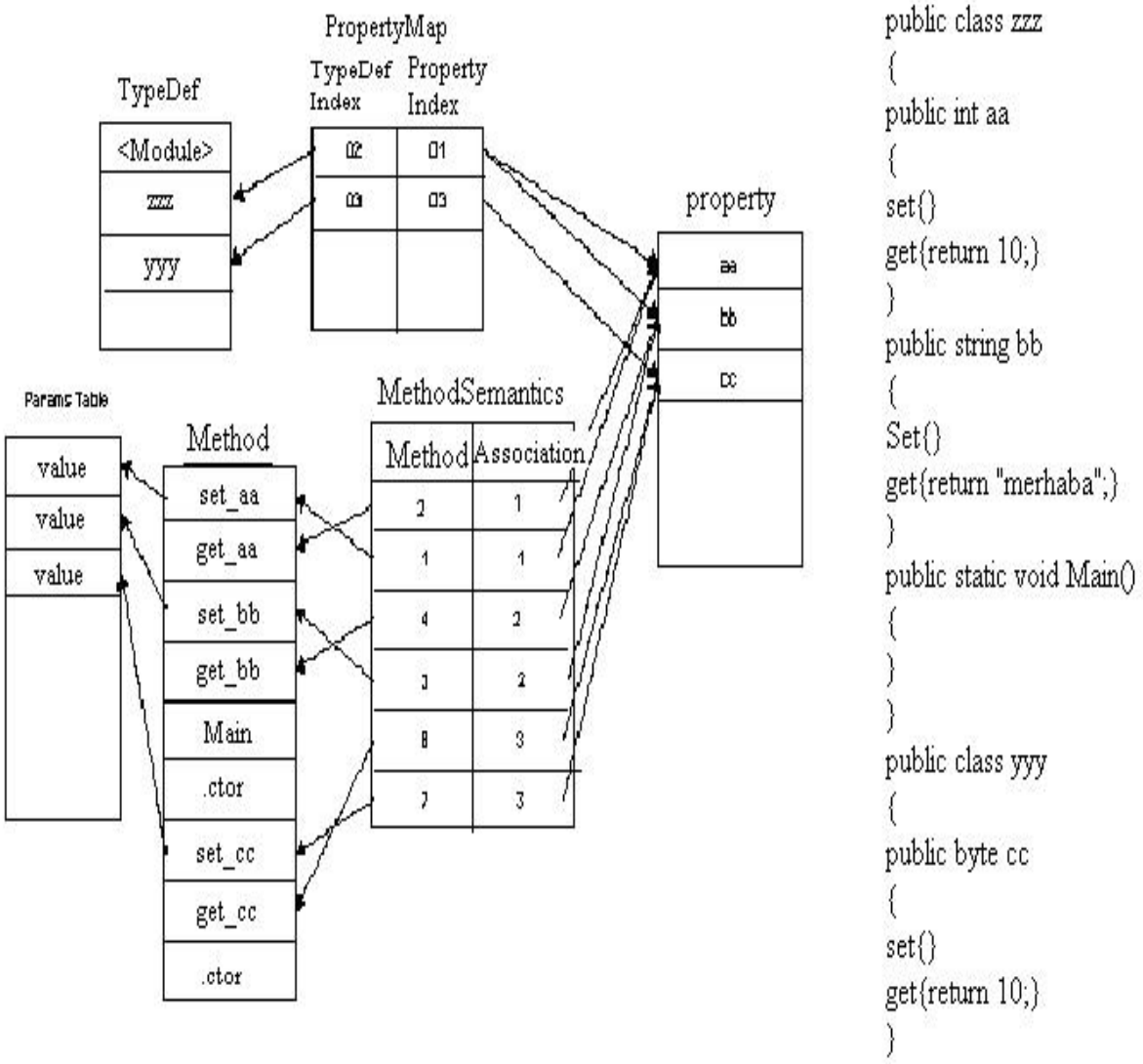
```
using System.Runtime.InteropServices;
[StructLayoutAttribute(LayoutKind.Explicit)]
public class zzz
{
[FieldOffset(2)] int i;
[FieldOffset(20)] long j;
public static void Main()
{
}
}
```



Şekil 14. FieldLayout ile Field tablosu arasındaki ilişki

2.3.2.5. Method Semantics Tablosu Verileri

Şekil 15'te örnek bir program için metod semantics tablosu ve bu tablodaki değerleri alabilmek için bu tablonun diğer tablolarla olan ilişkileri gösterilmiştir.



Şekil 15. MethodSemantics için Tablolar Arası İlişkiler

MethodSemantics tablosunun yapısını hatırlanacak olursa metod ismi metod tablosuna bir indeks değeriydi. Şekil 15'te görüldüğü gibi metod tablosuyla ilişkilendirilmiştir. Metod tablosu ise hatırlanacağı gibi param tablosuyla ilişkiliydi.

Metodsemantics tablosunun son elemanı Assocetion properties tablosuna bir indeks değeri içeriyordu.

2.4. Metodun Kodunun Geri Dönüşümü

Bu bölümde bir methodun geri dönüştürülmesi anlatılacaktır. Bir methodun geri dönüştürülebilmesi için o method'un Body'sini yani methodun iç kısmını okumamız gerekir. Bunun için Method'un RVA'sına gidilir ve okuma yapılır. Okuma işlemi methodun büyüklüğüne göre iki formatta olur. FatFormat (büyük methodlar, try catch veya local variable barındıran methodlar) ve Tiny format (Bu formatlar Bölüm 1.3 'te CLR başlığında detaylı anlatılmıştır). Tiny formatta method da sectionlar yoktur ve Code'lar buradan okunur. FatFormatta ise methodun kodu birden fazla section'a bölünür ve oradan sırası ile okunur. Aslında try catch kullandığınızda yada variable kullandığınızda method otomatikman FatFormatta olur. Çünkü TinyFormatta try catch ve Lacal Variable yoktur.

Method RVA'sını o RVA'yı kapsayan section'dan bulup o pozisyona gidilerek Method Body'si okunduğunda byte[] şeklinde bir data elde edilir. Okunan bu byte[] şeklindeki veri o methodun core datasıdır. Şimdi bunu işlemek gerekir. Aslında okunan bu byte[] tum IL datasıdır. Bunun için once IL'yi biraz inceleyelim. IL opcode'lardan oluşur, yani bir komut gibi düşünürsek bir işlemi ifade eden bir veri ve eğer varsa bunun aldığı bir data sonrasında da o data olacak sekilde dizilirler. IL'de her bir opcode'un bir Operand Type'ı vardır.

Bu Operand Türlerine örnek verecek olursak;

OperandType.InlineNone: Operand'dan sonra herhangi bir data olmaz. Sıradaki Operand'a geçilir.

OperandType.ShortInlineBrTarget: Operand'dan sonra bir sbyte'lık data hedef belirtir. Bu operand tipi branch yani zıplama operand'larında olur ve zıplamanın olacağı adresi belirtir. Yani goto... gibi düşünebilirsiniz.

OperandType.InlineBrTarget: Operand'dan sonra 4 byte lık bir integer data target belirtir. Bu operand tipi branch yani zıplama operandlarında olur ve zıplamanın olacağı adresi belirtir. Aynı ShortInlineBrTarget gibidir ancak tek fark gidilcek adres 4 byte'la ifade edilir.

OperandType.ShortInlineI: Operand'dan sonra bir sbyte'lık data vardır.

OperandType.InlineI: Operand'dan sonra 4 byte'lık bir integer data vardır

OperandType.InlineI8: Operand'dan sonra 8 byte'lık bir long data vardır

OperandType.ShortInlineR: Operand'dan sonra bir single data vardır

OperandType.InlineR: Operand'dan sonra bir double değer vardır.

OperandType.InlineString: Operand'dan sonra 4 byte'lık bir integer string'in US Heap deki indeksini belirtir. Bu indeksin önemli olan ilk 20 bit'i kullanılır. Yani (reader.ReadInt32() & 0x000fffff) ile dönen değer US Heap'deki indeksi verir. Buradan okunan indeksteki string değeri bu operand için data olarak kullanılır.

OperandType.ShortInlineVar: Operand'dan sonra bir byte'lık data vardır.

OperandType.InlineVar: Operand'dan sonra 2 byte'lık unsigned short değer okunur. Bir indeksi belirten bu değer method içinde tanımlanan local variable indeksidir. Yani sonuçta bu indeksteki variable bu operand'ın datasıdır.

Tablo 53'te program için metod tablosundan elde edilen sonuç değerleri vardır.

Tablo 53. Metot tablosu verileri

MethodStruct	**{Dimensions:[18]}**
[1]	
flags	145
impflags	0
name	247
param	1
rva	8272
signature	10
[2]	
flags	6278
impflags	0
name	252
param	1
rva	8299
signature	14
[3]	
flags	196
impflags	0
name	302
param	1
rva	8324
signature	22

Tablo 53’ ün devamı

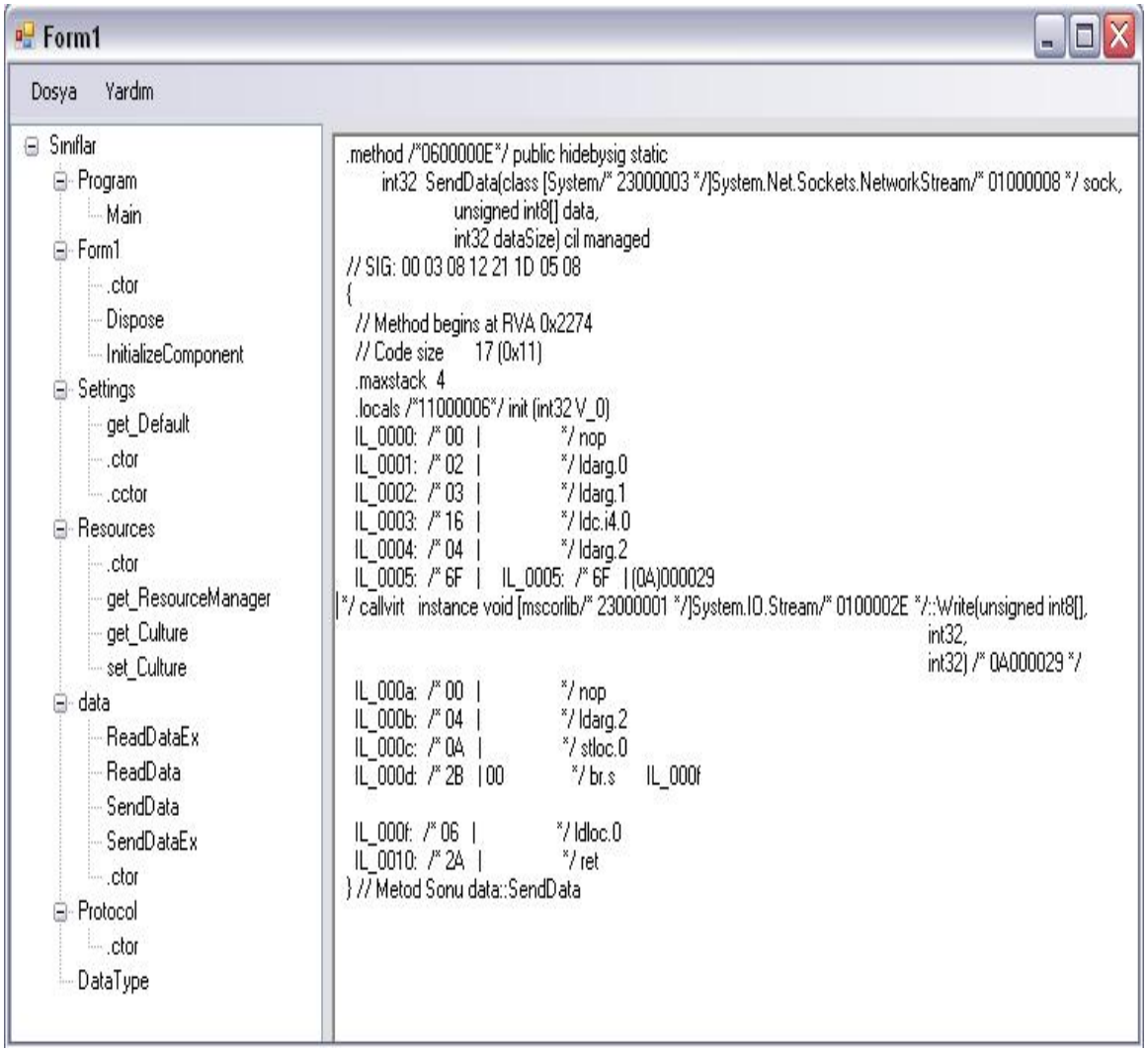
[4]	
flags	129
impflags	0
name	310
param	2
rva	8379
signature	14
[5]	
flags	2198
impflags	0
name	346
param	2
rva	8416
signature	31

RVA değerleri kod bloklarının başlama noktalarının adreslerini gösteriyor. Bu adreslere atlanarak her metodun içeriği okunur ve metonun koda dönüşümü gerçekleştirilir. Metod içeriğinin yapısı daha önceki bölümlerde verildiği için burada ayrıntısına girilmemiştir. İlerleyen örneklerde geri dönüşümü sağlanmış metod örnekleri verilecektir.

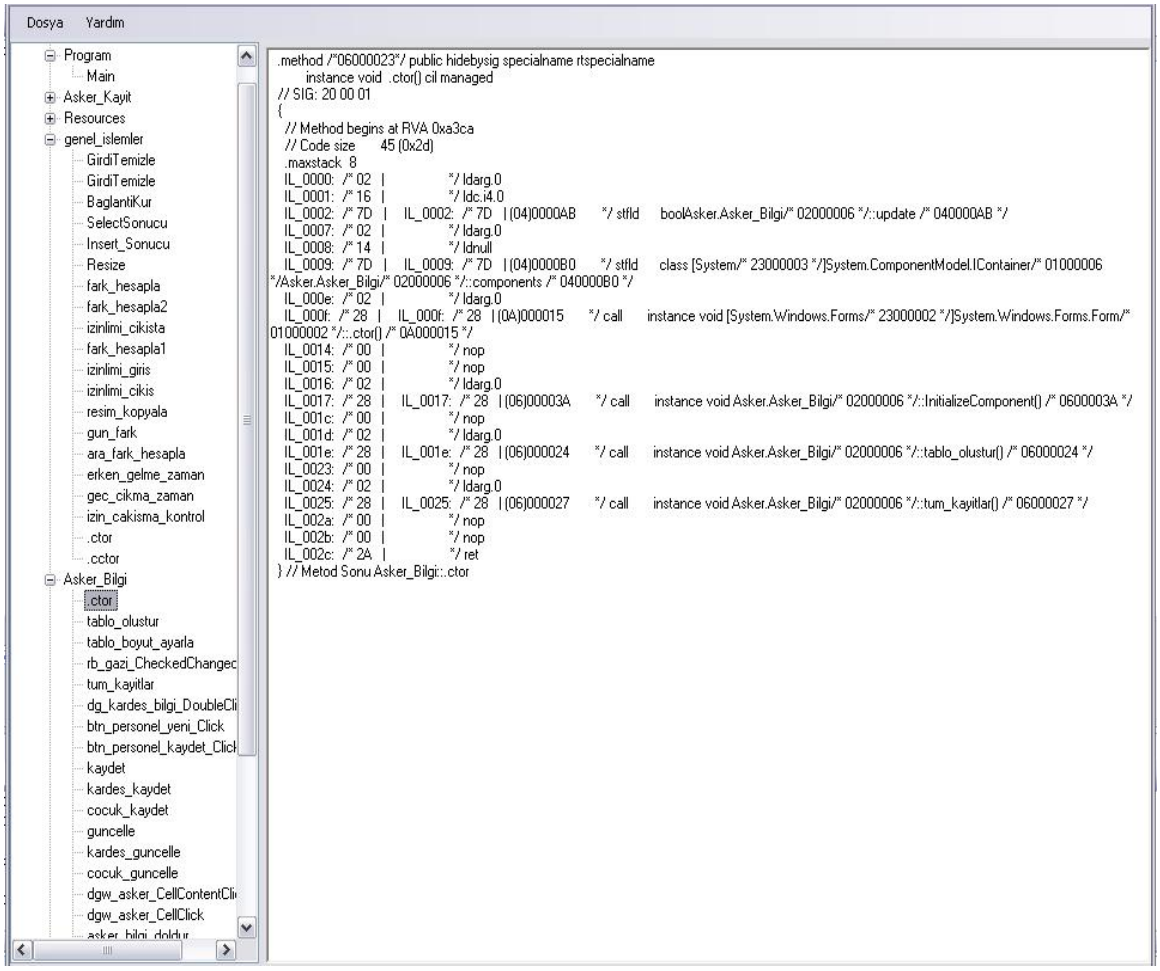
Flags değerinde metoda uygulanan nitelikler vardır. Örneğin ilk metod olan Main fonksiyonu (MethodStruct[1]) için bu değer “private hidebysig static”dir.

Param ise parametre tablosuna bir indeks değeridir ve metoda uygulanan parametre değerleri bu parametre tablosundan alınır.

Aşağıdaki 2 ayrı programdan elde edilen metotlar ve örnek bir metodun geri dönüş sonucu görülüyor. Şekil 16’da Tablo 53’teki metod tablosuna sahip Ek 14’te verilen programın sonucu, metotları ve bir metodun kodunun geri dönüşmüş hali gösterilmiştir.



Şekil 16. Örnek bir programın metotları ve bir metodun kodu 1

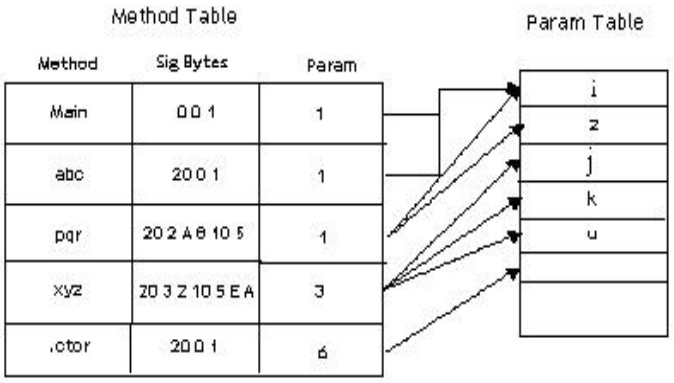


Şekil 17. Örnek bir programın metotları ve bir metodun kodu 2

Şekil 17'te görüldüğü gibi dosyanın içeriğinde sınıflar bulunmuş ve her sınıfın içindeki metotlar bir TreeWiev aracıyla gösterilmiştir. Ayrıca yapıcı fonksiyonunun (.ctor) kodu da sağ taraftaki metin bölümünde gösterilmiştir.

Param tablosu metod tablosunda kullanılır. Hatırlanacağı gibi metod tablosunda param diye bir alan vardır. Metot tablosundaki param alanı Param Tablosunu işaret eder. Aradaki ilişki aşağıdaki şekilde gösterilmiştir:

```
b.cs
public class zzz {
public static void Main()
{
}
public void abc()
{
}
public long pqr( int i , out byte  z)
{
z = 10;
return 0;
}
public bool xyz( ref byte  j , string k , long u)
{
return true;
}
}
```



Şekil 18. Metod tablosuyla param tablosu arasındaki ilişki

Yukarıdaki tabloda hem abc hemde prg fonksiyonu aynı parametreyi işaret ediyor. Ama bu yanıltıcı bir bilgidir. Normalde abc'nin parametresi yoktur. Blob kümesinde okunan count değeri 0'dır (count değeri parametre sayısını gösterir). O yüzden burada param alanının işaret ettiği yerin önemi abc fonksiyonu için yoktur. Normalde paramdaki her satır metod tablosundaki sadece bir alan içindir. İki alan için kullanılmaz.

Bir metodun kaç parametresi olduğu bulunur. Param tablosuna gidilir ve metod tablosundaki param değerinden itibaren kaç parametre varsa sırayla okunur.

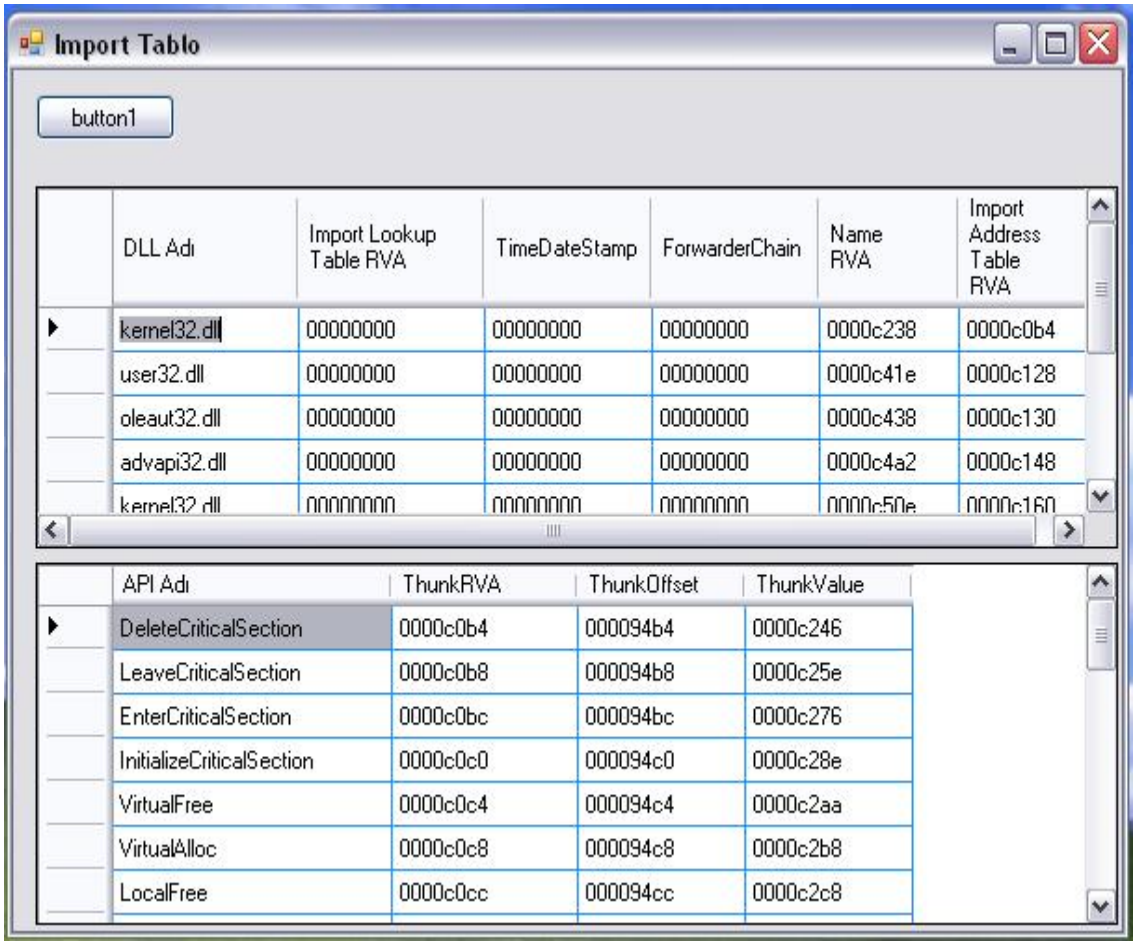
Uygun yaklaşım ilk önce metot tablosunun blob alanı okunur, bu parametrelerin sayısına ve türüne bakılır. Son olarak param tablosundan değeri alınır.

.Net te parametrelere default değer verilemeyeceğinden hasdefault daima 0 olur.

Sequence sayısı 0 olabilir. Bu return tipini gösterir.

2.5. Import Tablosundaki Bilgilerin Okunması

Bu bölümde ise Bölüm 1.3.2.7.'de anlatılan import tablosundan elde edilen sonuç değerleri gösterilecektir.



Şekil 19. Import tablosu örnek program sonuç değerleri

Şekil 19'da görüldüğü gibi import tablosundan program hangi dll'leri kullandığı, bu dll'lerin içinden hangi fonksiyonları kullandığı bulanabilir. Bu şekilde programın hangi sistem kaynaklarını kullandığı hakkında bilgi sahibi olunabilir.

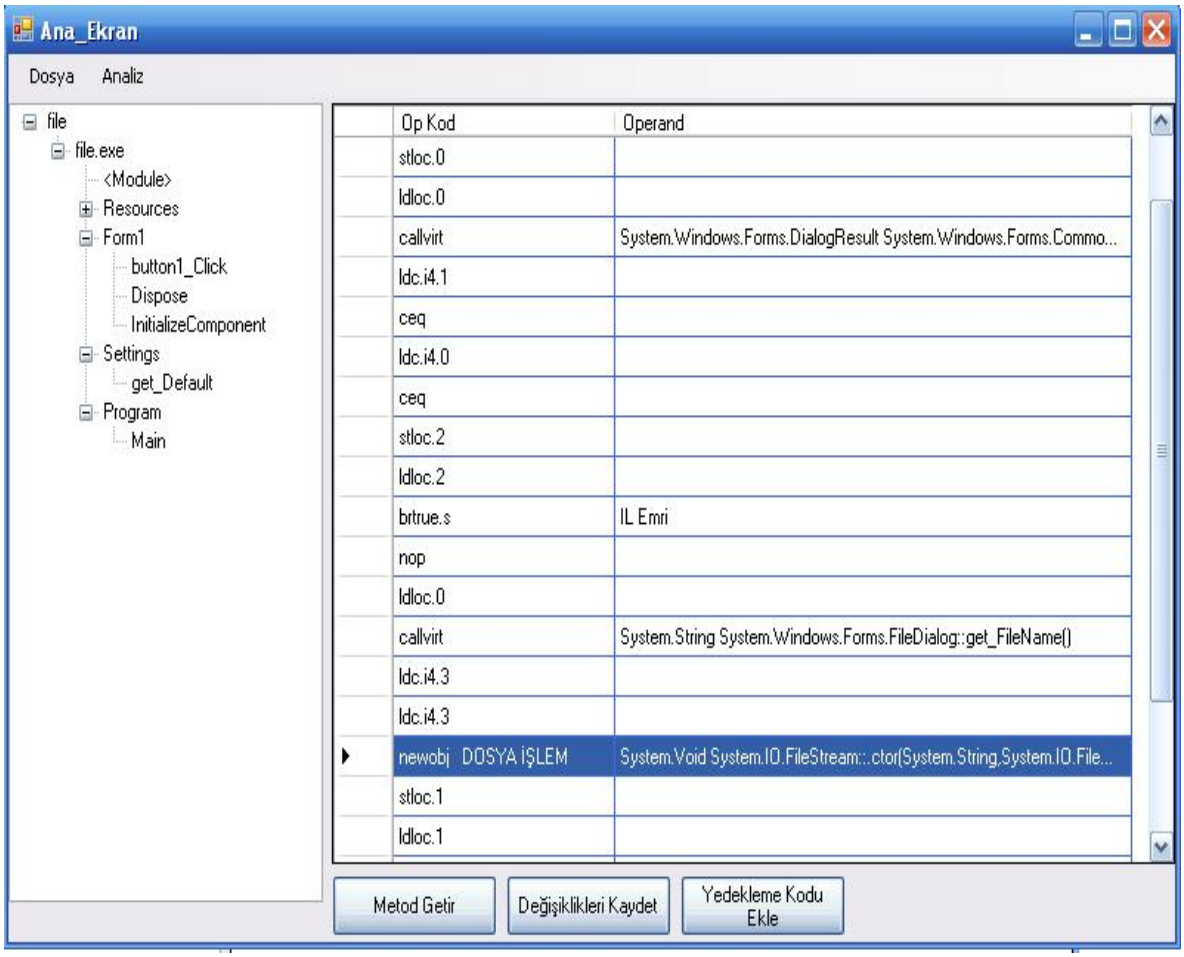
2.6. Dosya İşlemlerinin Algılanması

Şekil 20'de görüldüğü gibi örnek programda ilk önce dosya işlemleri incelenecek program Dosya->Aç menüsünden seçiliyor. Aç butonuna tıklandıktan sonra program dosyanın yapısını inceliyor ve dosyanın başlık bilgilerini (metadata tabloları, clr başlığı, imaj başlığı, vs.) belleğe yüklüyor. Sonra metadata tablolarından gerekli bilgileri alarak dosyanın metot sınıf hiyerarşisini treeview bileşenine aktarır.

2. aşamada ise dosya işlemlerinin algılanması için analiz aşaması gelir. Programda Analiz->Başla menüsüne tıklandığında analiz işlemi devreye giriyor. Bu işlemde

programdaki tüm metotların gövde bölümü sırasıyla inceleniyor. Bölüm 1.4.'te anlatıldığı gibi eğer dosya işlemi bulunursa, hangi metotta olduğu ve metodun hangi emrinde olduğu bilgileri bir diziye aktarılıyor. Dosya işlemlerinin nasıl algılandığı Bölüm 1.4.'te detaylı anlatılmıştır.

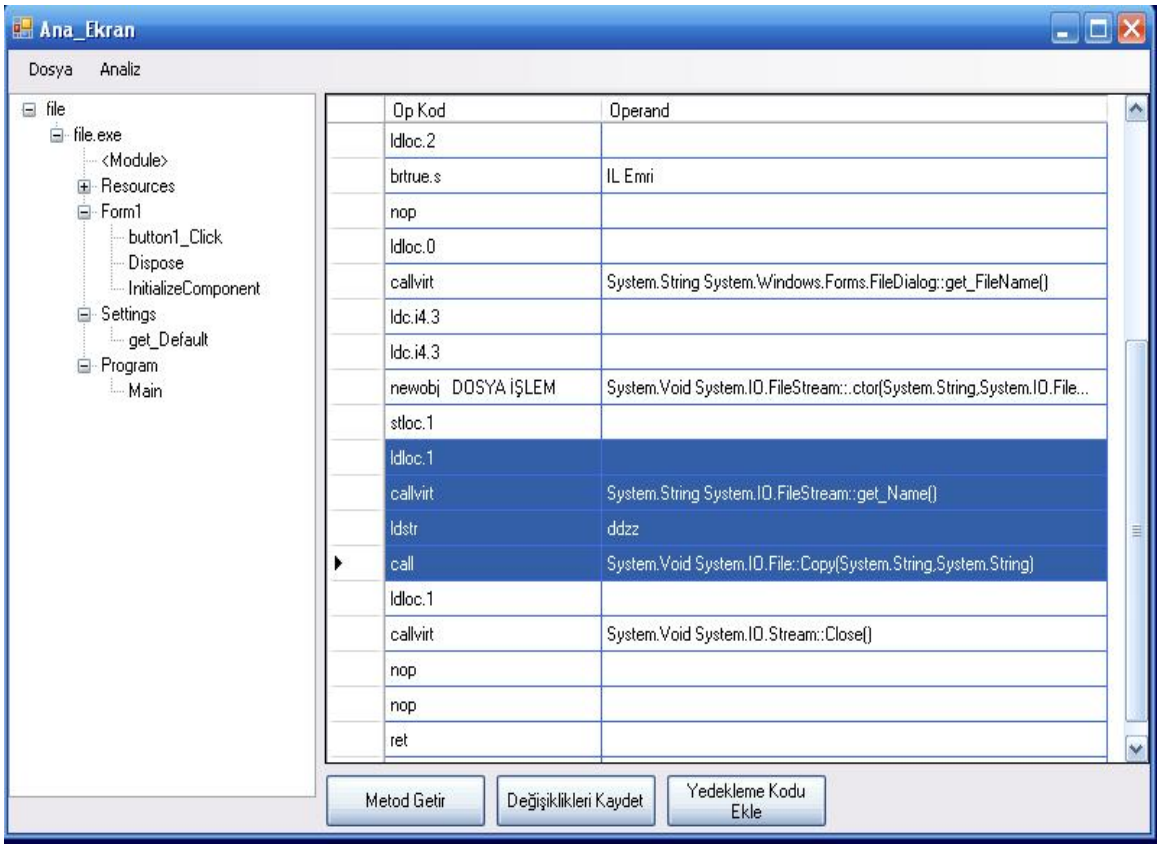
Son olarak ise Metod Getir butonu tıklanırsa, eğer dosya işlemi bulunan metot varsa bu metodun emirlerini sağ taraftaki datagrid'e dolduruyor. Ve dosya işlemi yapılan kod yerinin de görülebilmesi için emrin yanına "DOSYA İŞLEM" yazılır. Sonraki dosya işlemi için tekrar Metod Getir butonu tıklanır. Eğer dosya işlemi olan başka metot varsa ekrana sonraki metodun kodları getirilir.



Şekil 20. Dosya işlemi algılama

2.7. Dosya Yedekleme Kodu Eklenmesi

Dosyaya yedekleme kodu eklenmesi için programda öncelikle analiz işleminin tamamlanmış olması ve bir metodun seçili olması gereklidir. 2.6.'daki adımlar uygunlandıktan sonra "Yedekleme Kodu Ekle" butonu tıklanır. Karşımıza bir dosya kaydetme ekranı gelir. Bu ekran dosyaya eklenecek yedekleme kodu, değişiklik olacak dosyayı hangi dosyaya yedekleme yapılacağını belirlemek içindir. Bu dosya kaydetme işlemi de yapıldıktan sonra Tamam butonuna tıklanır ve aşağıdaki şekilde görüldüğü gibi yedekleme kodu metoda eklenmiş olur. Bu işlemin detaylı anlatımı bölüm 1.4. te verildiği için.burada yeniden ayrıntısına girilmemiştir.



Şekil 21. Yedekleme kodu eklenmiş metod

Şekil 21'da da görüldüğü gibi dosyaya yeni kodlar eklenmiştir. Son aşama ise değişikliklerin kaydedilmesidir. Bunun için Değişiklikleri Kaydet butonu tıklanır. Gelen ekrandan bir exe dosyası ismi yazılıp kaydete basılır.

Şekil 22. Yeni exe sonucu

Şekil 22'de file.exe orijinal dosyayı, file1.exe değiştirilmiş dosyayı gösteriyor. File1.exe çalıştığında kopyalanacak dosyayı ddzz (örnek programda seçili dosya) adında yeni bir dosyaya kopyalamaktadır.

3. SONUÇLAR

Bu çalışmadaki amaç .NET platformunda yazılmış programların koda geri dönüştürülmesi ve imaj dosyasına yeni eklentiler yaparak bazı bölümlerinde imaj dosyasının yapılmasına izin verilen işlemleri gerçekleştirmesini saklamaktır. En önemli yapıların koda dönüştürülmesi tamamlanmıştır. Import tablosundaki bilgilerin okunması, hangi sistem kaynaklarının kullanıldığının anlaşılması gerçekleştirilmiştir. Dosya başlıkları başarılı bir şekilde yorumlanarak program hakkında detaylı bilgilere ulaşılmıştır. Metadata ve akımlar başarılı bir şekilde okunarak metadata tablolarıyla akım tabloları doğru bir şekilde elde edilmiştir. Programdaki değişkenler, sabitler, sınıflar, namespaceler, metotlar, metot parametreleri başarılı bir şekilde alınmıştır. Metot içerikleri başarıyla inşa edilmiştir. İmaj dosyasındaki dosya işlemlerinin algılanması sağlanmıştır. Bir metot içerisindeki bir dosya yazılmak için açılmışsa bu metoda yedekleme kodu eklenmiştir. Bu sayede önemli dosyalar için bir geri dönüşüm mekanizması meydana getirilmiştir.

4. ÖNERİLER

1. Yazılım dünyasında geliştirilmiş, özellikle patent almış ürünlerin tersine mühendislik yöntemleri ile yeniden yazılabilir olması birçok yazılımcıyı bu mühendislik alanına yönlendirmiştir. Bir kişinin bir programı kodlarındaki değişkenleri, fonksiyon isimlerini değiştirerek kendisi yapmış gibi gösterebilmesine izin veren .NET platformunda yazılan programların güvenilirliği bir tartışma konusudur. Daha güvenli olması ve yazılım geliştiricilerin ya da şirketlerin haklarının korunması için CLR bölümündeki verilerin IL kodu olarak değilde native koda dönüştürülüp dosyaya eklenmesi gereklidir.
2. Bunun yanında derleyicinin fonksiyon isimlerini ve değişken isimlerini anlaşılmaz hale getirip dosyaya bu şekilde kaydetmesi geri dönüşüm sonucu programın anlaşılabilirliliğini zorlaştıracaktır. Native koda dönüşüm kadar olmasa bile bu konu da bir önlem olarak düşünülebilir.

5. KAYNAKLAR

1. Hoglund, G. ve McGraw, G., Exploiting Software How to Break Code, Addison Wesley, New Jersey, 2004

2. Dereli, T. ve Baykasoğlu, A., Tersine mühendislik, http://www.turkcadcam.net/rapor/tersine-muh/index.html, 13 Nisan 2005

3. Chess, B. ve West, J., Secure Programming with Static Analysis, Addison Wesley, New Jersey, 2007

4. Thai, T. ve Lam,H.Q., .NET Framework Essentials, O'Reilly, New York, 2003

5. MICROSOFT, Microsoft Portable Executable and Common Object File Format Specification, MICROSOFT, 2006

6. Microsoft, ECMA and ISO/IEC C# and Common Language Infrastructure Standards, 2003.

7. Pistelli, D. The .NET File Format. http://www.codeproject.com/KB/dotnet/dotnetformat.aspx, 13 Ekim 2006

8. Gough, J. ve Corney, D. Reading and Writing PE-files with PERWAPI, http://plas.fit.qut.edu.au/perwapi/files/PERWAPI.pdf 14 Ekim 2005..

9. Danehkar, A. The Code Project, http://www.codeproject.com/KB/system/inject2exe.aspx, 25 Aralık 2005.

10. Pietrek, M., Peering Inside the PE: A Tour of the Win32 Portable Executable File Format, http://msdn2.microsoft.com/en-us/library/ms809762.aspx, 21 Şubat 2006.

11. Sümerkent, K., Common Language Runtime 2, http://www.msakademik.net/makaleler_detay.aspx?id=79, 30 Haziran 2003.

12. TAŞDELEN, A., Assembly Kavramı, http://www.msakademik.net/makaleler_detay.aspx?id=7 , 22 Temmuz 2002.

13. Sümerkent, K., Common Language Runtime 1, http://www.msakademik.net/makaleler_detay.aspx?id=78, 12 Ocak 2003.

14. http://www.ssw.uni-linz.ac.at/Teaching/Lectures/Sem/2001/Literatur/FileFormatSpec.doc, File Format Spec, 10 Ekim 2000.

15. http://msdn2.microsoft.com/en-us/library/ms680301(VS.85).aspx IMAGE_COFF_SYMBOLS_HEADER Structure., 13 Nisan 2007.

16. http://msdn2.microsoft.com/en-us/library/ms680305(VS.85).aspx IMAGE_DATA_DIRECTORY Structure, 13 Nisan 2007.

17. http://msdn2.microsoft.com/en-us/library/ms680307(VS.85).aspx IMAGE_DEBUG_DIRECTORY Structure, 13 Nisan 2007.

18. http://msdn2.microsoft.com/en-us/library/ms680313(VS.85).aspx IMAGE_FILE_HEADER Structure, 13 Nisan 2007.

19. http://msdn2.microsoft.com/en-us/library/ms680316(VS.85).aspx IMAGE_FUNCTION_ENTRY Structure, 13 Nisan 2007.

20. http://msdn2.microsoft.com/en-us/library/ms680328(VS.85).aspx IMAGE_LOAD_CONFIG_DIRECTORY64 Structure, 13 Nisan 2007.

21. http://msdn2.microsoft.com/en-us/library/ms680336(VS.85).aspx IMAGE_NT_HEADERS Structure, 13 Nisan 2007.

22. http://msdn2.microsoft.com/en-us/library/ms680339(VS.85).aspx IMAGE_OPTIONAL_HEADER Structure, 13 Nisan 2007.

23. http://msdn2.microsoft.com/en-us/library/ms680341(VS.85).aspx IMAGE_SECTION_HEADER Structure, 13 Nisan 2007.

24. http://www.mono-project.com/Main_Page, Mono, 13 Nisan 2007.

25. http://etutorials.org/Programming/.NET+Framework+Essentials/Chapter+2.+The+Common+Language+Runtime/2.5+Intermediate+Language+IL/aspx, Intermediate Language (IL)

6. EKLER

Ek 1. MS-DOS Başlığı

```
internal unsafe struct IMAGE_DOS_HEADER
{
  public ushort e_magic;          // Magic number
public ushort e_cblp;           // Bytes on last page of file
public ushort e_cp;             // Pages in file
public ushort e_crlc;           // Relocations
public ushort e_cparhdr;        // Size of header in paragraphs
public ushort e_minalloc;       // Minimum extra paragraphs needed
public ushort e_maxalloc;       // Maximum extra paragraphs needed
public ushort e_ss;             // Initial (relative) SS value
public ushort e_sp;             // Initial SP value
public ushort e_csum;           // Checksum
public ushort e_ip;             // Initial IP value
public ushort e_cs;             // Initial (relative) CS value
public ushort e_lfarlc;         // File address of relocation table
public ushort e_ovno;           // Overlay number
public fixed short e_res[4];    // Reserved words
public ushort e_oemid;          // OEM identifier (for e_oeminfo)
public ushort e_oeminfo;        // OEM information; e_oemid specific
public fixed short e_res2[10];  // Reserved words
 public uint e_lfanew; }
```

Ek 2. İmaj Dosya Başlığı

Ek Tablo 1. İmaj dosya başlığı

Offset	Boyut	Alan	Açıklama
0	2	Machine	Bu numara hedef makinenın türünü tanımlar. Ek 3'te alabileceği değerler gösterilmiştir.
2	2	NumberOfSections	Bölüm (section) sayısını belirtir. Başlıktan hemen sonra başlayan bölüm tablosunun boyutunu gösterir. Windows ta maksimum bölüm (section) sayısı 96 ile sınırlanmıştır.
4	4	TimeDateStamp	00:00 1 ocak, 1970 (C run-time time_t value) dan başlayan bu 32 bitlik saniye değeri dosyanın ne zaman oluşturulduğunu gösterir.
8	4	PointerToSymbolTable	COFF sembol tablosunun ofset değeri(eğer kullanılmıyorsa tablo 0 değeri alır). İmaj dosyalarında 0 değeri alır.
12	4	NumberOfSymbols	Sembol tablosundaki girişlerin sayısı sembol tablosundan hemen sonra başlayan string tablosunun yerini göstermek için kullanılır. İmaj dosyalarında 0 değeri alır.
16	2	SizeOfOptionalHeader	Obje dosyasında kullanılmayan sadece icraedilebilir dosyalarda kullanılan opsiyonel başlığın boyutunu belirtir.
18	2	Characteristics	Dosyanın niteliklerini gösteren bayrağı belirtir. Ek 4'te kullanılan özel karakteristikler var.

Ek 3. İşlemci Türleri

Ek Tablo 2. İşlemci türleri

Sabit	Değer	Açıklama
IMAGE_FILE_MACHINE_UNKNOWN	0x0	Bu alanın içeriği uygulanabilir herhangi bir makine türü olduğunu farz eder.
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 veya daha sonra çıkan veya uyumlu olan işlemciler.
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium İşlemci Ailesi
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	FPU ile MIPS
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	FPU ile MIPS16
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Kayan nokta destekli Power PC
IMAGE_FILE_MACHINE_R4000	0x166	MIPS little endian
IMAGE_FILE_MACHINE_SH3	0x1a2	Hitachi SH3
IMAGE_FILE_MACHINE_SH3DSP	0x1a3	Hitachi SH3 DSP
IMAGE_FILE_MACHINE_SH4	0x1a6	Hitachi SH4
IMAGE_FILE_MACHINE_SH5	0x1a8	Hitachi SH5
IMAGE_FILE_MACHINE_THUMB	0x1c2	Thumb
IMAGE_FILE_MACHINE_WCEMIPSV2	0x169	MIPS little-endian WCE v2

Ek 4. Karakteristik

Ek Tablo 3. Karakteristik

Bayrak	Değer	Açıklama
IMAGE_FILE_RELOCS_STRIPPED	0x0001	Yalnızca imaj, Windows CE veya Microsoft Windows NT® veya sonrası.
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	Yalnızca imaj, bu imaj dosyasının geçerli ve koşabileceğini belirtir. Eğer setlenmemişse linker hatasını gösterir.
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	COFF satır numarası kaldırılmıştır. Onaylanmamıştır ve değeri 0 olmalıdır.
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	Lokal semboller için COFF tablo girişi kaldırılmıştır. Onaylanmamıştır ve değeri 0 olmalıdır.
IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010	Kullanılmıyor. Windows 2000 ve sonrası için önemsenmez ve 0'a setlenir
IMAGE_FILE_LARGE_ADDRESS_ AWARE	0x0020	Uygulama 2 GB adres tutabilir.
	0x0040	Gelecekte kullanılmak için ayrılmıştır.
IMAGE_FILE_BYTES_REVERSED_LO	0x0080	Onaylanmamıştır ve değeri 0 olmalıdır.
IMAGE_FILE_32BIT_MACHINE	0x0100	Makine 32 bit kelime mimarisi temelindedir.
IMAGE_FILE_DEBUG_STRIPPED	0x0200	Debugging bilgisi imaj dosyasından kaldırılmıştır.
IMAGE_FILE_REMOVABLE_RUN_ FROM_SWAP	0x0400	Eğer imaj dosyası sadece kaldırılabilir bir araçtaysa onu tam yükler ve swap dosyasına kopyalar.
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	Eğer imaj dosyası network aracındaysa onu tam yükler ve swap dosyasına kopyalar.
IMAGE_FILE_SYSTEM	0x1000	İmaj dosyasının kullanıcı dosyası olmayıp sistem dosyası olduğunu gösterir.
IMAGE_FILE_DLL	0x2000	İmaj dosyasının dinamik link kütüphanesi (DLL) olduğunu gösterir.

Ek Tablo 3’ ün devamı

Bayrak	**Değer**	**Açıklama**
IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	Dosyanın tek işlemcili bir makinede çalışması gerektiğini gösterir.
IMAGE_FILE_BYTES_REVERSED_HI	0x8000	Onaylanmamıştır ve değeri 0 olmalıdır.

Ek 5. Opsiyonel Başlık Standart Alanlar

Ek Tablo 4. Opsiyonel başlık standart alanlar

Ofset	Boyut	Alan	Açıklama
0	2	Magic	Bu işaretsiz tamsayı imaj dosyasının durumunu tanımlar. En çok kullanılan numara olan 0x10B normal çalıştırılabilir dosyayı tanımlar. 0x107 bunun ROM imajı olduğunu tanımlar ve 0x20B PE32+ çalıştırılabiliri olduğunu tanımlar.
2	1	MajorLinkerVersion	Linker'in asıl sürüm numarasını belirtir.
3	1	MinorLinkerVersion	Linker'in ikincil sürüm numarasını belirtir.
4	4	SizeOfCode	Kod (metin) bölümünün boyutunu veya eğer birden fazla kod bölümü varsa bunların tümünün toplamının boyutunu gösterir.
8	4	SizeOfInitializedData	Başlangıç değeri atanmış veri bölümünün boyutunu verir veya eğer birden fazla başlangıç değeri atanmış veri bölümü varsa bunların tümünün toplamının boyutunu verir.
12	4	SizeOfUninitializedData	Başlangıç değeri atanmamış veri bölümünün (BSS) boyutunu verir veya eğer birden fazla BBS bölümü varsa bunların tümünün toplamının boyutunu verir.
16	4	AddressOfEntryPoint	Çalıştırılabilir dosya belleğe yüklendiği zaman giriş noktalarının imaj tabanına göreceli adresi. Program imajı için bu başlama adresidir (c, c#, Java gibi diller için main fonksiyonu). Aygıt sürücüleri için başlangıç fonksiyonunun adresini verir. DLL için bir giriş noktası opsiyoneldir. Eğer giriş noktası kullanılmayacaksa bu alan sıfırlanır.
20	4	BaseOfCode	İmaj belleğe yüklendiğinde kod bölümünün başlangıcının, imaj tabanına göreceli adresidir.

Ek 6. Opsiyonel Başlık Windows Özel Alanlar

Ek Tablo 5. Windows özel alanlar

Offset (PE32/ PE32+)	**Boyut (PE32/ PE32+)**	**Alan**	**Açıklama**
28/24	4/8	ImageBase	İmaj belleğe yüklendiğinde imajın tercih edilen adresinin birinci baytıdır. 64 K'nın katı olmalıdır. DLL için varsayılan değer 0x10000000'dir. Windows CE için varsayılan değer 0x00010000'dir. Windows NT, Windows 2000, Windows XP, Windows 95, Windows 98 ve Windows Me için varsayılan değer 0x00400000'dir.
32/32	4	SectionAlignment	Bölümler belleğe yüklendiğinde bunların sıralanması gösterir. FileAlignment ten büyük veya ona eşit olmalıdır. Mimari için varsayılan sayfa boyutudur.
36/36	4	FileAlignment	Bölümlerin satır datasını hizalamak için kullanılan hizalama etmeni. 2'nin üstü olmalıdır ve 512 ile 64 arası değer alır. Varsayılan 512'dir.
40/40	2	MajorOperatingSystemVersion	Gerekli işletim sisteminin ana sürüm numarasını gösterir.
42/42	2	MinorOperatingSystemVersion	Gerekli işletim sisteminin ikincil sürüm numarasını gösterir.
44/44	2	MajorImageVersion	İmajın asıl sürüm numarasını gösterir.
46/46	2	MinorImageVersion	İmajın ikincil sürüm numarasını gösterir.
48/48	2	MajorSubsystemVersion	Alt sistemin asıl sürüm numarası.
50/50	2	MinorSubsystemVersion	Alt sistemin ikincil sürüm numarası
52/52	4	Win32VersionValue	Ayrılmış alan, 0 olmalıdır.
56/56	4	SizeOfImage	İmaj belleğe yüklendiği zaman tüm başlıkları içerecek şekilde imajın boyutu. SectionAlignment in katı olmalıdır.

Ek Tablo 5. Windows özel alanlar'ın devamı

Offset (PE32/ PE32+)	**Boyut (PE32/ PE32+)**	**Alan**	**Açıklama**
60/60	4	SizeOfHeaders	MS-dos stub programı, PE başlığı ve bölüm başlıklarının birleşiminin boyutunu gösterir. FileAlignment değerinin katına göre yuvarlatılmalıdır.
64/64	4	CheckSum	İmaj dosyasının checksum değeri. Checksum hesaplama algoritması IMAGHELP.DLL dosyasının içindedir.
68/68	2	Subsystem	Bu imajın koşması için gerekli olan alt sistemleri gösterir. Daha fazla bilgi için Ek 7 "Alt Sistemler" e bakınız.
70/70	2	DllCharacteristics	DLL'in karakterirstiğini gösteren bayraktır. Daha fazla bilgi için Ek 8 "DLL Karakteristiği" ne bakınız.
72/72	4/8	SizeOfStackReserve	Ayrılmış stağın (yığın) boyutu.
76/80	4/8	SizeOfStackCommit	İşlenmesi için yığının boyutunu gösterir.
80/88	4/8	SizeOfHeapReserve	Ayrılmış yerel küme boşluğunun boyutunu gösterir.
84/96	4/8	SizeOfHeapCommit	İşlenmesi için yerel yığın boşluğunun boyutu.
88/104	4	LoaderFlags	Ayrılmış alan, 0 olmalıdır.
92/108	4	NumberOfRvaAndSizes	Opsiyonel başlığın kalıntısındaki veri dizin kayıt numarasını gösterir.

Ek 7. AltSistemler

Ek Tablo 6. Altsistemler

Sabit	Değer	Açıklama
IMAGE_SUBSYSTEM_UNKNOWN	0	Bir bilinmeyen alt sistem olduğunu gösterir.
IMAGE_SUBSYSTEM_NATIVE	1	Aygıt sürücüleri ve geleneksel Windows süreçlerini gösterir.
IMAGE_SUBSYSTEM_WINDOWS_GUI	2	Windows grafiksel kullanıcı ara yüzü alt sistemini gösterir (GUI).
IMAGE_SUBSYSTEM_WINDOWS_CUI	3	Windows karakter alt sistemi
IMAGE_SUBSYSTEM_POSIX_CUI	7	Posix karakter alt sistemi.
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9	Windows CE.
IMAGE_SUBSYSTEM_EFI_APPLICATION	10	Bir uzatılabilir aygıt yazılımı ara yüz uygulaması (EFI) .
IMAGE_SUBSYSTEM_EFI_BOOT_ SERVICE_DRIVER	11	Boot servisleriyle EFI sürücüsü.
IMAGE_SUBSYSTEM_EFI_RUNTIME_ DRIVER	12	Koşma zamanı servisleriyle bir EFI sürücüsü.
IMAGE_SUBSYSTEM_EFI_ROM	13	Bir EFI ROM imajı.
IMAGE_SUBSYSTEM_XBOX	14	XBOX

Ek 8. Dll Karakteristiği

Ek Tablo 7. Dll karakteristiği

Sabit	Değer	Açıklama
	0x0001	Ayrılmış alan, 0 olmalıdır.
	0x0002	Ayrılmış alan, 0 olmalıdır.
	0x0004	Ayrılmış alan, 0 olmalıdır.
	0x0008	Ayrılmış alan, 0 olmalıdır.
IMAGE_DLL_CHARACTERISTICS_ DYNAMIC_BASE	0x0040	DLL yükleme zamanında yerleştirilebilir.
IMAGE_DLL_CHARACTERISTICS_ FORCE_INTEGRITY	0x0080	Kod bütünlük denetimi uygulanır.
IMAGE_DLL_CHARACTERISTICS_ NX_COMPAT	0x0100	İmaj NX uyumludur.
IMAGE_DLLCHARACTERISTICS_ NO_ISOLATION	0x0200	Yalıtımın farkındadır fakat imaj yalıtılmaz.
IMAGE_DLLCHARACTERISTICS_ NO_SEH	0x0400	Yapısal istisna (SE) işlemesi kullanılmaz. Bu imajda SE işleyicisi çağrılmaz.
IMAGE_DLLCHARACTERISTICS_ NO_BIND	0x0800	İmaj bağlanmaz.
	0x1000	Ayrılmış alan, 0 olmalıdır.
IMAGE_DLLCHARACTERISTICS_ WDM_DRIVER	0x2000	Bir WDM sürücü.
IMAGE_DLLCHARACTERISTICS_ TERMINAL_SERVER_AWARE	0x8000	Sonlandırma sunucusu farkındadır.

Ek 9. Veri Dizinleri

Ek Tablo 8. Veri dizinleri

Offset (PE/PE32+)	**Boyut**	**Alan**	**Açıklama**
96/112	8	Export Table	Export tablo adresi ve boyutunu gösterir.
104/120	8	Import Table	İmport tablo adresi ve boyutunu gösterir.
112/128	8	Resource Table	Kaynak tablosu adresi ve boyutunu gösterir.
120/136	8	Exception Table	Exception tablosu adresi ve boyutunu gösterir.
128/144	8	Certificate Table	Nitelik belgesi tablosu adresi ve boyutunu gösterir.
136/152	8	Base Relocation Table	Esas yerleşme tablosu adresi ve boyutu
144/160	8	Debug	Hata ayıklama başlama adresi ve boyutu
152/168	8	Architecture	Ayrılmış alan, 0 olmalıdır
160/176	8	Global Ptr	Global pointer kaydının saklanması için değerin RVA sı. Bu yapının boyutu sıfıra setlenmelidir.
168/184	8	TLS Table	Thread yerel kayıt (TLS) tablosunun adresi ve boyutu
176/192	8	Load Config Table	Yükleme biçim tablosunun adresi ve boyutu.
184/200	8	Bound Import	Bağlanan import tablosunun boyutu ve adresi.
192/208	8	IAT	Import adres tablosunun adresi ve boyutu
200/216	8	Delay Import Descriptor	Gecikme import tanımlayıcı adresi ve boyutu.
208/224	8	CLR Runtime Header	CLR çalışma zamanı başlığı adresi ve boyutu.
216/232	8	Ayrılmış Alan, 0 olmalıdır.	

Ek 10. Bölüm Başlığı

Ek Tablo 9. Bölüm başlığı

Ofset	Boyut	Alan	Açıklama
0	8	Name	UTF–8 enkodlama ile sonuna null değeri eklenmiş 8 byte boyutundaki string. Eğer string tam 8 karakter uzunluğunda ise sonuna null karakter eklenmez. Eğer daha fazla uzun bir isim olursa bu alan slash (/) karakterini ve bunu takip eden string tablosundaki konumu belirten bir decimal numarası içerir. Çalıştırılabilir imaj string tablosu kullanmaz ve 8 karakterden uzun ismi desteklemez.
8	4	VirtualSize	Bölüm belleğe yüklendiğinde, bölümün toplam boyutudur. SizeOfRawData'dan daha büyükse bölüme 0 eklenerek doldurulur. Sadece çalıştırılabilir imajlar için geçerlidir.
12	4	Virtual Address	İcra edilebilen imajlar için imaj dosyasının tabanına göre bölüm adresinin ilk byte'nı gösterir.
16	4	SizeOfRaw Data	Bölümün boyutu veya diskte başlatılan datanın boyutunu gösterir. İmaj dosyaları için opsiyonel başlıktaki FileAlignment değerinin katı olmalıdır.
20	4	PointerTo RawData	COFF dosyasının içinde bölümün ilk sayfası için gösterge. İmaj dosyaları için opsiyonel başlıktaki FileAlignment'in katı olmalıdır.
24	4	PointerTo Relocations	Bölümdeki yerleştirme girişlerinin başlangıcı için dosya göstergesi. İmaj dosyaları için veya hiç yerleştirme olmadığını göstermek için 0'a setlenir.
28	4	PointerTo Linenumbers	İmaj dosyaları için sıfıra setlenmelidir.
32	2	NumberOfRelocations	İmaj dosyaları için sıfıra setlenir.
34	2	NumberOfLinenumbers	İmaj dosyaları için sıfıra setlenmelidir.
36	4	Characteristics	Bölümün karakteristiklerini tanımlar.

Ek 11. MetaData Tabloları

Ek Tablo 10. Metadata tabloları

Sıra	Satır Boyutu	Tablo Adı
0	10	Module
1	6	TypeRef
2	14	TypeDef
3	2	FieldPtr
4	6	Field
5	2	MethodPtr
6	14	MethodDef
7	2	ParamPtr
8	6	Param
9	4	InterfaceImpl
10	6	MemberRef
11	6	Constant
12	6	CustomAttribute
13	4	FieldMarshal
14	6	DeclSecurity
15	8	ClassLayout
16	6	FieldLayout
17	2	StandAloneSig
18	4	EventMap
19	2	EvetnPtr
20	6	Event
21	4	PropertyMap
22	2	PropertyPtr
23	6	Property
24	6	MethodSemantics
25	6	MethodImpl
26	2	ModuleRef
27	2	TypeSpec
28	8	ImplMap
29	6	FieldRVA

Ek Tablo 10’ un devamı

Sıra	Satır Boyutu	Tablo Adı
30	8	EncLog
31	4	EncMap
32	22	Assembly
33	4	AssemblyProcessor
34	12	AssemblyOS
35	20	AssemblyRef
36	6	AssemblyRefProcessor
37	14	AssemblyRefOS
38	8	File
39	14	ExportedType
40	12	ManifestResource
41	4	NestedClass

Ek 12. Metot Tablosu İmzaları

Ek Tablo 11. Metot tablosu imzaları

ELEMENT_TYPE_END	0x0,
ELEMENT_TYPE_VOID	0x1,
ELEMENT_TYPE_BOOLEAN	0x2,
ELEMENT_TYPE_CHAR	0x3,
ELEMENT_TYPE_I1	0x4,
ELEMENT_TYPE_U1	0x5,
ELEMENT_TYPE_I2	0x6,
ELEMENT_TYPE_U2	0x7,
ELEMENT_TYPE_I4	0x8,
ELEMENT_TYPE_U4	0x9,
ELEMENT_TYPE_I8	0xa,
ELEMENT_TYPE_U8	0xb,
ELEMENT_TYPE_R4	0xc,
ELEMENT_TYPE_R8	0xd,
ELEMENT_TYPE_STRING	0xe,
ELEMENT_TYPE_PTR	0xf
ELEMENT_TYPE_BYREF	0x10
ELEMENT_TYPE_VALUETYPE	0x11
ELEMENT_TYPE_CLASS	0x12
ELEMENT_TYPE_VAR	0x13
ELEMENT_TYPE_ARRAY	0x14
ELEMENT_TYPE_GENERICINST	0x15
ELEMENT_TYPE_TYPEDBYREF	0x16
ELEMENT_TYPE_I	0x18
ELEMENT_TYPE_U	0x19
ELEMENT_TYPE_FNPTR	0x1B
ELEMENT_TYPE_OBJECT	0x1C
ELEMENT_TYPE_SZARRAY	0x1D
ELEMENT_TYPE_MVAR	0x1e
ELEMENT_TYPE_CMOD_REQD	0x1F
ELEMENT_TYPE_CMOD_OPT	0x20
ELEMENT_TYPE_INTERNAL	0x21
ELEMENT_TYPE_MAX	0x22

Ek Tablo 11'in devamı

ELEMENT_TYPE_MODIFIER	0x40
ELEMENT_TYPE_SENTINEL	0x41
ELEMENT_TYPE_PINNED	0x45
ELEMENT_TYPE_R4_HFA	0x46

Ek 13. Mapping Flags Değerleri

Ek Tablo 12. Mapping flags değerleri

pmNoMangle	0x0001
pmCharSetMask	0x0006
pmCharSetNotSpec	0x0000
pmCharSetAnsi	0x0002
pmCharSetUnicode	0x0004
pmCharSetAuto	0x0006
pmBestFitUseAssem	0x0000
pmBestFitEnabled	0x0010
pmBestFitDisabled	0x0020
pmBestFitMask	0x0030
pmThrowOnUnmappableCharUseAssem	0x0000
pmThrowOnUnmappableCharEnabled	0x1000
pmThrowOnUnmappableCharDisabled	0x2000
pmThrowOnUnmappableCharMask	0x3000
pmSupportsLastError	0x0040
pmCallConvMask	0x0700
pmCallConvWinapi	0x0100
pmCallConvCdecl	0x0200
pmCallConvStdcall	0x0300
pmCallConvThiscall	0x0400
pmCallConvFastcall	0x0500
pmMaxValue	0xFFFF

Ek 14. Parent alanının ilk 5 bitinin alabileceği değerler

Ek Tablo 13. Parent alanının ilk 5 bitinin alabileceği değerler

HasCustomAttribute: 5 bits to encode tag	**Tag**
MethodDef	0
FieldDef	1
TypeRef	2
TypeDef	3
ParamDef	4
InterfaceImpl	5
MemberRef	6
Module	7
Permission	8
Property	9
Event	10
StandAloneSig	11
ModuleRef	12
TypeSpec	13
Assembly	14
AssemblyRef	15
File	16
ExportedType	17
ManifestResource	18

Ek 15. Örnek Program Kodları

```
public class data
{
   public static int ReadDataEx(NetworkStream sock, ref Protocol.DataType
dataType, ref byte[] data)
   {
      //read data type
      ReadData(sock, ref data, Marshal.SizeOf(typeof(short)));
      dataType = (Protocol.DataType)BitConverter.ToInt16(data, 0);
      //read data size
      ReadData(sock, ref data, Marshal.SizeOf(typeof(int)));
      int dataSize = BitConverter.ToInt32(data, 0);
      if (dataSize <= 0)
         return 0;
      //get the data from the remote computer
      if (data == null || data.Length < dataSize)
         data = new byte[dataSize];
      ReadData(sock, ref data, dataSize);
      return dataSize;
   }

   public static void ReadData(NetworkStream sock, ref byte[] buffer, int
dataSize)
   {
      int size, index = 0;

      if (buffer == null || buffer.Length < dataSize)
         buffer = new byte[dataSize];

      while (dataSize > 0)
      {
         size = sock.Read(buffer, index, dataSize);
         if (size == 0)
            throw (new Exception("Bağlantı kapatıldı"));
         dataSize -= size;
         index += size;
      }
   }


   public static int SendData(NetworkStream sock, byte[] data, int dataSize)
   {
      sock.Write(data, 0, dataSize);
      return dataSize;
   }
```

```
        public static int SendDataEx(NetworkStream sock, Protocol.DataType
dataType, int dataSize, byte[] data)
        {
            int bytes;
            //send data type
            bytes = SendData(sock, BitConverter.GetBytes((short)dataType),
Marshal.SizeOf(typeof(short)));
            //send data size
            bytes += SendData(sock, BitConverter.GetBytes(dataSize),
Marshal.SizeOf(typeof(int)));
            if (dataSize > 0)
            {
                //send data
                bytes += SendData(sock, data, dataSize);
            }
            return bytes;
        }
    }


    public class Protocol
    {
        public enum DataType
        {
            AUTH,
            AUTH_OK,
            NO_AUTH,
            INFO,
            CITIZEN_LEFT,
            WANT_TO_CONNECT,
            SERVERDISCONNECT,
        }
    }

    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
```

```
        public partial class Form1 : Form
        {
            public Form1()
            {
                InitializeComponent();
            }
        }

    partial class Form1
        {
            /// <summary>
            /// Required designer variable.
            /// </summary>
            private System.ComponentModel.IContainer components = null;

            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
            protected override void Dispose(bool disposing)
            {
                if (disposing && (components != null))
                {
                    components.Dispose();
                }
                base.Dispose(disposing);
            }

            #region Windows Form Designer generated code

            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent()
            {
                this.components = new System.ComponentModel.Container();
                this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
                this.Text = "Form1";
            }

            #endregion
        }
```

```
        [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.E
ditors.SettingsDesigner.SettingsSingleFileGenerator", "8.0.0.0")]
        internal sealed partial class Settings :
global::System.Configuration.ApplicationSettingsBase
        {

            private static Settings defaultInstance =
((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new
Settings())));

            public static Settings Default
            {
                get
                {
                    return defaultInstance;
                }
            }
        }
```

ÖZGEÇMİŞ

1981 yılında Horasan/ERZURUM'da doğdu. İlkokulu İnkılâp İlkokulunda, ortaokulu Horasan Lisesi'nde, lise öğrenimini Akpınar Anadolu Öğretmen Lisesi'nde tamamladı. 2000 yılında Karadeniz Teknik Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde lisans programına başladı ve 2005 yılında bu bölümden mezun oldu. Aynı yıl Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans programına başladı. 2005-2007 yılları arası Ata Yazılım A.Ş. de çalıştı. 2007 yılanda DSİ'de çalışmaya başladı. Halen bu görevini sürdürmektedir. Yabancı dil olarak İngilizce bilmektedir.