Tersine Mühendislik Yöntemlerini Kullanarak .NET Çevrelerinde Kaynak Koda Dönüşüm Ve Dosya Sistemi İşlemlerinin Kontrolü

Salih ARAS[1] ve Hüseyin PEHLİVAN[2]

Bilgisayar Mühendisliği Bölümü
Karadeniz Teknik Üniversitesi, 61080, Trabzon
[1]arass@dsi.gov.tr, [2]pehlivan@ktu.edu.tr

Özet

Bu çalışmada, tersine mühendislik tekniklerinden yararlanarak, .NET çevreleri için geliştirilen programların kaynak koda dönüştürülmesi ve onların dosya sistemi işlemlerinin kontrolü amaçlanmıştır. Bu yol içerisinde PE dosya yapısı, CLR yapısı, metadata tabloları ve erişilen sistem kaynakları tespit edilmiştir. Dahası, programdaki değişkenler, sabitler, sınıflar, ad uzayları, metotlar ve onların parametreleri belirlenmiş ve bunun sonucu olarak metot gövdesi başarıyla oluşturulmuştur. Programların imaj dosyaları incelenerek, onların muhtemel dosya ve kayıt işlemlerinin algılanması gerçekleştirilmiştir.

Abstract

This work deals with the conversion of programs into source code and the control of their file system operations in .NET environments, using some reverse engineering techniques. In this way, PE file structure, CLR structure, metadata tables and accessed system resources are identified. Furthermore, variables, constants, classes, namespaces, methods and their parameters are determined, and thus method body is constructed successfully. Analysing the image files of programs, their possible file and registry operations are detected.

1. Giriş

Son zamanlarda tersine mühendislik çalışmaları hız kazanmaya başlamıştır. Daha kolay yoldan bilgiye erişmek isteyen insanlar, başkalarının elde ettiği başarıyı daha kısa zamanda yakalamanın yollarını aramaktadırlar. Tersine mühendislik sadece yazılım dünyasında değil, hayatın çoğu alanında karşımıza çıkabilmektedir. Örneğin, bir A firması üzerinde yıllarca çalışarak güzel bir makine geliştirmiş olsun. Eğer bir B firması da benzer bir makine geliştirmek istiyorsa ya A firmasının izlediği süreçten geçmesi ya da makinenin işlevlerini çözerek bir benzerini yapması gerekecektir.

Bu çalışmada da benimsenen yöntem, .NET çevrelerinde çalışan programlar analiz edilerek işlevlerinin çözümlenmesi üzerine dayandırılmıştır. Programların imaj dosyaları incelenerek bir yandan karşılığı olan IL kaynak kodlarına dönüşüm yapılmış diğer yandan muhtemel dosya ve kayıt işlemleri belirlenmiştir.

2. Tersine Mühendislik

Tersine mühendislik bir aygıtın, objenin veya sistemin; yapısının, işlevinin veya çalışmasının, çıkarımcı bir akıl yürütme analiziyle keşfedilmesi işlemidir. Bu yöntem, genellikle orijinalinden kopyalamadan onunla aynı işlevi gerçekleştiren yeni bir alet veya yazılım yapmaya çalışır ve sıklıkla bir şeylerin (örneğin; makine veya mekanik alet, elektronik bileşen, yazılım programı gibi) parçalarına ayrılması ve çalışma prensiplerinin detaylı şekilde analizini içerir. Tersine Mühendislikte kullanılmakta olan dört araç mevcuttur. Bunlar Hata Ayıklayıcı, Hata Enjeksiyon Araçları, Tersine Çevirici (Disassembler) ve Tersine Derleyicidir (Decompiler).

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

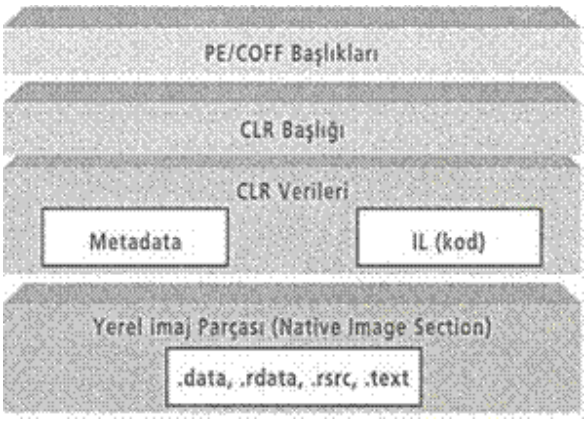
Beyaz kutu analizi kaynak kodun analizi ve anlaşılmasını içerir. Eğer ikili kod geri derlemeyle kaynak koda dönüştürülür ve sonra incelenirse beyaz kutu analizi göz önüne alınmalıdır. Bu çalışmanın da temelini oluşturan beyaz kutu analizi programlama ve uygulama hatalarının belirlenmesinde çok etkilidir (Hoglund ve McGraw, 2004).

Siyah kutu analizi değişik giriş değerlerine göre koşan bir programın incelenmesiyle ilgilenir. Bu analiz çeşidinde tümü herhangi bir yerde koşan ve giriş isteyen bir programa ihtiyaç duyulur (Hoglund ve McGraw, 2004).

Gri kutu analizi beyaz kutu analiziyle siyah kutu giriş testinin birleşiminden oluşur. Gri kutu analizine en basit örnek olarak bir hata ayıklayıcı programının içinde koşan bir hedef programa değişik girişler verilerek gözlemlenmesidir (Hoglund ve McGraw, 2004).

3. .NET Dosya Yapısı ve IL

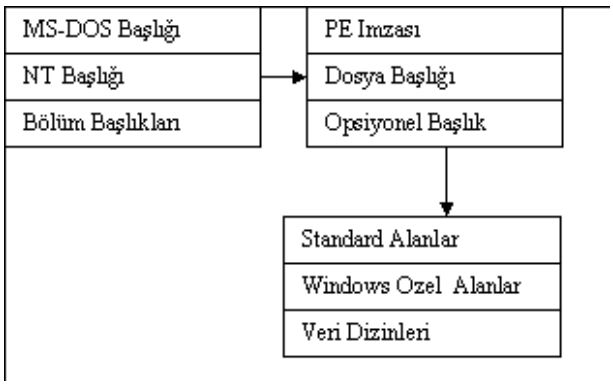
.NET dosyaları COFF (Ortak Nesne Dosya Formatı) dosya formatının bir türevi olan birer PE (Portatif İcra-edilebilir) dosyalarıdır (MICROSOFT, 2004). PE dosyalarının genel yapısı Şekil 1'de gösterilmiştir.



Şekil 1: PE Dosya Yapısı

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. Standart bir Windows PE dosyası bir MS-DOS başlığında başlayan, bir PE başlığıyla devam eden, isteğe bağlı bir başlık (Opsiyonel 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. CLR'ı desteklemek için, Microsoft PE/COFF dosya formatını metadata ve IL kodunu kapsayacak şekilde genişletmiştir (Thai ve Lam, 2003).

3.1. PE Başlığı



Şekil 2: PE Başlığı

Her PE dosyası MSDOS başlığıyla başlar. MS-DOS başlığındaki en önemli alan PE imzasına atlamak için gerekli olan ofset değerini içeren 0x3c adresindeki e_lfanew alanıdır. Bu alandaki ofset değerine atlanıldığında dosyanın PE dosyası olup olmadığını gösteren PE imzası değeri okunur. Eğer buradaki değer "PE\0\0" ise dosyanın PE dosyası olduğunu gösterir. İmzadan sonra standart bir coff dosya başlığı vardır. Her imaj dosyası yükleyiciye (loader) bilgileri sağlamak için opsiyonel başlıklara sahiptir. İmaj dosyası için bu başlık gereklidir. Obje dosyalarında ise bu başlığa gerek yoktur. Bizim için Opsiyonel başlıktaki en önemli yapı veri dizinleridir Çünkü 15. veri dizini CLR başlığının boyutunu ve konumunu gösterir. COFF dosya formatının standart yapısı "Microsoft Portable Executable and Common Object File Format Specification" da belirtilmiştir (MICROSOFT, 2004).
CLR başlığı çalışma zamanına özgü veri kayıtlarını ve çalışma zamanına özel diğer bilgilerin hepsini içerir. Bu başlık imajın sadece okunabilir ve paylaştırılabilir bölümünde konumlandırılmalıdır. Metadata verilerini okuyabilmek için CLR başlığındaki 4. alan olan MetaData alanındaki konum ve boyut verisine ihtiyaç duyulur. Buradaki konum verisi Metadata başlığına atlayabilmek için gerekli olan ofset değerini içerir. CLR başlığının yapısı Tablo 1'de gösterildiği gibidir.

Tablo 1: CLR Başlığı

Ofset	Boyut	Alan İsmi
0	0	BSJB
4	2	Major
6	2	Minor
8	4	
12	4	Length
16	m	Versiyon
16+m		
X	2	Flag
X+2	2	Akımlar
X+4		Akım Başlığı

Akım (stream) başlığının konumuna atlandıktan sonra akım sayısı kadar akım başlık bilgileri alınır. Her akım başlık bilgisi akımın konumunu içeren ofset değeri, akımın boyutu ve akımın isminden oluşur. Metadata'da maksimum 5 akım vardır. Bu akımlar;

#~ : Metadata tablolarını barındırır.
#string : Karakter dizilerini tutar
#guid : Guidleri barındırır
#Blob : İmzalar çözülür
#us : Kullanıcı tanımlı stringleri tutar.

Metadata başlığı hangi metadata tablolarının kullanıldığını, hangi tablonun kaç satırı olduğuna dair bilgileri tutar. Tablodaki "Akım Başlığı" alanı metadata başlığının konumunu gösterir. Metadata başlığının yapısı Tablo 2'de gösterilmiştir.

Tablo 2: Metadata Başlığı

Ofset	Boyut	Alan İsmi
0	4	Reserved
4	1	Major Version
5	1	Minor Versiyon
6	1	Heapsize
7	1	Reserved
8	8	Valid
16	8	Sorted
24	4*n	Rows
24+4*n		Tables

Metadata'da 43 tablo çeşidi vardır ve ilk tablonun ofset değeri 24 tür. Bu değerden sonra her tablo için 4 bayt eklenerek ilgili tablonun satır sayısını alacağımız ofset değeri bulunabilir. Örneğin, ilk tablonun satır sayısını alacağımız ofset değerini okuyabilmek için metadata'da 24. bayta gelinir. 24. bayttan itibaren 4 baytlık satır sayısı değeri okunur. 2. tablonun satır sayısını hesaplamak için 28. bayta gelinir ve buradan 4 baytlık veri okunur.

Metadata tabloları satırlardan oluşur. Örneğin MetotDef tablosuna programda oluşturulan her metot için bir satır eklenir. Param tablosuna her parametre için bir satır eklenir.

4. Metotların Geri Dönüşümleri

Metotların geri dönüşümü için metadata tabloları arasındaki ilişkinin çok iyi çıkartılması gerekmektedir. Metadata tablolarına örnek verecek olursak; MethodRef tablosu programda kullanılan başka assemblylerdeki metotları tutar. MethotDef programda bizim oluşturduğumuz metotların bilgilerini tutar. TypeRef tablosu programda kullanılan sınıf, tür referanslarını tutar. TypeDef tablosu ise programda oluşturulan sınıf ve türleri tutar. Constant tablosu sabitleri tutar. Param tablosu parametre bilgilerini tutar. Bir metodu geri dönüştürmek için bu tablolar ve akımlar arasındaki ilişkilerin bilinmesi gerekmektedir. Örneğin Metot tablosunda metodun parametreleri için param tablosuna bir indeks değeri vardır. Param tablosunda ise parametrenin ismi için string akımına bir indeks değeri vardır. Metot geri dönüşümü için metot tablosunun yapısı aşağıdaki gibidir.

Tablo 3: MethodDef Tablosu

Offset	Boyut	Alan İsmi
0	4	RVA
4	2	IMPL Flags
6	2	Flags
8	2	Name
10	2	Signature
12	2	ParamList

Burada paramlist metodun parametrelerini belirtir ve param tablosuna bir indeks değeri içerir. RVA ise metot gövdesinin adresini (kod bloğu) gösterir. RVA ile belirtilen ofsete atlanıldığında metot gövdesinin başlığı başlar. Metot bu başlığa göre çözümlenir. Metot başlığındaki ilk baytın en anlamlı 3 biti ne tür başlığın mevcut olduğunu belirtir. Eğer izin verilen yerel değişkenler, istisnalar, ekstra veri bölümleri yoksa ve işlem yığını 8 bayttan daha fazlasına gereksinim duymuyorsa TINY format kullanılır. Bu koşullardan herhangi biri sağlanmadığında ise FAT format kullanılır (File Format Spec, 2000).

TINY formatın yapısı aşağıdaki gibidir:

```
typedef struct IMAGE_COR_ILMETHOD_TINY
{
   BYTE Flags_CodeSize;
} IMAGE_COR_ILMETHOD_TINY;          (1)
```

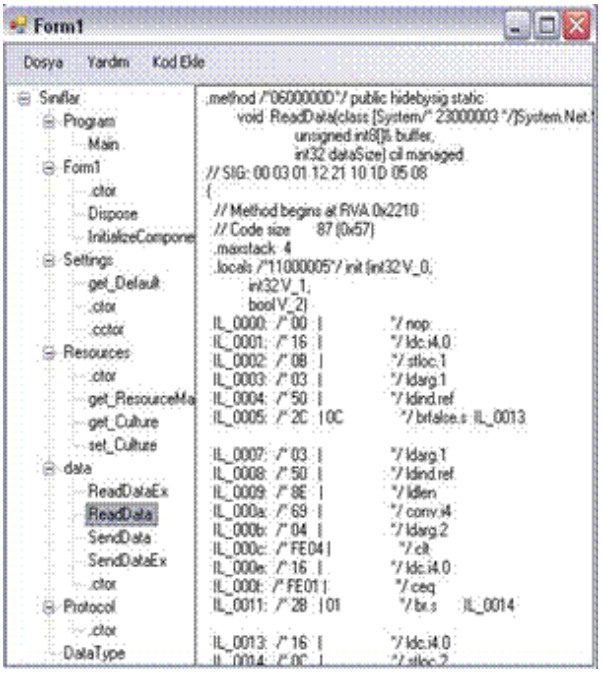
(1) de gösterilen yapıda görüldüğü gibi TINY format başlığı sadece kod bölümünün boyutunu gösteren bir baytlık veriden oluşur.

FAT formatın yapısı ise Tablo 4'te verilmiştir.

Tablo 4: FAT Format

Ofset	Boyut	Alan
0	12 bit	Flags
12 bits	4 bit	Size
16 bit	16 bit	MaxStack
4	4	CodeSize
8	4	LocalVarSigTok

Bu başlık bilgisinden sonra metodun kod bölümünün bilgileri alınır. Kod bölümünün boyutu CodeSize alanında belirtilmiştir. Başlık bilgileri alındıktan sonra dosyadan sırasıyla birer baytlık veriler okunur. Bunlar IL (intermediate language) komutlarıdır. Komutlar geri dönüştürülürken komutun türü önemlidir. IL'de değişik tür komut türleri vardır; örneğin, InlineMethod, InlineNone, InlineString gibi. Şimdi InlineString işlenen türünden bir komutun çözümlenmesini inceleyelim. "ldstr" komutu InlineString işlenen türünden bir komutu gösterir ve 114 değeriyle gösterilir. Bu işlenen türünden bir komut kendinden sonra kod dizisinde 4 baytlık bir değer alır. Bu 4 baytlık değerin en anlamsız 3 baytı us (user string) akımından kaç baytlık değer alınacağını ve başlangıç indeksini gösterir. Şekil 3'te örnek bir programın bir fonksiyonunun geri dönüştürülmüş hali gösterilmiştir.



Şekil 3: Metodun koda dönüştürülmüş hali

5. Dosya İşlemlerini Algılama

Bu işlem için öncelikle aşağıda verilen C# ve bunun karşılığı olan IL kodunu inceleyelim:

```
FileStream fs = new FileStream(“deneme.txt”,
     FileMode.Open, FileAccess.ReadWrite);                    (2)

ldstr      "deneme.txt"
ldc.i4.3
ldc.i4.3
newobj     instance void System.IO.FileStream::.ctor        (3)
```

(2)’ de gösterilen örnek C# kodunda deneme.txt adlı bir dosya açılıyor. Dosya oluşturma işlemi 3 parametre alıyor. Birincisi dosyanın adı karakter dizisi tipinde, ikinci parametre dosya açma modu, son parametre ise dosyaya erişim yetkisini gösteriyor. (3)’te gösterilen IL kodunda ise bu durum biraz daha farklı biçimdedir. IL’de işlemler işlem yığınında yapılır. Newobj ile bir metot çağrılmadan önce metodun parametreleri yığına yüklenmelidir. Bu yüzden örnek IL kodunda öncelikle ldstr ile dosya adı, sonra ldc.i4.3 ile 3 sabiti ve tekrar ldc.i4.3 ile 3 sabiti yığına itiliyor. Bunlar da sırasıyla FileStream objesinin aldığı parametreleri gösterir.

(3)’teki örnek IL kodundan da anlaşıldığı gibi dosya işleminin algılanması için buradaki “newobj” komutunun yakalanması gerekmektedir. “newobj” komutu metot gövdesinde 115 değeriyle temsil edilir ve InlineMethod türünde bir emirdir. Bu emir kendinden sonra 4 baytlık bir işaret alır. İlk bayt tabloyu gösterir. Kalan 3 bayt ise tablodaki satırı gösterir. Dosya işlemi olan fonksiyonlarda newobj’den sonra System.IO.FileStream::. ctor fonksiyonu çağrılır. Bu fonksiyon MemberRef (MethodRef) tablosunda saklanır. Bu tablonun yapısı Tablo 5’de verilmiştir.

Tablo 5: MethodRef Tablosu

Ofset	Boyut	Alan İsmi	Alan Tanımı
0	2	Class	Sınıf
2	2	Name	isim
4	2	Signature	

MemberRef tablosu 10 değeri ile gösterilir. Dosya işleminin oluşması için “newobj” komutundan sonra işaretin ilk baytının değeri 10 olması gerekir. Fakat henüz bu dosya işlem olduğunu göstermez. Bir sonraki aşamada ise MemberRef tablosundan hangi sınıfın hangi fonksiyonunun çağrıldığının bulunmasıdır. Çünkü MemberRef tablosunda diğer sınıflardan ve türlerden kullanılan fonksiyonlar da tutulur. İçinde birden fazla fonksiyon olması muhtemeldir. Burada önemli olan newobj ile dosya işlem fonksiyonu çağrılıp çağrılmadığını algılamaktır.

İşlaretin kalan bitleri, bulunan tablodaki (MemberRef) hangi satırdan değer alınacağını gösteren bir indeks değeridir. MemberRef tablosunda bulunan satıra gidildiğinde ilk önce satırdaki Class alanının çözülmesi gerekmektedir. MemberRef tablosundaki Class değerinin ilk 3 biti hangi tablodan değer alınacağını gösterir. Sonuç olarak dosya işleminin olması için bir tür referansının alınması ve buradaki değerin TypeRef tablosunu göstermesi gerekmektedir. Class alanının kalan bitleri ise bulunan tabloda (TypeRef tablosu) hangi satırdan değer alınacağını gösterir .

Sonraki aşamada ise TypeRef tablosundan bulunan satırdaki değerin okunması gereklidir. Bu işlem sonucunda dosya işleminin gerçekleşmesi için “System.IO.FileStream” türünün kullanıldığının bulunması gerekir. TypeRef tablosunun yapısı Tablo 6’da gösterilmiştir.

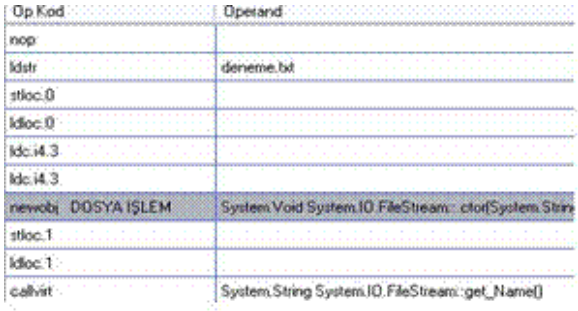
Tablo 6: TypeRef Tablosu

Ofset	Boyut	Alan İsmi
0	2	Resolution Scope
2	2	Name
4	2	Namespace

Dosya işlemi olması için bulunan satırdaki “Name” alanının değeri System.IO.FileStream olmalıdır. Son olarak ise MemberRef tablosunun “Name” alanında .ctor değerinin elde edilmesiyle dosya işlemi algısı tamamlanmış olur.

Dosyanın yazıldığını gösteren kalan son aşama ise elde ettiğimiz metodun parametrelerinin bulunmasıdır. Bu parametreler blob akımından alınır. Blob akımı metadata başlığı ilk okunduğunda karakter dizisine dönüştürülür. Bu dizinin işaretle belirtilen elemanı alınarak söz konusu metodun kullandığı parametrelere ulaşılır. Parametrelerden FileAccess parametresi kontrol edilir, eğer Write veya ReadWrite değerlerinden birini almışsa dosya yazma için açılmıştır şeklinde yorumlanır ve istenirse sonraki aşamada yedekleme kodu eklenir.

Aşağıda örnek bir program için analiz sonucu gösterilmiştir.



Şekil 4: Dosya İşlem algılaması

Dosya yedekleme işlemi için fonksiyona newobj emrinden sonra sırasıyla ldloc, callvirt, ldstr, call emirleri eklenir. callvirt komutu OperandType.InlineMethod türündedir. Bu emrin boyutu 5 bayttır. Bu emir metod gövdesine eklenirken, öncelikle MemberRef tablosuna bir satır eklenir. Bu satır get_Name() fonksiyonunu tanımlar. Ayrıca metod başlığındaki kod alanı 5 arttırılır. Metot gövdesinde ise ilgili emir sırasına ilk önce 111 eklenir. (111 calvirt emrini tanımlar.) Daha sonra ise bu emrin alacağı 4 baytlık değer eklenir. Bu 4 baytın ilk baytının değeri 10’dur. 10 MemberRef tablosunu gösterir. Kalan 3 bayt ise metodun MemberRef tablosundaki satır numarasını gösterir.. Elde edilen tablo ve indeks değeri

birleştirilerek 4 bayt olarak dosyaya yazılır.

Ldstr emri için önce us akımına ilgili karakter dizisi eklenir. Sonra ise 114 (ldstr) değeri metot gövdesine yazılır. Son olarak ise 4 bayt karakter dizisini alabilmemiz için gerekli olan değer metot gövdesine eklenir.

Sonraki aşama ise tüm metotların RVA'larının yeniden hesaplanması ve metot tablosunda ilgili RVA değerlerinin yeniden yazılmasıdır. Bunun nedeni ise tablolara yeni satırlar eklenmesi, us akımına yeni değerler eklenmesi ve metot gövdelerine yeni kodlar eklenmesidir. Yeni RVA değerleri her

Son olarak ise imaj başlığındaki bölüm başlıklarının RVA'ları yeniden hesaplanır. Ayrıca dataDirectory'deki veri dizinlerinin de RVA'ları yeniden hesaplanır.

Tüm bu elde edilen değerlerin sonuçları dosyaya yeniden yazılarak, imaj dosyasında değişiklik yapılır. Şekil 5 'de yedekleme kodu eklenmiş örnek bir program gösterilmiştir.

Op Kod	Operand
nop	
ldstr	deneme.txt
stloc.0	
ldloc.0	
ldc.i4.3	
ldc.i4.3	
newobj DOSYA İŞLEM	System.Void System.IO.FileStream::.ctor(System.Strin
stloc.1	
ldloc.1	
callvirt	System.String System.IO.FileStream::get_Name()

Şekil 5: Yedekleme kodu eklenmiş dosya

6. Kayıt İşlemlerini Algılama

Bu algılama işlemi de dosya işlemi algılamaya çok benzerdir. Sadece değişen newobj komutu yerine ldsfld komutunun analiz edilmesi yeterlidir. Ldsfld komutu InlineField türünden bir komuttur. InlineField türündeki bir komutun çözümlenmesi InlineMethod türündeki bir komutun çözümlenmesine benzer. Burada da işaretin ilk baytı tablo'yu, geri kalan baytlar ise tablodaki satır numarasını gösterir. Bu işlem için de yine MemberRef tablosuna gidip değerler alınır. MemberRef tablosunda sınıf ismi kontrol edilir. Eğer sınıf isminde Microsoft.Win32.RegistryKey varsa bu kayıt işlemi olarak yorumlanır. Şekil 6'da verilen örnekte bir fonksiyondan alınmış analiz sonucu görülmektedir.

Op Kod	Operand
nop	
ldsfld REGISTERY İŞLEMİ	Microsoft.Win32.RegistryKey Microsoft.Win32.Registry::Local
stloc.0	
ret	

Şekil 6: Kayıt işlemi algısı

7. Sonuçlar

Bu çalışmanın amacı, .NET platformunda yazılmış programların koda geri dönüştürülmesi ve imaj dosyasının bazı bölümlerine yeni eklentiler yaparak arzu edilen işlemlerin gerçekleştirmesini saklamaktır. Programlardaki en önemli yapıların koda dönüştürülmesi tamamlanmıştır. Import tablosundaki bilgiler okunmuş ve hangi sistem kaynaklarının kullanıldığı tespit edilmiştir. Dosya başlıkları başarılı bir şekilde yorumlanarak program hakkında detaylı bilgilerin elde edilmesi gerçekleştirilmiştir. Metadata ve akımlar da başarılı bir şekilde okunarak metadata tablolarıyla akım tabloları doğru bir şekilde elde edilmiştir. Programdaki değişkenler, sabitler, sınıflar, ad uzayları, metotlar, metot parametreleri belirlenerek metot içerikleri yeniden oluşturulmuştur. İmaj dosyasındaki dosya işlemlerinin algılanması tamamlanmıştır. Bir metot içerisinde bir dosya yazılmak için açılmışsa bu metoda yedekleme kodu eklenmesi gerçekleştirilmiştir. Bunun sonucu olarak önemli dosyalar için bir geri dönüşüm mekanizması tesis edilmiştir.

8. Kaynaklar

[1] Hoglund, G. ve McGraw, G., Exploiting Software How to Break Code, Addison Wesley, February 17, 2004

[2] Thai, T. ve Lam,H.Q., .NET Framework Essentials, O'Reilly, 2003

[3] MICROSOFT, Microsoft Portable Executable and Common Object File Format Specification, MICROSOFT, 2006

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

[5] ECMA and ISO/IEC C# and Common Language Infrastructure Standards, Microsoft, 2003.

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

[7] 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.

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

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

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

[11] http://www.mono-project.com/Main_Page, Mono, 13 Nisan, 2007.