

.NET Çalıştırılabilir Dosyalarının Statik Analizi ile

Dosya Sistemi Güvenliğinin Sağlanması

Hüseyin Pehlivan, Salih Aras, Mehmet Emin Tenekeci

Karadeniz Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü

pehlivan@ktu.edu.tr, arass@dsi.gov.tr, emintenekeci@ktu.edu.tr

Özet: Bu çalışmada, .NET çevrelerinde geliştirilen çalıştırılabilir dosyaların statik analizi gerçekleştirilmiştir. Bu statik analiz işlemi PE (Portable Executable — Portatif İcra edilebilir) dosya yapısı, CLR (Common Language Runtime — Ortak Dil Çalışma Zamanı) yapısı ve metadata tablolarının okunmasına dayandırılmaktadır. Okunan bu verilerden, dosya oluşturma, dosya açma veya dosya silme gibi dosya sistemi işlemleri algılanmıştır. Bu işlemlerin dosya sistemine zarar verebileceği düşünülerek, çalıştırılabilir dosya, üzerinde işlem yapılan dosyanın yedeğini alacak şekilde düzenlenmiş ve yeniden oluşturulmuştur. Burada mevcut çalıştırılabilir dosyanın yapısı değiştirilmiştir ve buna bağlı olarak PE yapısı, CLR yapısı ve metadata tabloları yeniden düzenlenmiştir. Böylece dosya sistemine zarar verebilecek uygulamaların vereceği zararlar telafi edilebilecektir.

Abstract: This study deals with static analysis of the executable files, developed in .NET environments. Static analysis is based on reading PE (Portable Executable) file structure, CLR (Common Language Runtime) structure and Metadata tables. File system operations such as creating, opening and deleting are detected using these data. These operations can be damage file system. Therefore executable files are modified and recreated for backup used files. In this wise structure of the executable files are modified. Thus, PE structure, CLR structure and metadata tables are reorganized. Consequently, damage caused by applications which is harmful to the file system can be recovered.

Anahtar Kelimeler: Statik analiz, IL, Metadata Tabloları, Dosya Sistemi Güvenliği.

1. Giriş

Son zamanlarda kötü niyetli uygulamaların artması ile bilgisayar güvenliği veya dosya sistemi güvenliği konusuna ilgi artmaktadır. İnternetten indirilen veya başka bir şekilde elde edilen uygulamaların güvenliği veya güvenilirliği belli değildir. Bu uygulamalar sistemimizi tehdit edebilecek işlevlere sahip olan virüs, truva atı veya solucan gibi zararlı uygulamalar olabilir. Kullanacağımız bu uygulamanın güvenliğinin bir şekilde test edilebilmesi gerekmektedir[1]. Bu test işlemi programların analiz edilmesi ile gerçekleştirilebilir. Analizin temel nedeni bu tip programlar için ulaşılabilir kaynağın olmasıdır. Bu programların nasıl davrandığını an-

lamının tek yolu onları analiz ederek çalışma prensiplerini belirlemektir. Analiz etmek için dinamik ve statik olmak üzere iki farklı yöntem kullanılmaktadır.

Statik analiz yöntemi programları çalıştırmadan programların incelenmesine dayanır[2]. Bu yöntem en güvenli test yöntemidir. Çünkü program çalışmadan test edildiğinden dolayı sisteme zarar verme ihtimali söz konusu değildir. Statik analiz programın kaynak kodu veya çalıştırılabilir dosya üzerinden yapılabilir. Analiz edilmek istenen program ulaşılabilir yerde olması gerekmektedir. Programın çalıştırılabilir dosyasına erişim imkânımız yoksa statik analiz yapmamız mümkün değildir. As-

ında bu yöntem beyaz kutu analiz yönteminde kullanılan metotlardan birisidir.[3]

Dinamik analiz yöntemi ise programları çalıştırarak programların incelenmesine dayanır. Bu yöntemde programın değişik giriş değerlerine göre değişik davranışları belirlenir. Ancak programın icrası esnasında sisteme verebileceği zararların göz ardı edilmemesi gerekmektedir. Bu sebepten dolayı güvenlik amaçlı olarak programların dinamik analizleri sanal makineler üzerinde gerçekleştirilebilir. Dinamik analizler uygulama olarak statik analizlerden daha basit ve daha etkili olmakla beraber daha tehlikelidir. Statik analizde belirlenemeyen bazı zararlı kodlar dinamik analiz ile belirlenebilir. Bu nedenle statik ve dinamik analiz yöntemini kullanan bazı çalışmalar vardır[4]. Ayrıca kaynak koda veya çalıştırılabilir dosyaya erişimimizin mümkün olmadığı durumlarda dinamik analiz yöntemini kullanmaktan başka çaremiz olmamaktadır. Bu yöntem siyah kutu analizinin bir çeşididir.[3]

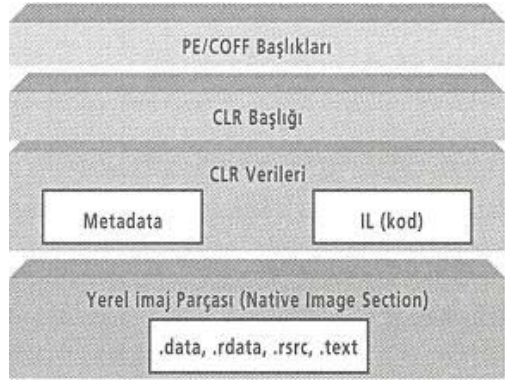
Bu çalışmada dosya sistemine olabilecek olası tehditlerin algılanmasına ve oluşabilecek problemlerin telafi edilmesi ile ilgilenilmiştir. Burada .NET program dosyalarının PE dosya yapısı, CLR yapısı ve metadata tabloları incelenerek yapmış olduğu işlevler belirlenmiştir. Bu işlevlerden dosya sistemine zarar verebileceği muhtemel dosya işlemleri araştırılmıştır. Bu muhtemel dosya işlemleri icrası esnasında üzerinde işlem yapılan dosyanın bir kopyası sistemde belirlenen belli bir yere kopyalayacak şekilde çalıştırılabilir dosyanın yapısı değiştirilmiştir. Bu şekilde oluşabilecek problemlerde veri kaybından kurtulmuş olunacaktır.

Bu çalışmaya benzer olarak Unix işletim sisteminde programları dinamik olarak izleyip dosya sisteminde oluşabilecek problemleri telafi edebilmek için işlem yapılan dosyayı yedekleyen çalışmalarda yapılmıştır [5,6]. Bu uygulamalarda çalışan programların yapmış oldukları sistem çağrılarını belirlenerek dosya sistemine

zarar verebilecek işlem icra etmeleri durumunda yedekleme işlemi gerçekleştirilmiştir.

2. .NET Dosya Yapısı

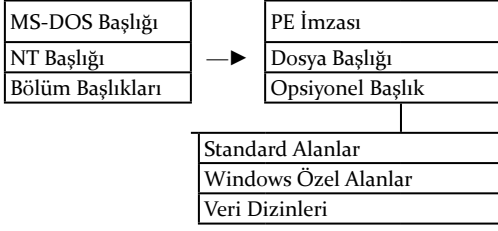
.NET çalıştırılabilir dosyaları COFF (Common Object File Format -Ortak Nesne Dosya Formatı) yapısının bir türevi olan PE dosya formatına uymak zorundadır[7]. PE dosyalarının genel yapısı Şekil 1’ de gösterilmiştir.



Şekil 1: PE Dosya Yapısı

Standart bir Windows PE dosyası bir MS-DOS başlığında başlar. PE başlığıyla devam eder. İsteğe bağlı bir başlık (Opsiyonel Başlık) tarafından takip edilir. Son kısımda NETteki .text, .data, .rdata ve .rsrc ayrımları dahil bazı yerel imaj parçalarına doğru giden bazı parçalara bölünmüştür. Microsoft PE/COFF dosya formatı CLR’yi desteklemek için metadata ve IL kodunu kapsayacak şekilde genişletilmiştir[8].

Şekil 2’ de görüldüğü gibi her PE dosyası MSDOS başlığıyla başlar. MS-DOS başlığındaki en önemli alan PE imzasına atlamak için gerekli olan ofset değerini içeren 0x3 c 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.



Şekil 2: PE Başlığı

İmzadan sonra standart bir coff dosya başlığı vardır. COFF dosya formatının standart yapısı [7] da belirtilmiştir.

Her çalıştırılabilir dosya yükleyiciye (loader) bilgileri sağlamak için opsiyonel başlıklara sahiptir. Çalıştırılabilir dosyalar için bu başlık gereklidir. Opsiyonel başlıktaki en önemli yapı ise 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. Bu başlık çalıştırılabilir dosyanın sadece okunabilir ve paylaşılabilir bölümünde konumlandırılmalıdır[9]. 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.

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

Tablo 1: CLR Başlığı

Akım 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 tabloları
- #string : Genel dizgiler
- #guid : Guidler
- #Blob : İmzalar
- #us : Kullanıcı tanımlı dizgiler

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.

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

Tablo 2: Metadata Başlığı

Metadata’ da 43 farklı tablo vardır. İlk 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 tablolarında programda kullanılan veriler depolanır. Bunların her birisi için tablolara satırlar eklenir. Örneğin MetotDef tablosuna programda oluşturulan her metot için bir satır eklenir. Param tablosuna her parametre için bir satır eklenir. Bu tablolardaki satırlardan programla ilgili verilere erişim sağlanır.

3. Metotların Belirlenmesi

Programların davranışlarını ve işlevlerini belirlemede icra edecekleri metotların belirlenmesi çok önemlidir. Bir önceki bölümde bahsedildiği gibi metotlarla ilgili bilgiler metadata tablolarında tutulmaktadır. Bu nedenle metotların belirlenmesi için metadata tabloları arasındaki ilişkilerin çok iyi belirlenmesi gerekmektedir. Tablolar arasındaki ilişkiler doğru bir şekilde belirlendikten sonra istenen verilerin belirli index değerlerine göre tablolar arasında ilerleyerek elde edilebilir. Ayrıca hangi tablodan hangi bilgilerin ne şekilde elde edileceğinin bilinmesigerektilmektedir. Metadata tablolarına örnek verecek olursak; MethodRef tablosu programda kullanılan başka assemblylerdeki metotları tutar. MethodDef 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. Bu tablolar kullanılarak metotlarla ilgili tüm verilere erişilebilir. Mesela, Tablo 3’ de görüldüğü gibi metot tablosunda metodun parametreleri için param tablosuna bir indeks değeri vardır. Param tablosunda ise parametrenin ismi için string akımına bir indeks değeri vardır.

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

Tablo 3: MethodDef Tablosu

Burada paramlist metodun parametrelerini belirtir ve param tablosuna bir indeks değeri içerir. RVA (Relative Virtual Address - Bağlı Sanal Adres) ise metot gövdesinin adresini gösterir. RVA ile belirtilen ofsete atlanıldığında metot gövdesinin başlığı başlar. Metot bu başlığa göre çözülür. Metot başlığındaki ilk baytın en anlamlı 3 biti ne tür başlığın mevcut olduğu-

nu belirtir. Eğer izin verilen yerel değişkenler, istisnalar, ekstra veri bölümleri yoksa ve işlem yığını 8 bayttan daha fazlasına gereksinim duymuyorsa TINY format kullanılır. TINY format başlığı sadece kod bölümünün boyutunu gösteren bir baytlık veriden oluşur. Bu koşullardan herhangi biri sağlanmadığında ise tablo 4’ gösterilen FAT format kullanılır[9].

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

Tablo 4: FAT Format

MethodDef tablosundaki RVA ile belirtilen ofset değerinin göstermiş olduğu yerdeki 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 çözülmesini 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 değer en anlamsız 3 baytı us (user string) akımından kaç baytlık değer alınacağını ve başlangıç indeksini gösterir.

Bütün kod bölümü okunarak IL komutları şeklinde programın yapmış olduğu işlemler elde edilir. Doğru bir şekilde kod bölümünün elde edilmesi için belirtilen formata dikkat edilmesi gerekir.

4. Dosya Sistemini Etkileyecek İşlemlerin Algılanması

Dosya sistemini etkileyecek işlemlerini algılanması işlemi metadata tablolarının okunması

sonucu elde edilen IL kodu üzerinden gerçekleştirilecektir. IL kodu içerisinde dosya sistemi ile ilgili kod parçalarının incelenmesi gerekmektedir. Bu nedenle dosya işlemi gerçekleştiren örnek bir C# kodu ve bunun karşılığında oluşturulan IL kodu incelenmelidir. Aşağıdaki kod parçasını bu amaç için kullanabiliriz.

```
FileStream fs = new FileStream("deneme.txt",  
    FileMode.Open, FileAccess.ReadWrite); (2)  
ldstr "deneme.txt"  
ldc.i4.3  
ldc.i4.3  
newobj instance void System. O.FileStream::.ctor (3)
```

(2)' de gösterilen kod parçasında deneme.txt adlı bir dosya okuma ve yazma modunda açılıyor. Böyle bir fonksiyon 3 parametre alıyor. Birincisi dosyanın adı karakter dizisi tipinde, ikinci parametre dosya açma modu, son parametre ise dosyaya erişim yetkisini gösteriyor. (3)' te gösterilen C# kodunun karşılığında IL kodunda ise bu durum biraz daha farklı biçimdedir. IL' de işlemler işlem yığnında yapılır[10]. Newobj ile bir metod 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 metod 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 MethodRef tablosunda saklanır. Bu tablonun yapısı Tablo 5'de verilmiştir.

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

Tablo 5: MethodRef Tablosu

MemberRef tablosu 10 değeri ile gösterilir. Dosya işleminin oluşması için "newobj" komutundan sonra işaretin ilk baytının değeri 10 olması gerekir. Fakat henüz bu dosya işlemi 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.

İşaretin kalan bitleri, bulunan tablodaki (MemberRef) hangi satırdan değer alınacağını gösteren indeks değeridir. MemberRef tablosunda bulunan satıra gidildiğinde ilk önce satırdaki Class alanının çözülmesi gerekir. MemberRef tablosundaki Class değerinin ilk 3 biti hangi tablodan değer alınacağını gösterir. Sonuç olarak dosya işleminin olması için tür referansının alınması ve buradaki değerinin 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 değerinin okunması gereklidir. Bu işlem sonucunda dosya işleminin gerçekleşmesi için "System.IO.FileStream" türünün kullanıldığının bulunması gerekir. TypeRef tablosunun yapısı Tablo 6'da gösterilmiştir.

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

Tablo 6: TypeRef Tablosu

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 işaretlerle 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.

5. Programın Güvenli Hale Getirilmesi

Program içerisindeki dosya işleminin belirlenmesinden sonra bu işlemin meydana getirebileceği muhtemel zararları telafi etmek için üzerinde işlem yapılan dosyanın yedeğini almamız gerekmektedir. Bu nedenle çalıştırılabilir dosya, üzerinde işlem yapılan dosyanın yedeğini alabilecek şekilde değiştirilir. Elimizde çalıştırılabilir dosyanın kaynak kodu olmadığından IL kod üzerinden değişiklik yapılacak ve buna uygun olarak PE yapısı ve metadata tabloları yeniden düzenlenmesi gerekir.

Dosya yedekleme işlemi için fonksiyona newobj emrinden sonra sırasıyla ldloc, callvirt, ldstr, call emirleri eklenir. callvirt komutu OperandType.InlineMethod türündedir. Bu emrin boyutu 5 bayttir. Bu emir metod gövdesine eklenirken, öncelikle MemberRef tablosuna bir satır eklenir. Bu satır get_Name() fonksiyonunu tanımlar. Ayrıca metod başlığındaki kod alanı 5 artırılır. Metod 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 tablosun-

daki satır numarasını gösterir. Elde edilen tablo ve indeks değeri birleştirilerek 4 bayt olarak dosyaya yazılır. ldstr emri için önce us (user string) akımına ilgili karakter dizisi eklenir. Sonra ise 114 (ldstr) değeri metod gövdesine yazılır. Son olarak ise 4 bayt karakter dizisini alabilmemiz için gerekli olan değer metod gövdesine eklenir. Sonraki aşama ise tüm metodların RVA değerlerinin yeniden hesaplanması ve metod tablosunda ilgili RVA değerlerinin yeniden yazılmasıdır. Bunun nedeni ise tablolara yeni satırlar eklenmesi, us akımına yeni değerler eklenmesi ve metod gövdelerine yeni kodlar eklenmesidir.

Yeni RVA değerleri her fonksiyon için yeniden hesaplanır. Son olarak ise dosya başlığındaki bölüm başlıklarının RVA'ları yeniden hesaplanır. Ayrıca dataDirectory'deki veri dizinlerinin de RVA'ları yeniden hesaplanır.

Bütün elde edilen değerlerin kaydedilmesi ile yeni çalıştırılabilir dosya oluşturulur.

5. Sonuçlar

Bu çalışmada .NET ortamında yazılmış çalıştırılabilir dosyaların PE yapısı, CLR yapısı ve metadata tabloları okunmuştur. IL kodu oluşturulup statik olarak analizi gerçekleştirilmiştir. Programın yaptığı dosya sistemi işlemi bu analiz sonucu belirlenmiştir. Muhtemel dosya sistemine verilebilecek zararları telafi etmek için üzerinde işlem yapılan dosyanın yedeği alınabilecek şekilde yeniden düzenlenmiştir. Buna bağlı olarak metadata tabloları yeniden oluşturulup kaydedilmiştir.

Gelecek çalışmada ise programlarda güvenliği tehdit edebilecek diğer aktivitelerin belirlenip, bu aktivitelere uygun önlemler alınması üzerinde durulacaktır. Ayrıca .NET programlarının statik incelenmesi ile beraber dinamik olarak da izlenilerek dosya sistemi güvenliği sağlanabilir.

Kaynaklar

- [1] Vigna G., Christodorescu M., Jha S., Maughan D., Song D., Wang Eds C. ‘Static Disassembly and Code Analysis’, *Advances in Information Security*, Springer, 2007
- [2] Chess B. ve West J., ‘Secure Programming with Static Analysis’, Addison Wesley, New Jersey, 2007
- [3] Hoglund G. ve McGraw G., ‘Exploiting Software How to Break Code’, Addison Wesley, February 17, 2004
- [4] Cova M., Felmetsger V., Balzarotti D., Jovanovic N., Kruegel C, Kirda E, Vigna G, ‘Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications’, Oakland, May 2008
- [5] Holyer, I., and Pehlivan, H., “A recovery mechanism for shells”, *The Computer Journal*, Vol. 4, 2000, No. 3, 1-9.
- [6] Hüseyin Pehlivan, Mehmet Emin Tenekeci, *Unix İşletim Sistemleri İçin Akıllı Geri Dönüşüm Kutusu Tasarımı Ve Gerçekleştirilmesi*, Eleco2006 Bursa
- [7] MICROSOFT, ‘Microsoft Portable Executable and Common Object File Format Specification’, MICROSOFT, 2006
- [8] Thai T. ve Lam H.Q., ‘.NET Framework Essentials’, O’Reilly, 2003
- [9] ‘File Format Spec.’, <http://www.ssw.uni-linz.ac.at/Teaching/Lectures/Sem/2001/Literatur/FileFormatSpec.doc>, 10 October 2000
- [10] Pistelli D. ‘The .NET File Format’, <http://www.codeproject.com/KB/dotnet/dotnetformat.aspx>, 13 Ekim, 2006.