UNİX İŞLETİM SİSTEMLERİ İÇİN AKILLI GERİ DÖNÜŞÜM KUTUSU TASARIMI VE GERÇEKLEŞTİRİLMESİ

Hüseyin Pehlivan[1,†] ve Mehmet Emin TENEKECİ[2]

Bilgisayar Mühendisliği Bölümü, Karadeniz Teknik Üniversitesi
Trabzon, 61080

[1] *e-posta: pehlivan@ktu.edu.tr* [2] *e-posta: emintenekeci@ktu.edu.tr*

ABSTRACT

Recycle bin is a crucial mechanism for wide working environments like operating systems. In current operating systems, such facilities are implemented either in no user-oriented fashion or very poorly. Although various intrusion detection mechanisms are developed to prevent any damage, very few offer the repair of the user's file system as an additional level of protection. This paper presents how to build a recycle bin mechanism for Unix operating systems entirely at the user level. The mechanism involves the control of system resources in a more intelligent way. Using tracing facilities supported by many Unix operating systems, a particular class of system calls are intercepted and untrustworthy programs are prevented from doing irreparable damage. A program called `otokor` *has been constructed and experimented to investigate potential consequences of the recycle bin mechanism.*

Anahtar sözcükler: *Süreç gözetleme, sistem çağrısı analizi, dosya yedekleme*

1. GİRİŞ

Geri dönüşüm mekanizmaları genellikle istenmeyen işlemler sonucu silinen dosyaların kurtarılması amacıyla kullanılır. Mevcut bilimsel çalışmalarda, kaynak erişiminin etkin kontrolündeki zorluklardan dolayı, işletim sistemleri için kullanıcı yönlendirmeli kurtarma mekanizmalarının tedariki ile çok az ilgilenilmiştir. Bir takım restorasyon komutları ile (undo, redo, vb.) işletim sistemlerine bir kurtarma mekanizmasının yerleştirilebileceği gösterilmiştir [1]. Undo ve redo komutları küçük çevreli sistemler için oldukça kullanışlıdır ve kolaylıkla gerçeklenebilir. İşletim sistemleri gibi geniş çevrelerin çoklu işlem yapabilme kabiliyetlerinin olması ve de komutlar arası olası bağımlılıkların ortaya çıkabilmesi restorasyon komutlarının bu çevreler içinde kullanımını zorlaştırır. Windows üzerinde yapılan bir çalışmada güvenilir olmayan programlar izlenerek, yaptığı işlemler kaydedilmiştir [2]. Daha sonra bu kayıtlar incelenerek kötü niyetli programlar tespit edilip silinebilir ve zarar verdiği dosyalar onarılabilir. Bununla birlikte, bu çeşit geri dönüşüm araçları bir anda sadece bir kullanıcının rol aldığı çevreler için geliştirilmişlerdir.

Sistemleri korumaya yönelik yaklaşımlar genel olarak iki kategoriye ayrılabilir; sistem restorasyonu ve istenmeyen sistem erişimlerin tespiti. Sistem restorasyonuyla ilgilenen yaklaşımlar sistemde belirli zaman aralıkları için denetim noktaları oluştururlar ve bu noktalarda bilgi toplama ve depolama işlemlerini yürütürler [3-5]. Kullanıcılar seçtikleri bir aktivite kümesinin etkilerini, sistemi denetim noktalarından birine hareket ettirerek çıkarabilirler. Günümüzde yazılan hemen hemen bütün programlar bu özelliklerle donatılmaya başlanmıştır. Sistemlere erişimi denetlemek için geliştirilen yaklaşımlar ise kullanıcı yada sistem davranışında ortaya çıkan anormallikleri tespit etmeye çalışırlar [6-12]. Normal kullanıcı yada sistem davranışları programların davranışları (yaptıkları sistem çağrılarının sabit yada değişken uzunluklu dizileri) analiz edilerek belirlenir. Normal davranışında bulunmayan sistem çağrılarının bir dizisini yapan programlar anormal kabul edilirler.

Bu bildirinin amacı Unix işletim sistemleri için kullanıcı yönlendirmeli geri dönüşüm araçları sağlamaktır. Windows geri dönüşüm kutusu gibi mekanizmalardan farklı olarak bu onarma araçları, işlemlerini daha akıllı bir yol içinde gerçekleştirir ve dinamik olarak silinen dosyalar için de kullanılabilir. Daha açıkça söylemek gerekirse, geliştirilen araçlar kullanıcıdan kaynaklanan dosya silme işlemleri ile ilgilenir. Sistemin kendisinin sebep olduğu kayıplar

[†] Bu çalışma Karadeniz Teknik Üniversitesi Araştırma Fonu tarafından KTU-2004.112.009.04 nolu proje kapsamında desteklenmiştir.

kapsam dışında bırakılmıştır ve sistem yöneticisi tarafından giderilmelidir.

Sunulan mekanizma `otokor` (OTOmatik KORuma) olarak adlandırılmıştır ve işletim sistemi çekirdeğinde herhangi bir değişiklik yapılmadan tamamen kullanıcı seviyesinde C++ dilinde yazılmıştır. Mekanizma iki alt programda oluşur; kullanıcı aktivitelerini izleyen `gozetle` programı ve silinen dosyaları geri çağıran `geri_al` programıdır. Hizmet programı (daemon) gibi çalışan `gozetle` programı sistem yöneticisi tarafından çalıştırılır. Her bir kullanıcıya ait aktiviteleri denetleyebilen bu program, sistemde bulunan yada sisteme sonradan eklenen bütün komutlarla ve programlarla ilgilenebilir. `Geri_al` programı normal bir sistem programı gibi çalışır ve her kullanıcının kendi ihtiyaçlarına göre yapılandırılabilir.

Diğer bütün kurtarma mekanizmalarında olduğu gibi `otokor` mekanizması da kullanıcı komutlarının ve programlarının dosya sisteminde yaptığı değişiklikleri önceden tespit etmelidir. Sistemde bulunan komut yorumlayıcılar (shell) komutların dosya sisteminde yapacağı değişiklikleri belirlemede kullanılamazlar. Bütün programlar dosya sistemi değişikliklerini sistem çağrıları yardımıyla gerçekleştirdikleri için, bu çağrıların gözetlenmesi etkin bir dosya sistemi denetimi sağlar. Gözetleme gereksinimi birçok Unix işletim sisteminde bulunan /proc dosya sistemi gibi mekanizmalarla karşılanmaktadır. Örneğin, süreçleri gözetlemeye ihtiyaç duyan truss benzeri programlar /proc mekanizmasını kullanırlar. Bu mekanizma ile süreçler, yapılan sistem çağrıları işletim sistemi tarafından icra edilmeden hemen önce durdurularak dosya sisteminde oluşacak değişiklikler belirlenebilir.

2. RESTORASYON KARAKTERİSTİĞİ

Bu çalışma durum ve dosya restorasyonlarını birbirinden ayırır. Sistem durumunda yapılacak bir restorasyon geri yükleme komutlarının (undo ve redo gibi) kullanımı sonucu ortaya çıkar [1]. Dosyaların isimleri ve içerikleri ile birlikte bulundukları dizin ve erişim hakları bir sistemin durumunu belirler. Dosya isminde yapılan bir değişikliğin sistemi başka bir duruma götürdüğü kabul edilir. Bu nedenle, durum restorasyonu bir komutun icrasından etkilenen her sistem bileşenin eski durumuna geri döndürülmesi anlamına gelir. Dosya restorasyonu ise dosya isimleri ve erişim haklarından ziyade sadece dosya içerikleri ile ilgilendiğinden dolayı genellikle durum restorasyonundan faklıdır

Uygulamada bu ayırım kullanıcı ihtiyaçları açısından yararlı işlevler tedarik etmeye yardım eder. Bir kullanıcı çoğunlukla bir işlevden dosyaları doğru içerikleri ile geri getirmesini bekleyecektir. Restorasyonda sadece dosya içerikleri önemli olduğu için, dosyaların korunmasını ve restorasyonunu gerçekleştirebilmek için daha az bilgi saklanması gerekecektir.

Diğer yandan, geri yükleme komutlarının bütün mevcut sistem durumlarına uygulanabilirliğini temin etmede bazı sorunlar vardır. Bu konuda önemli bir sorun programların eşzamanlı çalışmalarını kontrol edebilme gereksinimidir. Eşzamanlılık gerektiği gibi kontrol edilse bile, zamanın paylaşımlı kullanılması sonucu ortaya çıkan rasgele dosya işlemleri sırasından dolayı bazı programların etkilerinin geri alınabilmesi mümkün olmayabilir. Örneğin, ayrı komut satırlarından girilen `P1` ve `P2` gibi iki programın eşzamanlı icrasıyla aşağıda verilen dosya işlemlerinin yapıldığını varsayalım.

```
10> Sil dosyaA 5  P1
11> Yap dosyaB 11 P1
12> Oku dosyaA 11 P2
13> Yap dosyaB 13 P2
14> Oku dosyaB 13 P1
```

Bu işlemler iki programın etkilerinin birbirinden bağımsız olarak düzeltilebilmesine imkan vermezler ve `dosyaA`'nın eski içeriğine (`5` numaralı versiyonu) ulaşmayı imkansız yaparlar. Restorasyon işlemleri sırasında kullanıcının bir anda sadece bir komutu seçmesine izin verildiği için böyle durumlardan kurtulmanın kullanışlı bir yolu mevcut değildir. Bu nedenle her bir dosya işlemi ile ayrı ayrı ilgilenilmesi gerekir.

Dosya koruma ile güvenli komut çalıştırma arasında yakın bir ilişki vardır. Bir dosya sistemini korumanın en iyi yolu çalıştırılan her sistem komutunun güvenilir olmasını temin etmektir. Literatürde bulunan birçok çalışma programların güvenli çalışmaları için bazı kısıtlamalar getirmiştir ve restorasyon gereksinimini karşılamamışlardır. Komutların güvenli çalışmasının tedariki için çalışma çevresinin kısıtlanmasına gerek yoktur. `otokor` mekanizması kullanıcıların çalışma ortamına her hangi bir kısıtlama getirmemeyi amaçlamıştır.

3. TASARIM VE GERÇEKLEŞTİRME

`Otokor` programı iki parçadan oluşmaktadır; `gozetle` ve `geri_al`. `Gozetle` hem kullanıcı aktivitelerini izler hem de kullanıcının ana dizininde oluşturulan *`.otokor`* isimli dizinde restorasyon için gerekli bilgileri saklar. Saklanan bilgilerinin miktarı tümüyle yolu kesilen sistem çağrılarının dosya sistemde meydana getireceği etkilere bağlıdır. Her bir sistem çağrısı bağımsız olarak ele alınmıştır. `Geri_al` programı ise dosyaların eski içeriklerini geri getirerek restorasyon gereksinmelerini karşılar.

Bu program parçalarının sahipliği sistem yöneticisine verilmiştir ve bunun sonucu olarak kullanıcıların *`.otokor`* dizinine doğrudan erişimleri engellenmiştir. Bu kısıtlama restorasyon bilgilerini kullanıcının

çalıştırdığı diğer programlardan korumak için gereklidir. Ayrıca, daha etkin bir koruma için, restore isimli bir sistem grubu oluşturulmuş ve `otokor` mekanizmasını çalıştıracak kullanıcıların isimleri bu gruba eklenmiştir. `Geri_al` programının grup sahipliği de restore olarak yapılandırılmıştır.

Kullanıcıların bütün aktivitelerinin kontrol edilebilmesi için sadece komut yorumlayıcıların (login shell gibi) gözetlenmesi yeterli değildir. Unix işletim sistemleri çeşitli uzaktan erişim seviyelerine sahiptir. Her bir seviye için kullanıcıların hesaplarını uzaktan yönetebilmelerini sağlayan birçok araç vardır. Bu araçların ve onlara hizmet veren sistem programlarının bir listesi Tablo 1'de verilmiştir.

Table 1. Unix sistemlerindeki uzaktan bağlantı araçları ve hizmet programları

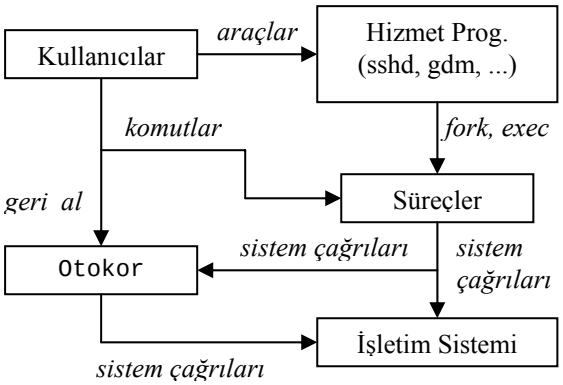
Araçlar	Örnek	Hizmet Programları
Telnet/SSH	PuTTy	telnetd, sshd
FTP/SFTP	WinSCP3	ftpd, sftpd
X sunucuları	X-Win32	kdm, gdm
Web sunucuları	Apache	httpd, webservd

Tablo 1'de verilen hizmet programlarının isimleri sistemden sisteme değişiklik gösterir. Örneğin, KDE ve GNOME gibi masaüstü çevrelerini kullanabilmeyi sağlayan X–Window sistemi sunucuları, kdm ve gdm gibi hizmet programları ile sisteme bağlantı kurarlar. Bütün bu araçlar içerisinden yapılan her dosya sistemi aktivitesi kontrol altında tutulmaya ihtiyaç duyar.

Unix çevreleri metin düzenleme ve program geliştirme amaçları için birçok yardımcı program içerir. Bu programlar "üretilen dosyalar" olarak tanımladığımız çeşitli geçici yada kalıcı dosyalar üretirler ve onları sistemin yada kullanıcının ana dizininde oluşturulan alt dizinler (`/var` ve `~/.ssh` gibi) içinde tutarlar. `Otokor` programı kullanıcının ana dizininde ve onun alt dizinlerinde bulunan bütün dosyaların korunmasıyla ilgilenir. Bazı dillere ve programlara özel ("`.aux`", "`.log`" ve "`.o`" gibi uzantılı) dosyalar kapsam dışında bırakılmıştır. Her kullanıcı kendi ihtiyaçlarına göre `geri_al` programının sağladığı bir menü yardımıyla mevcut yapılandırmayı değiştirebilir.

`Otokor` programı restorasyon bilgisini *`.otokor`* dizini altında bulunan *`.otokor`* (silinen dosyaların kaydını içerir) ve *`.conf`* (korunması istenmeyen dosya ve dizinlerin isimlerini içerir) olarak adlandırılan iki gizli dosya içinde saklar. *`.otokor`* dizininin diğer üyeleri ise ilgili dosya ismine artan sırada bir numara uzantısı verilerek (`dosyaA_5` gibi) içerikleri bütün olarak saklanan dosya yedekleridir.

`Otokor` programının nasıl çalıştığı Şekil 1'de gösterilmeye çalışılmıştır. Sistemin kullanıcılara hizmet veren bütün süreçleri gözlem altında tutulmaktadır. Hizmet programları genellikle yeni süreçler oluşturarak kullanıcılara hizmet verirler. `Otokor`, bir *open* sistem çağrısıyla karşılaştığı zaman çağrının ilk argümanının kullanıcının ana dizini altında bulunan bir sistem yolunu işaret edip etmediğini kontrol eder. İşaret edilen dosya diğer kullanıcılara yada paylaşılan bir kütüphaneye aitse herhangi bir restorasyon bilgisi saklanmaz ve sistem çağrısı çalışmaya devam etmesi için serbest bırakılır. Aksi durumda *open* çağrısının ilgili dosyayı etkileyip etkilemediğini anlamak için ikinci argümanı (bayrak da denilen) incelenir. Çağrının etkisi bir dosya silme işlemine karşılık geliyor ise ilgili dosyanın tam bir kopyası alınır (*open* çağrısı için bir dosya silme işlemi `O_WRONLY`, `O_RDWR` and `O_TRUNC` gibi bayrakların kullanımı sonucu ortaya çıkar). Sonra *open* çağrısının çalışması devam ettirilir. Dosya içeriklerinde yapılan değişiklikler de silme işlemleri gibi düşünülür. Dosya sistemi üzerinde çalışan her sistem çağrısı için benzer kontrol ve bilgi saklama işlemleri yapılır.



Şekil 1. `Otokor` mekanizmasının mimarisi

Gözetleme esnasında, `gozetle` programı bilgi toplamayı ve saklamayı her bir kullanıcının kendi yapılandırmasına göre yapar. Bir sistem sürecinin bir kullanıcıdan daha fazlasına hizmet verebilmesinden dolayı kullanıcı aktivitelerini farklı süreçlerle bireysel olarak gözetlemek uygun değildir. Bu nedenle bütün kullanıcıların aktiviteleri `gozetle` programının tek bir örneği ile gözetlenir. Diğer yandan her kullanıcı için `geri_al` programının başka bir örneğini çalıştırılır. Her `geri_al` örneği kendisini çağıran kullanıcının ana dizininde bulunan *`.otokor`* içindeki restorasyon bilgilerini kullanır. Bir restorasyon işlemi silinen bir dosyanın *`.otokor`* dizininden sistemde eski bulunduğu dizine taşınmasından ibarettir.

4. EŞZAMANLI KONTROL

Dosya restorasyonunda ilgilenilmesi gereken önemli bir konu eşzamanlılıktır. Sistem çağrılarının kullanımından dolayı birbirlerinden bağımsız olarak çalışan ve sistem kaynaklarını rasgele paylaşan süreçler sistemde belirsizliğe neden olabilirler. Sistem kaynaklarının kontrolsüz paylaşımından kaynaklanan belirsizliği azaltmak için `otokor` programı işletim sisteminden yada mevcut komut yorumlayıcılardan

daha akıllı bir yol içinde eşzamanlılıkla ilgilenmek zorundadır. `/proc` arayüzü ile bütün kullanıcı süreçlerinin ve onların oluşturduğu yeni süreçlerin sadece bir gözetleme süreci ile kontrol edilmesi mümkündür. `Gozetle` programı süreç eylemlerini dinlemeyi sağlayan *poll* sistem çağrısını kullanarak bir gözetleme süreci gibi davranır. Çalışmakta olan süreçlerin kontrolünde C++ dilinde tanımlanan aşağıdaki süreç tablosundan (*procT*) yararlanılır.

```
int nprocs;
int current_time;
struct pollfd Pollfds[MAXPROCS];
struct processTable {
   int time;
   pid_t pid;
   int procfd;
   prstatus_t *pstatus;
} procT[MAXPROCS];
```

`Pollfd` yapısı, süreç eylemlerinin etkin kontrolünü eşzamanlı olarak gerçekleştirmek için poll çağrısı tarafından kullanılır. *pid* ile temsil edilen her yeni süreç farklı bir *time* değeri ile tabloya eklenir. Tabloda bütün süreçler aynı seviyeli olarak tutulurlar. Eş zamanlı dosya erişimlerinin tespit edilmesi ve kaydının tutulması için ikinci bir tablo (*fileT*) tanımlanmıştır.

```
static int nfiles;
struct fileTable {
   int nprocfds;
   int procfd[MAXPROCS];
   char *path;
} fileT[MAXFILES];
```

Açılan her dosya için tabloya yeni bir kayıt eklenir. Aynı dosyayı açan süreçler aynı kayıt içinde birlikte tutulurlar. Bir dosyanın açık kaldığı zaman aralığı *open* ve *creat* benzeri çağrılar ile close çağrısı tarafından belirlenir. Dosya tanımlayıcıların (*dup* ve *fcntl* gibi çağrılarla) kopyalanabilmesinden dolayı bir *close* çağrısı ile karşılaşma her zaman ilgili dosyanın kapatıldığı anlamına gelmez. Süreçlerin miras yapıları da dosya tanımlayıcılarını açık bırakabilir. Bir sürecin oluşturduğu dosya tanımlayıcısının `FD_CLOEXEC` bayrağı temizlendiği zaman, ilgili dosya *fork* ve *exec* çağrılarından sonra açık kalır. Aslında dosya tanımlayıcılarının izlenmesi süreçlerin çalışma çevrelerinde birçok dinamik kontrole ihtiyaç duyacağından kolay ve pratik değildir. Sistemin performansını etkilememek için, bir dosya tanımlayıcısını oluşturan yada miras alan bir süreç sonlanana kadar ilgili dosyanın açık kaldığı varsayılmıştır.

Dosya yedeklemesinin gerekip gerekmediği süreçlerin dosya açma işlemleriyle belirlenir. Her dosya açma işlemi bir kopya almayı gerektirmez. Dosyayı yazma amacıyla açan ilk süreç için bir kopya alınır. Dosya açık kaldığı sürece, aynı dosyayı açan diğer süreçler için yeni kopyalar alınmaz. Bir dosyayı açan bütün süreçler sonlandıktan sonra, yeni bir sürecin aynı dosyayı yazma amacıyla açmaya teşebbüs etmesi kopya alınmasını gerektirir.

5. PERFORMANS ÖLÇÜMÜ

Sisteme getirilen yükü belirlemek için, gözetleme altında sistem programların çalışma süreleri ve `otokor` programının disk alanı kullanımı ölçülmüştür. Çalışma süresi sistem çağrılarının yakalanması, dosya içeriği değişikliklerinin tespit edilmesi ve gerekli bilgilerin saklanması işlemlerinden etkilenir. Disk kullanımı ise `.otokor` dizininin boyutu ile belirlenir. Root yetkileri ile sistemin üç hizmet programı (*sshd*, *sftp-serve*r, *dtlogin*) gözetlenmiştir. Dosya sistemi ile etkileşimde bulunacak yeterli miktarda web içeriği olmadığından *webservd* programı gözetlenememiştir. Tablo 2'de iki işlemcili, 1.28 GHz, 4 GB RAM' lı, 74 GB 4 Ultra160 SCSI hard diski olan Sun Solaris Sparc makinesinde değişik programların çalışma süreleri verilmiştir. Bu makineye, Tablo 1'de verilen çeşitli araçlarla günde yaklaşık 12 uzaktan bağlantı yapılmaktadır. Ağ ortamının etkilerini hesaplamalara yansıtmamak için bağlantı süreleri ölçülmemiştir. Sadece işlemci zamanları verilmiştir.

Table 2. `Otokor` programının sisteme getirdiği yük (programların icra süreleri)

Program	Gözetlemesiz	Gözetlemeli	Yük
cp	35ms	37ms	%5
latex	160ms	182ms	%13
xterm	390ms	439ms	%12
netscape	2093ms	2142ms	%2

Sistem çağrılarının işletim sistemi çekirdeği içinde yakalanmasına dayandırılan teknikler ile karşılaştırıldığında, kullanıcı seviyeli mekanizmalar daha fazla sistem yüküne neden olurlar. Bu durum, her bir sistem çağrısı için gözetlenen süreç ile gözetleyen süreç arasında içerik değişimleri ortaya çıkmasından dolayı kaçınılmazdır. İçerik değişimlerinin sayısı bir programın yaptığı sistem çağrılarının sayısına bağlıdır. Dosya sistemi işlemleri düşünüldüğünde her bir kullanıcı programı çok az sayıda sistem çağrısı icra eder. Örneğin, "`rm *`" komutu çalışma dizininde bulunan her bir dosya için bir *unlink* çağrısı yapar. Çoğu Unix komutlarının dosya sistemi aktivitelerinin az olması sistem üzerinde oluşan yükü de azaltacaktır.

Disk alanı kullanımı da yukarıda verilen özelliklere sahip makine üzerinde 1 haftalık zaman dilimi için ölçülmüştür. Bir kullanıcı için disk alanının boyutu öncelikle işgal edilen alanın %18'i kadar artmıştır. Diğer bir kullanıcı için artış sadece %12'dir. Bu yükler dosya yedekleme işlemleri sonucu oluşur. İşgal edilen alanı küçük tutmaya bir çözüm olarak saklanan dosyalar sıkıştırılabilir, fakat bu işlemin saklama zamanında ciddi bir artışa neden olmamasına dikkat edilmelidir.

5. SONUÇLAR

Bu bildiride Unix sistemleri üzerinde dosya sistemine verilen zararları kullanıcı yönlendirmeli bir yol içinde onarmak için bir geri dönüşüm mekanizması sunulmuştur. Mekanizma dosya sisteminin bütünlüğünü ve güvenliğini tehdit eden bütün durumlarla ilgilenir ve kullanıcıya istemeyerek silinen dosyaları geri alma olanağı sağlar. Bu işlevler bireysel kullanıcı aktiviteleri izlenerek, zarar verilen dosyalar yedeklenerek ve aktivitelerin etkilerini ortadan kaldırmak için restorasyon bilgisi kullanılarak başarılmıştır. Programların olası dosya işlemlerine karşı mekanizmanın sistem üzerindeki etkileri ölçülmüş ve programların çalışma süreleri ile disk alanı kullanımının kabul edilebilir olduğuna karar verilmiştir.

Gelecek çalışmada geri dönüşüm kutusunun optimizasyonu üzerinde durulacaktır. Bu hedefi başarmak için bir konu dosya yedeklerinin kullanıcının disk alanında ne kadar süre ile saklanacağını karar vermektir. Bazı kullanıcılar dosya korumanın birkaç oturumu kapsamasını bekleyebilirler. Uzun süreli dosya korumalarından kaçınmak için mekanizmanın etkin kullanımını sağlayacak en uygun süre belirlenmelidir. Bu diğer bir konuyu ortaya çıkarır; yedeklenen dosyaların disk alanı kullanımı. Her bir kullanıcı için müsaade edilen disk kotaları dikkate alarak daha az disk alanı işgal edilmesi sağlanmalıdır.

KAYNAKLAR

[1] Holyer, I., and Pehlivan, H., "A recovery mechanism for shells", *The Computer Journal, Vol. 4, 2000, No. 3, 1-9*.

[2] Chen, H., Hsu, F., Li, J., Ristenpart, T., "Back to the Future: A Framework for Automatic Malware Removal and System Repair", *Technical Report UCD//CSE-2005-6, UC Davis*, 2005.

[3] Brown, A.B., Patterson, D.A., "Undo for operators: Building an undoable e-mail store", In *Proceedings of the 2003 Annual USENIX Technical Conference*, 2003.

[4] Zhou, C., and Imamiya, A., "Object-based nonlinear undo model", In *Proceedings of the Twenty-First Annual International Computer Software and Applications Conference, COMPSAC97,* 1997, 50-55.

[5] Berlage, T., Genau, A., "From undo to multi-user applications", In *Proceedings ofthe Vienna Conference on Human-Computer Interaction* (Vienna, Austria, Sept 20–22), 1993, 213-224.

[6] King, S., and Chen, P., "Backtracking intrusions", In *Proceedings of the 2003 Symposium on Operating Systems Principles* (SOSP), 2003.

[7] Qiao, Y., Xin, X.W., Bin, Y., Ge, S., "Anomaly intrusion detection method based on HMM", *Electronics Letters, Vol. 38, 2002, No. 13, 663-664*.

[8] Warrender, C., Forrest, S., Pearlmutter, B., "Detecting intrusions using system calls: alternative data models", *Proceedings of the 1999 IEEE Symposium on Security and Privacy* ,1999, 133-145.

[9] Forrest, S., Hofmeyr, S.A., Somayaji, A., "Intrusion Detection using sequences of System Calls", *Journal of Computer Security Vol. 6, 1998, 151-181*.

[10] Sun,W., Liang, Z., Venkatakrishnan, V.N., Sekar, R., "One-way isolation: An effective approach for realizing safe execution environments", In *Proceedings of Network and Distributed Systems Symposium* (NDSS), 2005.

[11] Christodorescu, M., and Jha, S., "Testing malware detectors", In *Proceedings of the International Symposium on Software Testing and Analysis* (ISSTA), .2004.

[12] Sekar, R., Bendre, M., Bollineni, P., Dhurjati, D., "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors", *IEEE Symposium on Security and Privacy*, 2001.