Bölüm 19

DİJİTAL DELİLLERİN KURTARILMASI LINUX SİSTEMLERİNDEN

Philip Craiger

Özet Linux çekirdeği tabanlı işletim sistemleri yaygınlaştıkça, kolluk kuvvetlerinin cezai soruşturmalarda işlemesi gereken Linux sistemlerindeki kaçınılmaz artış. Microsoft-Windows tabanlı sistemlerden kanıt kurtarmak için gereken beceriler ve uzmanlık, Linux sistemlerine mutlaka çevrilmez. Bu makale, Linux sistemlerinden kanıt kurtarmak için dijital adli prosedürleri ele almaktadır. Özellikle, diskten ve geçici bellekten silinen dosyaları tanımlama ve kurtarma, önemli ve Truva atı dosyalarını tanımlama, gizli dosyaları bulma ve yeniden adlandırılmış uzantılara sahip dosyaları bulma yöntemlerini sunmaktadır. Tüm prosedürler Linux komut satırı yardımcı programları kullanılarak gerçekleştirilir ve özel veya ticari araçlar gerektirmez.

Anahtar kelimeler: Dijital delil, Linux sistem adli bilimi

giriiş

Linux'un özellikle sunucular için tercih edilen işletim sistemi olarak popülerliği arttıkça, suç mahallerinde Linux sistemlere daha fazla rastlanacak.
Ancak Microsoft-Windows tabanlı bir sistemden kanıt kurtarmak için gereken beceriler ve uzmanlık, Linux sistemindeki aynı görevlere mutlaka çevrilmez.
Örneğin, Microsoft NTFS, FAT ve Linux EXT2/3 dosya sistemleri yeterince farklı çalışır ve birini anlamak diğerinin nasıl çalıştığı hakkında pek bir şey söylemez. Bu makalede, Linux komut satırı yardımcı programlarını kullanarak Linux sistemleri için dijital adli tıp prosedürlerini gösteriyoruz. Çalışan bir sistemden kanıt toplama yeteneği özellikle önemlidir çünkü adli tıp ilk müdahale görevlisi canlı kanıt toplamayı önceliklendirmezse RAM'deki kanıtlar kaybolabilir.

Tartışılan adli prosedürler arasında, RAM ve manyetik ortamlardan silinen dosyaları tanımlama ve kurtarma, silinen dosyaları tanımlama yöntemleri yer almaktadır.

tablo dosyaları ve truva atları, gizli dosyaları ve yeniden adlandırılmış dosyaları (uzantıları değiştirilen dosyalar) bulma.

Canlı bir RAM'den silinen dosyaların kurtarılmasını açıklayarak başlıyoruz (Çalışıyor) Linux sistemi. Linux sistemleri genellikle sunucu olarak kullanıldığından, gösterilerin çoğu saldırganların bir sisteme girdikten sonra kullandıkları bilinen etkinliklere ve tekniklere yöneliktir.

2. RAM'den Dosyaları Kurtarma

İçeriği diske yazılmış silinmiş bir dosya yine de kurtarılabilir. Adli tekniği örneklendirmek gerekirse, bir saldırganın bir programı çalıştırıp sonra varlığını gizlemek için onu diskten sildiğini varsayalım.
Bu, örneğin bir saldırganın net cat yardımcı programını çalıştırarak kurban sistemine bir arka kapı kurması ve ardından yardımcı programı diskten silmesiyle gerçekleşir. Program bellekte çalışan bir işlem olarak kaldığı sürece, orijinal yürütülebilir dosya kurtarılabilir. Dosya kurtarılabilir çünkü Linux çekirdeği, çalışan işlemler, bağlanmış dosya sistemleri, çekirdek bilgileri ve birkaç yüz diğer kritik sistem bilgisi parçası dahil olmak üzere sistemin genel durumunu izlemek için bir sözde dosya sistemi kullanır [6]. Bu bilgiler sanal bellekte tutulur ve /proc dizini aracılığıyla erişilebilir. Aşağıdaki (kısmi) liste, çalışan bir Linux sistemindeki /proc dizininin içeriğini gösterir.

Aşağıdaki sayıların her biri bir işlemci kimliğine karşılık gelir ve çalışan işlemle ilgili bilgileri içeren bir dizindir.

```
# /proc 1 4
 4513 4703 4777 execdomains mdstat takasları 1693 40 4592 4705
 acpi fb meminfo sys
2 4045 4593 4706 asound dosya sistemleri çeşitli 2375 41 4594
4708 buddyinfo fs mm sysvipc 2429 4163 4595 4709 bus ide
modülleri tty 2497 4166 4596 4712 cmdline kesintiler bağlar
çalışma süresi 2764 4186 4597 4713 config.gz iomem mtrr sürüm 29 42 4620 4715
cpufreq ioports net vmstat
```

Silinen bir dosyanın kurtarılmasını göstermek için, bir saldırganın bir parola kırıcı indirdiğini ve sistem parolalarını kırmaya çalıştığını varsayalım - saldırganlar için çok yaygın bir hedef. Saldırgan, pass adlı bir dosyada parolaların bir listesiyle j ohn (www.openwall.com) parola kırıcıyı çalıştırır. Saldırgan daha sonra hem yürütülebilir dosyayı hem de parolaları içeren metin dosyasını siler, yürütülebilir dosya işlem sonlandırılana kadar bellekte çalışmaya devam eder. ps komutu çalışan işlemleri görüntüler. Aşağıdaki liste, "John" yürütülebilir dosyasının "pass" dosyasıyla sabah 10:10'da çağrıldığını, 22 saniyedir çalıştığını ve root'a ait olduğunu gösterir.

```
# ps aux I grep John root 5288 97.9 0.0
1716 616 pts/2 R+ 10:10 0:22 ./John pass
```

Yukarıdaki listeye göre yürütülebilir işlem kimliği (PID) şudur:
5288. /proc/5288 dizini, aşağıdaki (kısmi) listede gösterildiği gibi, çalışan işlemle ilgili bilgileri içerecektir.

```
# -al /proc/5288
toplam 0
dr-xr-xr-x 3 kök kök 0 17 Oca 10:11 .
dr-xr-xr-x 108 kök kök 0 Oca 17 04:00 ..
-r—r—r— 1 kök kök 0 17 Ocak 10:11 cmdline
Irwxrwxrwx 1 kök kök 0 Jcin 17 10:12 cwd -> /j
-R 1 kök kök 0 Jein 17 10:12 çevre
Irwxrwxrwx 1 root root 0 17 Oca 10:12 exe -> /j/john (silindi)
Irwxrwxrwx 1 kök kök 0 17 Oca 10:12 kök -> /
-r—r—r— 1 kök kök 0 17 Oca 10:11 istatistik
-r—r—r— 1 kök kök 0 17 Oca 10:12 statm
dr-xr-xr-x 3 kök kök 0 17 Ocak 10:12 görev
```

/proc/5288 dizini birçok dosya ve dizin içerir, bunların en önemlisi çalışan parola kırıcıya bir sembolik hnk olan exe'dir (izinlerin ilk sütunundaki 1'e dikkat edin).

İşletim sistemi (yardımcı bir şekilde) dosyanın diskten silindiğini belirten bir not görüntüler. Yine de, exe'yi dizinden ayrı bir dizine kopyalayarak dosyayı kurtarabiliriz.

```
# cp /proc/5288/exe ./John.kurtarıldı # md5sum ./John.kurtarıldı ./
John.original 83219704ded6cd9a534baf7320aebb7b ./John.kurtarıldı
83219704ded6cd9a534baf7320aebb7b ./John.original
```

Yukarıdaki örnekte exe'yi /proc/5288'den başka bir dizine kopyaladık ve sonra yürütülebilir dosyanın MD5 karma değerini John'un bilinen bir kopyasının karma değeriyle karşılaştırdık. Karma değerlerinin aynı olduğunu görüyoruz, bu da dosyayı başarıyla kurtardığımızı gösteriyor. Bu dosya kurtarma yöntemi, işlem bellekte kaldığı sürece her tür dosya için işe yarar.

3. Dosyaları Türe Göre Kurtarma

Bir dosyayı, dosyanın başında bulunan dosya başlığı için tahsis edilmemiş alanı arayarak elle kurtarabiliriz. Örneğin, bir saldırganın birkaç yüz bitmap grafik içeren bir dizini sildiğini bildiğimizi varsayalım. Tahsis edilmemiş alanda, bir bitmap grafiğinin imzası olan BM ile başlayan bir sektör arayabiliriz. Bulduğumuzda, dosyayı Linux dd komutunu kullanarak elle kurtarabiliriz. Bu prosedürün başarısı şunları varsayar: (i) başlık bilgilerini belirleyebiliriz.

(ii) dosyanın üzerine yazılmamış olması ve (iii) dosyanın parçalanmamış olması.
Dosya parçalanmışsa, dosyayı daha önce oluşturan blokları tanımlayamayacağımız için dosyanın yalnızca bir kısmını kurtarabiliriz. Bu gösterimde, birkaç silinmiş *.jpg dosyası içeren bir görüntümüz (veya bağlanmamış bölümümüz) var. Öncelikle, bir JPG dosyasında yaygın olarak bulunan JFIF metnini arayarak her JPG dosyasının ilk sektörünü tanımlamalıyız. Aşağıdaki liste, silinmiş bir JPG dosyası için bir başlangıç sektörü göstermektedir. (Not: Bir *.jpg dosyası için gereken tek başlık kısmı ilk üç bayttır: ff d8f f eO. Deneylerde, başlıktaki JFIF'i silmenin uygulamaların dosya türünü doğru bir şekilde tanımlamasını engellemediğini, ancak ilk üç bayttan herhangi birini kaldırmanın engellediğini bulduk.)

```
0004200: ffd8 ffeO 0010 4a46 4946 0001 0200 0064
```

Yukarıdaki hsting, dosyanın 0x4200'de (hex) başladığını gösteriyor. Bunu ondalık sayıya, 16.896'ya dönüştürüyoruz ve 512'ye (sektördeki bayt sayısı) bölüyoruz, sonuç olarak 33 çıkıyor, bu da dosyanın başlangıç sektör numarası, yani görüntünün başlangıcından itibaren. Birçok durumda silinen bir dosyanın tam boyutunu bilemeyeceğiz, bu da boyutu hakkında eğitimli bir tahminde bulunmamızı gerektirecek. Çok düşük tahminde bulunursak ve dosyayı eksik kurtarırsak, dosyayı uygulamasında görüntülemek çok az sektörün kurtarıldığını gösterecek ve dosya tamamlanmış görünmeyecektir. Çok fazla sektör kurtarırsak, aşırı kurtarma yapmış oluruz. Deneyimlerimize göre çok fazla sektörü kurtarmak dosyaya zarar vermez. Dosyayı kurtardıktan sonra, tahminimizin doğruluğunu belirlemek için uygun uygulamada görüntüleyebiliriz.

Dosyayı görüntüden ayırmak için UNIX/Linux dd komutunu kullanıyoruz.

Giriş dosyasını görüntümüz olarak belirtiyoruz (if=image.dd) ve kurtarılan dosya için bir ad seçiyoruz (of=recoveredl.jpg). Oymaya başlamak için başlangıç sektörünü belirtmeliyiz. Önceki hesaplamamıza göre görüntü fiziksel sektör 33'te başlıyor. dd'deki varsayılan blok boyutu 512 bayttır ve bunu olduğu gibi bırakacağız. Son olarak kurtarılacak ardışık blok sayısını belirtmeliyiz. Bu örnekte her biri 512 bayt boyutunda 30 blok tahmin edeceğiz, bu nedenle 15K boyutunda dosyaları kurtarıyoruz.

```
# dd if=image.dd of=recoveredl.jpg atla=33 sayım=30
30+0 kayıt
30+0 kayıt çıktı
# dosya recoverl.jpg
recoverl.jpg: JPEG görüntü verisi, JFIF standardı 1.01
```

JPG dosyasının 30 ardışık sektörünü başarıyla kurtardık.
dosya komutu başlığı başarıyla kurtardığımızı gösteriyor.

Bu kurtarma yöntemi, dosya başlık bilgisi bozulmadan kaldığı sürece her türlü dosya türüyle kullanılabilir. Bu yöntemin başarısı

yine dosya parçalanmasının olmamasına ve dosyanın herhangi bir bloğunun yeniden kullanılıp kullanılmadığına dair şansa bağlıdır.

3.1 Genel Dosya Kurtarma Prosedürü

Bir dosya başlığı üzerine yazılmışsa ve dosya esas olarak metinden oluşuyorsa, yalnızca arama yapmak için bazı anahtar sözcükleri bilmemizi ve elbette dosyanın tamamen üzerine yazılmamış olmasını gerektiren daha genel bir kurtarma prosedürü kullanabiliriz.

Bu gösteri için Linux genel günlük dosyası /var/log/messages'ı kurtaracağız. Bu dosya, saldırganın izlerine dair kanıt içereceğinden, genellikle bir saldırgan tarafından hedef alınır. Acemi saldırganlar, bir yönetici için bir saldırının gerçekleştiğine dair açık bir kanıt olan tüm dosyayı siler. Buna karşılık, yetenekli saldırganlar, izinsiz girişlerine işaret eden para cezalarını cerrahi olarak kaldırarak kalan içerikleri korur. Günlük dosyasını kurtarmak için dosyada bulunan anahtar sözcükleri belirlemeliyiz. İdeal olarak, dosyaya özgü anahtar sözcükleri belirleyerek yanlış pozitif sonuçların sayısını azaltmalıyız. Bu örnekte, günlük dosyaları sık sık döndürüldüğü için bazı yanlış pozitif sonuçlarla karşılaşmamız olasıdır, bu nedenle aramamızın günlük dosyasının önceki sürümlerinden anahtar sözcükler alması muhtemeldir. Mesajların bulunduğu /var dizinini içeren bölümü kaldırıyoruz. /var kendi bölümündeyse bu basittir:

```
# /dev/hda3'ü umount et
```

/var kök dizinle aynı bölümdeyse, Linux önyüklenebilir CD kullanarak sistemi yeniden başlatmamız ve prosedürleri önyükleme diskinden gerçekleştirmemiz gerekir [2]. Sonraki kullanımda fiziksel aygıtta (bağlanmamış bölüm) anahtar sözcükleri aramak için grep kullanırız. Fiziksel aygıtı kullanıyoruz çünkü fiziksel aygıt aracılığıyla tahsis edilmemiş alana erişmemiz gerekiyor:

```
# grep -ia -f anahtar sözcükler -C 2 /dev/hda3
```

fiag i, büyük/küçük harfe duyarlı olmayan bir arama belirtir. fiag a, şunu belirtir: girdiyi (fiziksel aygıt /dev/hda3 içerikleri) ASCII metni olarak işlemek için; bunu yapmazsak grep yalnızca dosyanın anahtar sözcüğü içerip içermediğini gösterecektir. fiag f, ardından gelenin aranacak anahtar sözcüklerin bir listesini içeren bir metin dosyası olduğunu belirtir. Esasen, örneğin silinen dosyamızın tam olarak hangi anahtar sözcükleri içerdiğinden emin olmadığımızda kullanabileceğimiz birden fazla anahtar sözcük için eş zamanlı bir arama yürütüyoruz. fiag -C 2, bir anahtar sözcük isabetinden iki satır önce ve sonra iki satır bağlam istediğimizi belirtir. Son olarak, /var dizinini içeren aranacak fiziksel aygıtı belirtiriz. Bunun için

Gösterim saldırganın kök hesabına giriş yapmak için birkaç başarısız girişimde bulunduğunu varsayıyoruz - bir saldırıda yaygın bir durum. Bu başarısız giriş girişimleri mesaj günlük dosyasında belirtilecektir. Aramamızın sonuçları aşağıda gösterilmektedir:

18 Ara 19:13:09 gheera gdm(pain_unix) [2727]: kimlik doğrulama hatası; logname= uid=0 euid=0 tty=:0 ruser= rhost= user=schmoopie
18 Aralık 19:13:13 gheera gdm-binary[2727]: Kullanıcı doğrulanamadı
18 Aralık 19:13:16 gheera gdm(pam_unix) [2727]: oturum, (uid=0) tarafından schmoopie kullanıcısı için açıldı

20 Aralık 18:33:29 gheera gdm(pain_unix) [2752]: kimlik doğrulama hatası; lognaine= uid=0 euid=0 tty=:0 ruser= rhost= user=schmoopie
21 Ara 18:16:55 gheera gdin(pain_unix) [2750]: kimlik doğrulama hatası; logname= uid=0 euid=0 tty=:0 ruser= rhost= user=schmoopie

22 Ara 17:49:33 gheera gdin(pam_unix) [2756]: kimlik doğrulama hatası; logncane= uid=0 euid=0 tty=:0 ruser= rhost= user=schmoopie
22 Ara 17:49:36 gheera gdm-binary[2756]: Kullanıcı doğrulanamadı
22 Ara 17:49:48 gheera gdm(pain_unix) [2756]: oturum, (uid=0) tarafından schmoopie kullanıcısı için açıldı

Anahtar sözcükler kalın yazılmıştır. schmoopie kullanıcısının 18, 19, 21 ve 22 Aralık'ta kök olarak başarısız bir şekilde oturum açmaya çalıştığı anlaşılıyor. Her arama sonucu öncesinde ve sonrasında iki satırlık bağlamı not edin. Pratikte bu kadar sınırlı bir sonuç istemezdik: Günlük dosyasının tüm içeriğini kurtarmayı tercih ederdik, bunu da bağlam için çok daha büyük bir değer isteyerek yapabiliriz, örneğin -C 100. Dosyanın ne kadar büyük olduğunu önceden bilmediğimiz için bu deneme yanılma çabası olacaktır.

3-2 EXT2 Disklerinden Dosyaları Kurtarma

Son kurtarma yöntemi, dosya sisteminin EXT2 olduğunu varsayar, bu Linux dosya sistemidir (daha eflScient günlükleme dosya sistemleriyle değiştiriliyor olsa da). Bu yöntemde dosyayı bulmak ve kurtarmak için sistem hata ayıklayıcısını kullanabiliriz. Bu örnek için, yakın zamanda işten çıkarılan bir çalışanın /home dizininden önemli bir dosyayı sildiğini varsayalım. (İşten çıkarılan çalışanlar için alışılmadık bir olay değildir.) Diyelim ki dosyanın bir zip arşivi olduğu bilgisi bize iletildi. Dosya kurtarma işlemine başlamadan önce sabit diskin geometrisini, bölüm sayısını, bölümlerin nasıl biçimlendirildiğini belirlemeliyiz. Linux komutu fdisk -1 şu bilgileri sağlar:

fdisk -1 Disk /
dev/hda: 30.0 GB, 30005821440 bayt 16 kafa, 63 sektör/parça, 58140 silindir

Birimler = 1008 * 512 = 516096 baytlık silindirler
Aygıt Önyükleme Başlangıç Bitiş Blokları Kimlik Sistemi /dev/hdal 1
41613 20972826 83 Linux

/dev/hda2 57147 58140 500976 f W95 Genişletilmiş (LBA) /dev/hda3
41613 52020 5245222+ 83 Linux
/dev/hda5 57147 58140 500944+ 82 Linux takası

Dört bölümlü tek bir 30 GB IDE sabit diskimiz olduğunu görüyoruz. İlk bölüm (/dev/hdal) EXT2'de biçimlendirilmiş birincil bölümdür. İkinci bölüm (/dev/hda2) iki mantıksal bölüm içeren genişletilmiş bir bölümdür, biri EXT2 dosya sistemi (/dev/hda3) ve ikincisi Linux takas dosyası (/dev/hda5). Sonra hangi dizinlerin hangi bölümlere bağlandığını bilmemiz gerekiyor. Bu bilgiyi görüntüleyen mount komutunu çalıştırıyoruz.

mount I sütunu -t
/dev/hdal / üzerinde ext2 (rw,acl,user_xattr) proc /proc üzerinde proc
(rw) tmpfs /dev/shm üzerinde tmpfs (rw)
devpts /dev/pts üzerinde devpts
(rw,mode=0620,gid=5) /dev/hda3 /home üzerinde ext2 (rw,acl,user_xattr) /dev/
hdc /media/cdrom üzerinde subfs ...

mount komutu bize /home dizininin /dev/hda3 aygıtına bağlandığını gösterir. /home dizinini kaldırırız veya silinen dosyanın üzerine yazma olasılığı olmaması için salt okunur olarak yeniden bağlarız.
Bu ne kadar çabuk yapılırsa o kadar iyi olur; dosyanın üzerine yazılma olasılığı yüksek olduğundan bölüm o kadar uzun süre bağlı kalır.
Dizini ayırmak için şu komutu veriyoruz:

umount /home

Bölümü açmak ve silinen dosyayı kurtarmak için debugfs hata ayıklayıcısını kullanırız. Hata ayıklayıcıda bölümdeki tüm silinen dosyalara ait inode bilgilerini görüntülemek için Isdel komutunu çalıştırırız. (İnode, dosya meta verilerini tutan bir veri yapısıdır. İnode'lar hakkında daha fazla bilgi için [1, 4]'e bakın.)

debugfs /dev/hda3 debugfs
1.35 (28-Aralık~2004) debugfs: Isdel

Inode Sahibi Modu Boyut Bloklar Silinen Zaman 272319 0 100755 3383328
828/ 828 Per 23 Ara 23:45:22 2004 1 silinmiş inode bulundu, satırlar 1-3/3
(SON)

Isdel komutu, 272319 inode numarasıyla temsil edilen bir dosyanın 23 Aralık'ta silindiğini ve 3MB boyutunda (828 bloktan oluşan) olduğunu belirtir. İnode numarasına sahip olduğumuzda stat komutuyla daha detaylı bilgi alabiliriz:

hata ayıklama: istatistik <272319>

Inode: 272319 Tür: normal Mod: 0755 Bayraklar: 0x0 Nesil: 92194859

Kullanıcı: 0 Grup: 0 Boyut: 3383328
Dosya ACL: 0 Dizin ACL: 0
Bağlantılar: 0 Blok sayısı: 6624
Parça: Adres: 0 Sayı: 0 Boyut: 0 ctime: 0x41cb9ee2 — Per 23 Ara 23:45:22 2004

atime: 0x41cb9d68 — Per 23 Ara 23:39:04 2004
mtime: 0x41cb9d68 ~ Per 23 Ara 23:39:04 2004
dtime: 0x41cb9ee2 — Per 23 Ara 23:45:22 2004
BLOKLAR:
(0-ll):582122-582133, (IND):582134, (12-826):582135-582949
TOPLAM: 828

Stat komutu bize silinen dosyanın değiştirilmiş, erişilmiş, değiştirilmiş ve silinmiş tarih ve saatleri dahil olmak üzere çeşitli bilgiler sağlar. (NTFS ve FAT dosya sistemlerinin aksine, Linux EXT2 dosya sistemi bir dosyanın silinmiş tarih ve saatini izler.) stat komutu ayrıca bize BLOKLAR bölümü altında doğrudan, dolaylı ve çift dolaylı blokların sayısını gösterir. (EXT2 dosya sisteminin daha ayrıntılı bir açıklaması için [1, 5]'e bakın). Görünüşe göre tüm bloklar sağlam, yani hiçbir blok üzerine yazılmamış, yani tüm dosyayı kurtarabiliriz.

Dump komutu, kurtarılan dosyayı çağırmak için bir inode numarası ve bir ad argümanı alır:

debugfs: <272319> hda3.recovered dökümü

Hata ayıklayıcıdan çıktığımızda kurtarılan dosya türünü şu şekilde belirleriz:
dosya komutu. Dosya komutu, başlık bilgilerini kullanır
dosyanın türünü belirleyin.

dosya hda3.recovered
hda3.recovered: Zip arşiv verisi, en azından çıkarmak için vl.O

Kurtarılan dosyamız beklendiği gibi bir ZIP arşivi. Prosedürümüzün başarısını kurtarılan dosyamızın karma değerini orijinal dosyanın karma değeriyle karşılaştırarak belirliyoruz (buradaki gösterimiz için elimizde olan). Karmalar eşleşiyor ve bu da dosyayı başarıyla kurtardığımızı gösteriyor. (Ya da orijinalin MD5'i yoksa, bu durumda dosyayı basitçe sıkıştırılmış halden çıkarıyoruz.)

md5sum orijinal.dosya.zip hda3.kurtarıldı ed9a6bb2353ca7126c3658cb976a2dad
orijinal.dosya.zip ed9a6bb2353ca7126c3658cb976a2dad hda3.kurtarıldı

Bu işlemin başarısı bir dizi kritik faktöre bağlıdır.
Birincisi, dosyanın silinmesi ile silinmeye çalışılması arasındaki zaman aralığıdır

kurtarma. Silme ile kurtarma arasındaki süre ne kadar uzun olursa, dosyanın bir kısmının veya tamamının üzerine yazılması olasılığı o kadar artar. İkinci faktör dosya boyutudur. Daha küçük dosyaların (doğrudan bloklara uyanlar) kurtarılma olasılığı, dolaylı ve çift dolaylı blokların kullanımını da gerektirebilecek daha büyük dosyalardan daha yüksektir.

3.3 Önemli Dosyaları ve Truva Atlarını Belirleme

Saldırganların iki temel amacı, bir izinsiz giriş gerçekleştirmek ve kurban sistemde mümkün olduğunca uzun süre kalmaktır. Sistemde gizli kalmak genellikle bir rootkit yükleyerek gerçekleştirilir. Bir rootkit, birkaç önemli sistem dosyasını "Truva atı" sürümleriyle değiştirir. Truva atı sürümleri, saldırganın çalışan işlemler, açık dosyalar veya açık soketler gibi herhangi bir izini göstermemesi dışında orijinal sistem dosyaları gibi çalışır. Genellikle Truva atı olan yardımcı programlar arasında ps (sistem işlemlerini görüntülemek için), netstat (soketleri ve ağ bağlantılarını görüntülemek için) ve top (etkinliğe göre sıralanmış işlem bilgilerini görüntülemek) bulunur.

Truva atı dosyalarını tespit etmenin basit bir yolu karma analizidir.
karma analizi, "dikkat çekici" dosyaların tek yönlü kriptografik karmalarını sistemdeki dosyaların karmalarıyla karşılaştırır. İki karma eşleşirse, bir dosyanın Truva atı sürümüyle değiştirildiği anlamına gelir.

Truva atlarını tanımlamanın ikinci bir yöntemi, bir dizindeki dosyaların inode numaralarını karşılaştırmaktır. Bir dizindeki diğer dosyaların inode numaralarıyla önemli ölçüde uyumsuz olan bir inode numarası, dosyanın değiştirildiğinin bir göstergesi olabilir.

Bir dosya sabit diske kaydedildiğinde ona bir inode numarası atanır.
Kısa aralıklarla kaydedilen dosyaların inode numaraları şu şekilde olacaktır:
ardışık veya neredeyse öyle. Bu aşağıda gösterilmiştir, /bin dizininin içeriklerinin inode numarasına göre sıralanmış (ilk sütunda bulunur) (kısmi) bir dizin listesi görüntülenir.

```
# Is -ali /bin I sort 130091 -rwxr-xr-x 1
root root 59100 Oct 5 11:50 cp 130092 -rwxr-xr-x 1 root root 15516 Get 5 11:50 unlink

130093 -rwxr-xr-x 1 kök kök 161380 11 Ekim 09:25 tar
130094 -rwxr-xr-x 1 kök kök 16556 5 al 11:50 rmdir
130095 -rwxr-xr-x 1 kök kök 26912 5 Ekim 11:50 İçinde
130096 -rwxr-xr-x 1 kök kök 10804 30 Eyl 08:49 ana bilgisayar adı 130097 -rwxr-xr-x 1 kök kök 307488 21 Eyl
17:26 tcsh 569988 -rwr-x kök 2936 04 ps 569990 -rwxr-xr-x 1 kök kök 92110 18 Oca 2004 netstat
```

Dosyaların sabit diske kaydedilme sırası şu şekildedir:
inode numaralarının artan dizisiyle gösterilmiştir. Açıkça bir

ps ve netstat komutları için inode numaralarında anormallik.
Bir dosyanın inode numarası, farklı bir dosya ile değiştirildiğinde değişecektir.

Truva atı orijinal dosyadan çok sonra yüklendiği için Truva atı inode numarası orijinal dosyanınkinden daha yüksek olacaktır. Bu nedenle, Truva atlarını tanımlamanın basit bir yöntemi, özellikle bir rootkit'in parçası olma olasılığı olan dosyalar için "aykırı değerler" olan inode numaralarını aramaktır. Yukarıda gösterildiği gibi, ps ve netstat'ın inode numaraları diğer dosyaların inode numaralarıyla önemli ölçüde uyumsuzdur ve bu da orijinal yardımcı programın bir Truva atı sürümüyle değiştirilmiş olma olasılığını gösterir. Bu, yukarıdaki karma analizinin aksine, dosyaların bilinen Truva atları olduğunun bir garantisi değildir. Her şeye rağmen, daha fazla inceleme haklıdır.

3.4 Yeniden Adlandırılmış Uzantılara Sahip Dosyaları Belirleme

Bir dosyayı gizlemenin en basit yolu dosya uzantısını değiştirmektir.
Örneğin, chix.jpg dosyasını homework.doc olarak değiştirmek, şüpheli içerikli bir dosyayı alıp, onu zararsız görünen bir dosyaya dönüştürür.
Bu teknik özellikle Windows'ta etkili olabilir çünkü Windows, dosya uzantısının dosya türünü gerçekten yansıtıp yansıtmadığına bakılmaksızın, dosyanın uzantısına dayalı bir simge görüntüler.
Daha önce açıklandığı gibi, bir dosya türü başlığında (bazen imza olarak adlandırılır) yansıtılır. Bir dosya başlığı, uygulamalara dosyayı nasıl işleyeceklerine dair bir işarettir. Örneğin, tüm modern Microsoft Office dosyaları aşağıdaki 8 baytlık imzalarla (kalın olarak) başlar:

dOcf lleO albl lael 0000 0000 0000 0000

Uzantısı değiştirilmiş grafik dosyalarını bulmanın bir yolu to ve grep'tir. En iyi yol üç GNU yardımcı programını birleştirin: find, file. ,
Prosedürü bir gösteri yoluyla açıklayın.

1 Sabit diskteki tüm normal dosyaları bulmak için find komutunu kullanın.
2 Bu komutun sonuçlarını , aşağıdaki komutu görüntüleyen dosya komutuna aktarın: Başlık bilgisine göre dosya türü.
3 Bu komutun sonuçlarını grep komutuna aktararak grafikle ilgili anahtar kelimeleri arayın.

Aşağıda tüm grafiksel görüntüleri tanımlamak için üç yardımcı programı birleştiriyoruz yeniden adlandırılmış bir uzantıya sahip olanlar:

```
# find / -type f ! —name' * .jpg — o — name' * .bmp' — o — name' * -png' -printO I xargs
-0 dosya I grep - eğer graphics.files
```

Bunu adımlara böldüğünüzde anlamak daha kolay olacaktır:

1 / argümanı, başlatılacak dizini belirtir; burada kök dizindir.

2 -type f bayrağı, aygıtlar veya dizinler gibi özel dosyaların aksine normal dosyalarla ilgilendiğimizi belirtir. find komutu varsayılan olarak yinelemelidir, bu nedenle esasen / (kök) dizininden başlayarak tüm normal dosyaları yinelemeli olarak bulur.

3 Ünlem işareti (!) parantez içindeki içeriği değiştirir ve uzantısı *.jpg, *.png, *.bmp veya *.tiff olmayan dosyaları işlemek istediğimizi belirtir.

4 printO, yazdırmayı biçimlendirmek için gereken özel bir biçimlendirme komutudur. Bir sonraki komuta yönlendirmek için find çıktısı.

5. Sonuçları *.jpg, *.bmp, vb. uzantılı olmayan dosyaların bir listesini xargs -0'a aktarın; bu, her dosya adını file komutuna gönderir; file, her dosya imzasını değerlendirir ve dosya türünün açıklamasını döndürür.

6 Bu sonuçlar, belirli anahtar sözcükleri aramak için grep'e iletilir. vardır graphics.files dosyasında bulunur. grep için argümanlar arasında büyük /küçük harfe duyarsız arama için i ve PNG, GIF, bitmap, JPEG ve resim anahtar kelimelerinin listesini içeren dosya olan f graphics.files bulunur.

Aramamız yanıltıcı ad ve uzantılara sahip üç dosya tespit etti:

```
# find / -type f ! ( -name
 '*.jpg' -o -name '*.bmp' -o -name '*.png' ) -printO I xargs-0 dosya I grep -if
graphics.files /var/cache/exec: JPEG resim verisi, JFIF stcindard 1.01 /var/log/
ppp/0xl2da2: PC bitmap verisi, Windows 3.x biçimi /var/log/ppp/README.txt: PNG
resim verisi, 8-bit/renk RGB
```

Arama, üç dosyayı doğru bir şekilde tanımladı: bir *.jpg5, bir *.bmp ve bir *.png, gerçek türlerini gizlemek için adı ve/veya uzantısı değiştirilen. Bu teknik, dosya imzası bozulmadan kaldığı sürece doğru şekilde çalışacaktır.

4. Sonuçlar ve Gelecekteki Çalışmalar

Bu makalede açıklanan teknikler, kolluk kuvvetlerinin karşılaşacağı davaların büyük bir kısmı için dijital kanıtları belirleme ve yeniden kurtarmada iyi çalışır. Ancak teknolojideki değişiklikler, özellikle depolama kapasitesindeki artışlar, kolluk kuvvetleri için sorun yaratmaya başlıyor. Örneğin, FBI bilgisayar analizi ve yanıt ekibi (CART), 1999'dan 2003'e kadar davalarda üç kat artış gördü; ancak veri miktarı 46 kat arttı [3]. Ajanların, her biri potansiyel bir kanıt parçası olan milyonlarca belgeye eşit olan terabaytlarca veri depolayan sunucularla karşılaşması alışılmadık bir durum değildir. Kolluk kuvvetleri için kritik soru şudur: Milyonlarca dijital eserden hangisi kanıtlayıcı "kanıt"tır ve hangisi değildir? Bu bölümde açıklanan teknikler, bu kadar muazzam veri sistemlerine iyi ölçeklenemez.

Bazı adli prosedürler otomatikleştirilmiş olsa da - örneğin karma analiz ve aramalar - birçoğu manuel girdi veya insan yorumu gerektirir. Aslında, neredeyse hiçbir geleneksel dijital adli teknik terabayt boyutundaki sistemlere iyi ölçeklenemez. Veri miktarı arttıkça, makul bir zaman diliminde delilleri işlemek için dijital delilleri tanımlama, kurtarma ve inceleme için otomatik prosedürler gerekecektir. Aşağıda, büyük ölçekli sistemlerde kanıtlayıcı delilleri tanımlamak için otomatik bir sistemin temeli olabilecek bir dijital eser taksonomisini açıklıyoruz. Taksonomi, dijital eserleri üç özniteliğe göre kavramsallaştırır: (i) eser içerikleri, (ii) ilişkili meta verileri ve (iii) ortam bilgileri. Bu özniteliklere ait dijital eser değerleri hem dijital hem de tanımlanabilirdir, yani dijital bir eserin tanımlayıcısını (örneğin, dosya adı veya inode numarası) bilerek eser içeriklerini, meta verilerini ve ortam bilgilerini belirleyebilir ve dolayısıyla kurtarabilirsiniz. Sonuç olarak, bu değerleri kurtarabilen ve herhangi bir manuel girdi veya yorumlama ihtiyacını ortadan kaldıran otomatik bir prosedürün geliştirilebileceği düşünülebilir.

Referanslar

[1] B. Buckeye ve K. Liston, Linux'ta silinen dosyaların kurtarılması (www.samag.com/documents/s=7033/sam0204g/sam0204g.htm), 2003.

[2] P. Craiger, Bilgisayar adli tıp prosedürleri ve yöntemleri, ortaya çıkacak Bilgi Güvenliği El Kitabı^ H. Bigdoli (Ed.), John Wiley, New York, 2005.

[3] P. Craiger, M. Pollitt ve J. Swauger, Dijital deliller ve dijital adli bilimler, Handbook of Information Security^ H. Bigdoli (Ed.), John Wiley, New York, 2005'te yayınlanacaktır .

[4] A. Crane, Linux silinen dosyaları geri yükleme kılavuzu (www.praeclarus.demon.co.uk/tech/e2-undel/html/howto.html), [5] S. Pate, 1999.

UNIX Dosya Sistemleri: Evrim, Tasarım ve Uygulama^ John Wiley, New York, 2003 [6] T. Warren, /proc'u keşfetme (www.freeos.com/

articles/2879/), 2003.