

# Güvenli Ortam (Jail): Her Şeye Kadir *root* 'u Sınırlamak\*

Metin KAYA

EnderUNIX Üyesi

Endersys Yazılım Mühendisi

metin at enderunix.org

metin.kaya at endersys.com.tr

<http://www.enderunix.org>

<http://www.endersys.com.tr>

27 Nisan 2008 Pazar EEST 01:11:45

\* Robert N. M. Watson ve Poul-Henning Kamp 'ın <http://watson.org/~robert/freebsd/sane2000-jail.pdf> adresli makalesinden birebir çeviridir.

## ÖZET

Geleneksel UNIX güvenlik modeli sadedir ama çok açıklayıcı değildir. Daha ayrıntılı kontroller eklemek bu açıklayıcı olmamak sorunu giderir; ancak hem sistem yönetimini hem de kodun gerçekleşmesini bir hayli karmaşıklaştırır. Daha karışık yönetim modellerinin olduğu ortamlarda bazı yönetim fonksiyonlarını değişken güvenlik seviyesi altında harici uygulamalara atamak, temel UNIX modeli ve en doğal uzantılar kullanmak sonuç olarak çok uygun değildir. Karşılıklı güvensizliğin olduğu yerde, özellikle de veri bütünlüğü ve güvenliğin korunması gereken yerlerde, bu “uygunsuzluk” hızla “kâbusa” dönüşür.

FreeBSD ‘nin *güvenli ortam* modeli UNIX *root* modelinin sadeliğini korurken işletim sistemi ortamlarını parçalara ayırma yeteneği sunar. Güvenli ortamlarda yetkilendirilmiş kullanıcılar, sistem yöneticilerinin tanımladığı yönetim her bir sanal makine kuralları çerçevesinde, kendi isteklerinin belli alanlarda kısıtlandığı görürler. Bu tarzda sanal makineler oluşturmanın pek çok potansiyel kullanım alanı vardır – örneğin; internet servis sağlayıcılarının kullandığı sanal makineler.

---

Bu çalışma <http://www.servetheweb.com/> desteğiyle yürütülmekte olup FreeBSD işletim sistemini içeren FreeBSD Projesi ’ne armağan edilmiştir. FreeBSD 4.0-RELEASE bu kodu içeren ilk sürümdür. Devam eden çalışmanın destekçisi ise Safeport Network Services, <http://www.safeport.com/>, ‘dir.

# 1. Giriş

UNIX 'teki erişim kontrol mekanizmaları 2 tip kullanıcılar için tasarlanmıştır: yönetici hakkı olan ve olmayan kullanıcılar. Bu bağlamda her girişim için kolay dosya paylaşımına ve süreçler (process) arası iletişime izin veren açık bir sistem sağlanır. UNIX ailesinin bir üyesi olan FreeBSD, bu kalıtsal özellikleri taşımaktadır. Geleneksel olmayan UNIX çevrelerindeki FreeBSD kullanıcıları; ihtiyaçlarını güçlü uygulama desteği, yüksek ağ performansı / işlevselliği ve UNIX güvenlik mekanizmalarında gerçekleşmesi çok zor veya imkânsız olan alternatif güvenlik modellerini düşük sahiplik maliyeti için ayarlamak zorundadır.

Bu konuda bazı (tümü değil) yönetici fonksiyonları güvenilir olmayan veya az güvenilen uygulamalara atamak ve sistem genelinde, eş zamanlı, tüm süreçler arası etkileşim ve paylaşımlarda zorunlu olan kurallar tanımlamak gibi bir fikir akla gelebilir. Günümüz FreeBSD 'sinde böyle bir ortam oluşturmak hem zor hem de maliyetlidir: güvenlik kurallarının yükü kullanıcı uygulamalarının sırtına binecektir ki bu da esnekliğin kaybedilmesinin yanında temel kodların boyutunun ve karmaşıklığının artmasına neden olacaktır.

Bu muhtemel risk, gerçek dünyadan pratik bir uygulama üzerinde daha açık hale gelir: birçok sistem sağlayıcısı FreeBSD işletim sistemini yüksek performansı, ağ tabanlı sunucu ortamı nedeniyle web sitelerine hizmet vermek amacıyla kullanıyor. Ancak bu servis sağlayıcıları aynı zamanda kendi dosyalarını müşterilerinden ve bir müşterinin verilerini diğer müşteriden (kazayla veya bilerek erişebilme ihtimaline karşın) saklamak zorundadır. Öte yandan, müşterilerine web sunucularla ilgili yazılımları kurabilmeleri, kendi servislerini yönetebilmeleri için özerklik vermek zorundadır.

Bu problem açık bir şekilde müşteri süreçleri ile asıl sistem süreçlerinin yalıtılmasını / ayrılmasını (farklı bölümlerin süreçler arasındaki veri transferinin, dosyalarının değiştirilmesinin engellenmesi) gerektirmektedir. Sistemdeki yönetim fonksiyonları tüm sistem genelini etkilememek, bölümler arası veri güvenliği ve bütünlüğünü de devre dışı bırakmamak, koşuluyla bazı uygulamalara bırakılabilir. Ancak UNIX tarzı erişim kontrolü bu çözümü "kötü" olarak nam salacak bir bölümlenmeye sürüklemektedir. *chroot(2)* gibi mekanizmalar ılımlı bir bölümlenme sağlasa da etkin olacakları alan ve performans gibi çok ciddi sorunları da beraberinde getirdikleri bilinmektedir [**CHROOT**].

*chroot(2)* çağrısıyla sürecin görebildiği dosya sistemi, dosya sisteminin belli bir alt alanı olacak şekilde kısıtlanır. Ancak bu kısıtlama sürecin ve ağ yapısının tüm uzayında geçerli olmadığından süreçlerin kendi bölümlerinin dışına çıkması hala mümkündür.

İşte bu noktada etkin sanal makine ortamlarına güçlü bir bölümlenme çözümü sunan, *chroot(2)* gibi mekanizmaları değiştirerek, FreeBSD 'nin "Güvenli Ortam" özelliğini tanımladık. Güvenli ortamdaki bir süreç, sahip olduğu tüm dosyaları, süreçleri, ağ servislerini kendi bölümleri dışındaki dosyalara, süreçlere ve ağ servislerine erişmeden istediği gibi yönetebilir.

Diğer ayrıntılı güvenlik çözümlerinin aksine güvenli ortam kuralları sistem yöneticisinin işini karmaşıktırılmaz. Çünkü her güvenli ortam birbirinden bağımsız, ana sistemle aynı özelliklere sahip ve uygulamalarla uyumlu sanal bir FreeBSD ortamıdır.

## 2. Geleneksel UNIX Güvenliđi veya “Tanrı, *root*: aradaki fark ne?” [UF]

Geleneksel UNIX erişim modelinde her sistem kullanıcıasına sayısal bir *uid* değeri atanır. Her süreç, sahibinin *uid* değeriyle etiketlenir. *uid* ‘ler 2 amaca hizmet eder: isteđe bađlı erişim kontrol mekanizmalarının nasıl uygulanacağını ve özel izinlerin ayarlanıp ayarlanmadığını belirlemek.

İsteđe bađlı erişimlerin kontrolünde koruması öncelikli olan nesne dosyadır. *uid* (ve buna bađlı olarak grubu tanımlayan *gid* ‘ler) kısıtlı bir erişim kontrol listesi gibi, UNIX ‘in nezaketi, her nesne için belli bir kurallar kümesine haritalanmıştır. Her ne kadar yönetim açısından bakıldığında önemli bir ilgi görünse de güvenli ortamlar genelde isteđe bađlı erişim kontrol mekanizmalarının anlamını deđiştirmeyle alakalı deđildir.

Bir sürecin imtiyazlı hakları olup olmadığını anlamak için yapılacak kontrol basittir: “*uid* sıfır mı?”. Yanıt evet ise süreç süper kullanıcı haklarıyla çalışıyordur ve istediđi her şeyi yapabilmesine olanak tanıyan her yere erişim iznine sahiptir. <sup>1</sup>

Üzerinde görüş birliđi sađlanarak 0 *uid* ‘i *root* kullanıcıasına atanmıştır [**ROOT**]. Güvenli ortamın amaçları için bu davranış tamamen uygundur: bu ayrıcalıklı işlemlerin çođu sistem donanımını ve yapılandırmasını, dosya sistemi uzayını ve özel ađ işlemlerini yönetmede kullanılabilir.

Bu modele getirilebilecek kısıtlamaların çođu bariz bir şekilde ortadadır: *root* kullanıcı saldırıların hedefi olabilecek yazılım istismarlarının ihtiyacı olan tek ve merkezi kaynaktır. Saldırganın sistem yönetimini tamamen ele geçirmesi için *root* yetenekleri kümesine sahip olması gerekir. Hatta saldırı riski olmaksızın tek yöneticili bir sistem yapısı yine ciddi tehlikeler taşır: tüm sistemin yönetimini ilgilendiren bir yetkiyi deneyimsiz bir yöneticiye atamak düşündürücüdür. Tüm bu özellikler her şeye kadir *root* hesabını keskin, etkili ve son derece tehlikeli bir araç haline getirmektedir.

BSD ailesindeki işletim sistemleri mevcut sistem yapılandırmasının ve yönetim fonksiyonlarının makine kapatılıp tekil kullanıcı modunda (single user mode) açılmadığı müddetçe *root* tarafından deđiştirilmesini engelleyen “securelevel” mekanizmasını gerçekledi. Bu yapı her ne kadar güvenlik sorunlarının bir kısmını çözse de mevcut *root* yeteneklerinin atanması konusunda hiçbir çözüm sunmamaktadır.

---

<sup>1</sup> ne kadar patentli bir aptal olabileceğinden bahsetmeye gerek bile yok.

### 3. *root* Sorununa Diğer Çözümler

Pek çok işletim sistemi bahsedilen kısıtlamaları adresleyebilmek için sistem kaynaklarına detaylı erişim kontrolü sağlama yoluna girmiştir [BIBA]. Bu çabalar bazen başarılı bazen de başarısızdır; ama hepsi 3 ciddi kısıtlamadan dolayı acı çekmektedir:

İlk olarak; güvenlik kontrollerinin artırılması yönetim sürecinin karmaşıklaşmasına, isteğe bağlı yönetici ve kaynakların yanlış yapılandırılmasına neden olur. Çoğu durumda artan bu karmaşıklık yönetici için büyük hüsrana sonuçlanacak 2 felaketi beraberinde getirir: “çok fazla sorun vardır ve bütün kapılar açıktır” ve “aslında güvenilir olmasa da sisteme güven”.

Sorunun daha da büyümesini en iyi anlatan detaylı güvenlik kontrol araçları sunan çok büyük bir endüstrinin ortaya çıkmış olmasıdır [UAS].

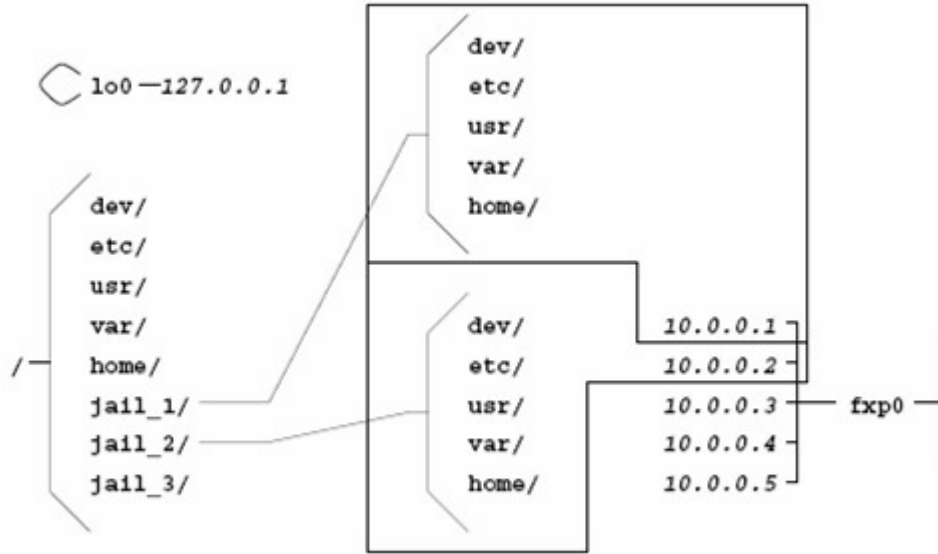
İkincisi; yararlı ayırma yetenekleri ve bunları çalışan koda atamak çok zordur. UNIX ‘teki pek çok ayrıcalıklı işlem birbirinden bağımsız görünür ama aslına bakacak olursak birbirleriyle çok ilgilidirler ve gerçekte birinin imtiyazları diğerine geçebilir. Örneğin; bazı güvenilir işletim sistemlerinde bir sistem yeteneği yedeklemek maksadıyla her dosyayı okuyabilsin diye çalışan bir sürece atanmış olabilir. Öte yandan bu yetenek gerçekte her hangi bir hesaba geçiş yapmak (*su*) ile eşdeğerdir. Çünkü her dosyaya erişebilme yeteneği, sistemin doğrulama mekanizmasını kontrol eden anahtarlama araçlarına da erişimi sağlamaktadır. Benzer şekilde; birçok işletim sistemi yönetim yeteneklerini denetim yeteneklerinden ayırma girişiminde bulunmaktadır; fakat “yönetim yetenekleri” yöneticiye “denetleme yeteneklerini” kendine veya başka bir kullanıcıya atama izni vermektedir – ayırma yeteneğinin tuzağı.

Sonuncusu; yeni güvenlik özellikleri sıklıkla yeni güvenlik yönetimi arayüzlerini gerektirmektedir. Detaylı yetkilendirme özelliğinin, UNIX ‘teki *setuid* mekanizmasının yerini alması uygulamaların eskiden “çalışmadan önce *root* hakkı ile çalıştırılıp çalıştırılmadıklarını” kontrol ederken artık “*root* hakkı ile çalıştırılmalarına gerek olmadığı” bilinip bilinmemesini” kontrol etmelerini gerektirmektedir. İmtiyazlı çalışan ve diğer programları çalıştıran uygulamalar için şu an gönüllü olarak düzenlenmesi gereken yeni ayrıcalıklar kümesi mevcut. Bu değişiklikler var olan uygulamalarla büyük uyumsuzluğa neden olabilir ve farklı sistemlerdeki değişik güvenlik mekanizmalarıyla uğraşmak istemeyen uygulama geliştiricileri için hayatı çok zorlaştırabilir [POSIX1e].

## 4. Güvenli Ortam Bölümleme Çözümleri

Güvenli ortam, bölümleme sorunlarının çoğunun üstesinden kolaylıkla gelir. Bir FreeBSD ortamını detaylı erişim kontrol mekanizmasına dönüştürmekten çok, bu ortamı bir yönetim platformu (süreçler, dosya sistemi, ağ kaynakları) ve güvenli ortam alt bileşenlerine ayırdık. Bu sayede mevcut UNIX güvenlik modelini sürdürüp aynı zamanda *root* kullanıcısının etkin olduğu alanı kısıtlarken her güvenli ortamda çok sayıda kullanıcı ve bir imtiyazlı *root* kullanıcısı bulunmasına izin verebiliriz. Sonuç olarak FreeBSD sistemin yöneticisi makineyi farklı güvenli ortamlara ayırabilir ve tüm sistemin kontrolünü kaybetmeden her bir güvenli ortamın *root* kullanıcısına erişimi sağlayabilir.

Bir bölümdeki süreç “kafeste” diye tabir edilir. Yeni bir FreeBSD sistemi kurulduktan sonra ilk kez başlatıldığında kafeste hiç süreç yoktur. Bir süreç kafese alındığında, yani güvenli ortamda oluşturulduğunda, bu süreç ve bu sürecin çocukları (child process) aynı güvenli ortamda bulunur. Bir süreç yalnızca bir kafeste bulunabilir ve bu kafesten çıkması mümkün değildir. Ayrıcalıklı bir sürecin *jail(2)* sistem çağrısında bulunmasının ardından kafesler oluşturulur. Her *jail(2)* sistem çağrısında yeni bir kafes oluşturulur. Yeni bir sürecin bir kafese girebilmesinin tek yolu, hali hazırda bu kafeste bulunan bir süreçten kafese erişim iznini miras olarak almasıdır. Süreçler, oluşturuldukları veya içinde buldukları kafesi asla terk edemez.



Şekil 1: 2 Güvenli Ortamdan Oluşmuş Makinenin Şematik Diyagramı

Bir kafeste bulunmanın getirdiği kısıtlamalar vardır: görünür dosya sistemi *chroot(2)* tarzındadır, ağ kaynaklarını yalnızca yetkilendirilmiş bir IP adresi yönetebilir, sistem kaynaklarını ve imtiyazlı işlemleri yönetme yetkisi kesin bir şekilde kısıtlanmıştır ve yalnızca aynı kafesteki süreçler birbirleriyle etkileşimde bulunabilir.

Güvenli ortamlar *chroot(2)* 'nin dosya sistemi konusunda sağladığı avantaja sahiptir. Bir kafes oluşturulduğunda bu kafesin görebildiği yer, dosya sisteminin belli bir kısmıyla sınırlanır. Süreçler, adresleyemedikleri dosyalara karışamaz ve bu sayede kafes dışındaki dosyaların bütünlüğü ve gizliliği korunmuş olur. *chroot(2)* 'de meydana gelebilecek patlakları önlemek amacıyla geleneksel mekanizmalar devre dışı bırakıldı. Ümit edilen ve belgelendirilen yapılandırmada her kafesin kendine özel bir dosya sistemi *root* 'u ve standart FreeBSD dizin yapısı vardır; ancak bu yapı güvenli ortam tarafından istibdat altına alınmamıştır.

Her güvenli ortama tekil bir IP adresi atanmıştır: kafesteki süreçler içeriye gelen veya dışarı giden bağlantılarda başka bir IP adresi kullanamaz; ağ servislerinin belli bir güvenli alanda kısıtlı kalması bu şekilde gerçekleşmektedir. FreeBSD tüm IP adreslerini bağlama girişimlerini, belli bir IP adresine bağlama girişimlerinden ayırt edebildiği için tüm IP adresi bağlama istekleri belli güvenli ortam adreslerine yönlendirilir. Ayrıcalıklı çağrılarla bağlantılı bazı ağ işlevleri önerilen çalışma şeklinin, IP adreslerinin aldatılması ve olası yıkıcı ağ trafiğinin devre dışı bırakılması, doğası gereği toptan devre dışı bırakılmıştır.

*root* ayrıcalığı olmaksızın çalışan süreçler kafesin içi ve dışıyla ilgili çok az değişiklik fark edebilir. Öte yandan *root* haklarıyla çalışan süreçler, yaptıkları sistem çağrılarının pek çok kısıtlamayla karşılaştığını görecektir. Bazı çağrılarına erişim hatasıyla yanıt verilecektir – örneğin; bir aygıt düğümü oluşturma girişimi başarısız olacaktır. Diğer isteklere ise kısmen yanıt verilecektir – örneğin; erişilebilir tüm adreslere yapılacak port atama girişimleri, sadece o kafese ait IP adresine port atanmasıyla sonuçlanacaktır. Bunun dışındaki çağrılar başarılı olacaktır: *root*, görünür dosya sisteminde bulunan herhangi bir kullanıcı ait dosyayı okuyabilir.

Kafesteki süreçler, güvenli ortamın dışındaki süreçlerle haberleşemediklerini fark edeceklerdir – kafesteki süreçler güvenli ortamın dışındaki süreçlere sinyal gönderemedikleri gibi bu süreçlere hata ayıklayıcılar ile de erişemezler hatta bu süreçleri *sysctl* ve süreç dosya sistemini takip eden mekanizmalarda da göremeyeceklerdir. Güvenli ortamdaki süreçler kafes dışındaki süreçlerle sadece izin verilen yöntemlerle, saklı kanallar ve arayüz üzerinden gerçekleşen iletişim mekanizmaları, haberleşebilirler – örneğin; IP arayüzü üzerinden soket aracılığıyla başka bir süreçle iletişim kurulabilir.

Bu girişimlerin standart FreeBSD API ve ana çatısını alıkoymasını engellemek için tüm uygulamalar etkilenmeden çalışacaktır. Telnet, FTP ve SSH gibi standart sistem servisleri, çoğu 3. parti uygulamalarda olduğu gibi, popüler Apache de dâhil olmak üzere normal davranacaktır.

## 5. Kafes Uygulaması

*root* izinleriyle çalışan süreçler kafeste olduklarında normalde yapabildikleri pek çok işi yapma yeteneklerinin elinden alındığını görür – özellikle de kafesin dışını etkileyen işlemlerde:

- Çalışan çekirdeği (kernel) doğrudan erişimle değiştirmek ve çekirdek modülü yüklemek yasaklanmıştır.
- Ağ yapılandırmasını, arayüzünü, adreslerini ve yönlendirme tablosunu değiştirmek yasaktır.
- Dosya sistemi bağlamak ve çözmek (mount/unmount) yasaktır.
- Aygıt düğümü oluşturmak yasaktır.
- *raw*, *divert* ve *routing* soketlere erişim yasaktır.
- *sysctl* ayarları gibi çekirdeğin çalışma zamanı parametrelerini değiştirmek yasaktır.
- Güvenlik seviyesini (securelevel) değiştirmek yasaktır.
- Kafesle ilgili olanlar hariç, ağ kaynaklarına erişim yasaktır.

Diğer ayrıcalıklı etkinliklere güvenli ortamın sınırları dâhilinde kaldığı müddetçe izin verilir:

- Kafesteki herhangi bir sürece sinyal gönderilebilir.
- Kafesteki herhangi bir dosyanın sahibi ve çalışma izinleri dosya bayrakları olanak veriyorsa istenildiği gibi değiştirilebilir.
- Güvenli ortamdaki herhangi bir dosya, bayrakları izin veriyorsa silinebilir.
- Kafese ayrılan TCP ve UDP port numaraları yine kafese tahsis edilmiş IP adresine atanabilir (IN\_ADDRANY ile gerçekleştirilen port atamasının sonucu kafese ayrılan IP adresine yönlendirilecektir)
- *uid/gid* uzayında çalışan süreçler – kafes diğer dosya sisteminden yalıtıldığından – istediği gibi çalışmakta serbesttir.

Bu kısıtlamalar *root* imtiyazıyla çalışan süreçleri sınırlar, çoğu uygulamanın engele takılmadan çalışmasını sağlar; ama kafesin dışındaki süreçleri ve sistemin genel yapılandırmasını etkileyecek çağrılarını yasaklar.

## 6. FreeBSD Çekirdeğinde Güvenli Ortamın Gerçeklenmesi

### 6.1. *jail(2)* Sistem Çağrısı, Bellek Alma, Başvuru Sayısının Saklanması ve `struct prison` 'ın Bellek İadesi

*jail(2)* sistem çağrısı FreeBSD 'de isteğe bağlı bir sistem çağrısı olarak gerçekleştirilmemiştir. Diğer sistem çağrıları derleme zamanı seçeneği olarak çekirdek yapılandırma dosyası tarafından kontrol edilir; ancak önemsiz ayak izi nedeniyle *jail(2)* gerçekleştirilmesi FreeBSD 'de bir standart haline getirilmiştir.

*jail(2)* sistem çağrısının basit ve sade tasarlanmıştır: bir veri yapısı için bellek alınır ve verilen argümanlara göre yapılar kurulur. Bu veri yapısı mevcut sürecin `struct proc` yapısına iliştilir, başvuru sayısı 1 olarak atanır ve *chroot(2)* sistem fonksiyonu çağrılarak görev tamamlanır.

Yeni süreç oluşturma ve mevcut bir süreci sonlandırma durumları başvuru sayısını etkiler ve başvuru sayısı 0 olduğunda veri yapısı için bellekten alınan yer iade edilir. Kafesteki her yeni süreç kendisini etkin bir şekilde güvenli ortama koyan başvuru sayısını kalıtsal olarak alır.

*jail* veri yapısını güvenli ortam oluşturulduktan sonra değiştirme mümkün değildir ve kafesin dışında yaratılmış her hangi bir süreci kafese almanın bir yolu yoktur.

### 6.2. *chroot(2)* Özelliğini Görünür Dosya Sistemi Açısından Güçlendirme

*chroot(2)* 'nin kısıtladığı dosya sistemini aşmanın çeşitli yolları bilinmektedir. *chroot(2)* 'nin bir güvenlik mekanizması olması zaten niyet edilmemiştir ama *ftp* servisi anonim *ftp* erişimlerinde büyük oranda *chroot(2)* 'nin sağladığı güvenlik mekanizmasına dayanmaktadır.

Öz yinelemeli (recursive) *chroot(2)* çağrıları, “..” ve *fchdir(2)* tabanlı çağrılar olmak üzere *chroot(2)* sınırlarından kurtulmanın 3 temel yolu vardır. Tüm bu açıklar gösteriyor ki *chroot(2)* görünür dosya sistemini kısıtlamada çok başarılı değil.

Bu saldırıları sezmek ve önlemek amacıyla *chroot(2)* 'nin sınırlarının öğrenilmesine karşı yeni kodlar yazıldı. Ayrıca *chroot(2)* dizininde öz yinelemeli gezinme yapılması ve *chroot(2)* dosya tanımlayıcıları açıkken yeni bir *chroot(2)* çağrısının yapılması yasaklandı.

### 6.3. Süreç Görünürlüğünün ve Etkileşiminin Kısıtlanması

Çekirdekte bir sürecin başka bir süreci etkileyip etkilemediğini gösteren bir makro zaten mevcuttur. Bu makro *uid* ve *gid* değerlerinin karmaşık kontrolünü yapar. Bu makronun karmaşıklığının IOCCC giriş seviyesinin düşük kenarına kadar yaklaştığı düşünülüyordu ve bu nedenle makro daha önce bahsedilen kontrollere ek olarak kafesle ilgili kontrolleri de gerçekleştiren `p_trespass(p1, p2)` şeklinde bir fonksiyona dönüştürüldü. Bu kontroller; köklerdeki süreç kafeste ve hedef süreç aynı güvenli ortamda değilse başarısızlıkla sonuçlanır.

Süreçlerin görünürlüğü FreeBSD 'de 2 mekanizma sayesinde kontrol edilir: *procfs* dosya sistemi ve *sysctl* ağacının bir alt ağacı. Bunların her ikisi de yalnızca aynı kafesteki süreçlerin birbiriyle haberleşmesini sağlayacak şekilde değiştirildi.

#### 6.4. Yalnızca Bir IP Adresinin Kullanılmasına Yönelik Kısıtlama

TCP ve UDP erişimlerini sadece bir IP adresine kısıtlama işi tamamen “protokol kontrol blokları” (protocol control blocks) kodunda yapılmıştır. Kafesteki bir süreç bir sokete bağlandığında sürecin verdiği değil kafes için daha önceden tahsis edilmiş IP adresi kullanılır. BSD tabanlı TCP/IP ağ yapıları, 127.0.0.1 sihirli IP adresli özel *loop-back* arayüzüyle övünür. Bu arayüz yerel makinede bulunan sunucularla haberleşen süreçler tarafından sıklıkla kullanılır ve sonuç olarak kafesler için ihtiyaç duyulan arayüzlerdir. Kafeslerde bu durumun düzgün değerlendirilebilmesi için bağlantı kurulum aşamaları da düzenlenmelidir – kafesteki 127.0.0.1 IP adresleri o kafese ayrılan IP adresiyle değiştirilmelidir.

Son olarak da ağ yapılandırması ve bağlantı durumuyla ilgili API ‘ler sadece kafese tahsis edilmiş IP adresiyle ilgili geçerli bilgileri raporlayacak biçimde değiştirildi.

#### 6.5. İstenilen Aygıt Sürücülerini Kafes Yapısından Haberdar Etme

Kafes yapısından haberdar edilmesi gereken aygıt sürücüsü çiftinin teki *pty* sürücüsüdür. *pty* sürücüsü telnet, ssh, rlogin ve X11 uç birimi (terminal) penceresi programları gibi servislerin çalışması için gerekli sanal uç birimler sunar. Bu nedenle kafesler *pty* sürücüsüne erişmeye ihtiyaç duyarlar ve kodlar bu nedenle *pty* sürücüsünü aynı anda birden fazla kafesin erişimine kapatacak şekilde değiştirilmeliydi.

#### 6.6. Güvenli Ortamdaki Süper Kullanıcı Haklarının Genel Olarak Kısıtlanması

Bu konu gerçekleşmesi en sade ama en sıkıcı olandır. Sıkıcıdır; çünkü çekirdeğin süper kullanıcıya izin verdiği tüm yerler elle gözden geçirilmelidir. Sadedir; çünkü güvenli ortam süper kullanıcısının çok az yerde olmasına müsaade edilir. FreeBSD-4.0 çekirdeğindeki 260 kontrolün yalnızca 35 ‘i güvenli ortam *root* ‘una izin verecektir.

FreeBSD çekirdeğindeki sürücüler veya yeni kodlar otomatik olarak kafeslerden haberdardır (kafes süper kullanıcısının ayrıcalıklarını reddedeceklerdir). Çünkü güvenli ortam *root* ‘larının ayrıcalık kabul etmemesi ön tanımlı davranıştır. Bu korumanın diğer parçası kafes süper kullanıcısının *mknod(2)* sistem çağrısıyla yeni aygıt düğümü oluşturamamasıdır. Bu yüzden sistem yöneticileri güvenli ortam dosya sistemi ağacındaki belli aygıtlar için aygıt düğümleri oluşturmaksızın kafeste etkin sürücü bulunmayacaktır.

Bu işin yan etkisine karşı *suser(9)* toparlandı ve sadece kafes özelliği için değil gelecekteki ayırma özellikleri için de geliştirildi.

#### 6.7. Gerçekleme İstatistikleri

*suser(9)* arayüzündeki değişiklik yaklaşık olarak 100 kaynak kodu dosyasında 350 satırın düzenlenmesidir. Bu değişikliklerin büyük bir kısmı otomatik betiklerle yapıldı.

Güvenli ortamın gerçekleşmesi için 50 dosyaya yaklaşık 200 satır kodun eklenip toplam 200 satır olan 2 yeni çekirdek dosyası yazıldı.

## 7. Kafes Yönetimi ve Kafes Dosya Sistemi

### 7.1 Yeni Bir Kafes Oluşturmak

*jail(2)* çağrısı her ne kadar çok farklı yapılandırmalarla yapılabilse de her kafesin bir tam bir FreeBSD kurulumu olması önerilmektedir. Yani; gerekli çalıştırılabilir sistem dosyaları, veri dosyaları ve kendine has */etc* dizininin olması tavsiye edilir. Bu tarz bir yapı kafeslerin birbirinden tamamen bağımsız olmasını, bir kafesin diğerine karışma olasılığının en aza indirgenmesini ve güvenilmeyen kullanıcılara daha rahat *root* izni verilmesini sağlayacaktır.

Bir makine üzerinde güvenli ortam kullanılacaksa 2 tip çalışma ortamına başvururuz: hizmet veren çalışma ortamı ve kafes ortamı. Hizmet veren ortam arayüzleri yapılandıran, kafesi gerçekleyen ve başlatan gerçek işletim sistemidir. Özellikle sanal FreeBSD makineleri şu an kullanımda olan güvenli ortamlardır. Kafesleri kullanmadan önce hem hizmet veren ortamı hem de kafesle ilgili arayüzleri yapılandırmak olası çakışmalardan kaçınmak için önemlidir.

Kafese alınana sanal makinelere atanan IP adresi normal takma (alias) IP adresi tanımlama yöntemiyle atanır ve bu IP adresleri hizmet veren ortamdaki uygulamaların kullanımına açıktır. Eğer gerçek işletim sistemindeki uygulamaların kafese ait IP adresini kullanmasını istemiyorsanız bu uygulamaları, gerekli adresi yalnızca dinleyecek şekilde yapılandırmanızdır.

Kafeslerin kullanımda olduğu üretim ortamlarının çoğunda hizmet veren ortama bir adet IP adresi ayrılır ve bir grup IP adresi de her kafesinki farklı olacak şekilde güvenli ortamların kullanımına ayrılır. Bu durumda gerçek işletim sistemindeki ağ uygulamalarının yalnızca gerçek işletim sisteminin IP adresini dinlemesini sağlayacak bir yapılandırma yeterlidir. Bu yapılandırma genellikle *inetd* ve SSH için uygun IP adresini belirlemekten ve adres aralığını sınırlandıramayan *sendmail*, *port mapper* ve *syslogd* servislerinin devre dışı bırakılmasından ibarettir. Gerçek işletim sistemindeki diğer 3. parti yazılımlar da bu şekilde yapılandırılmalıdır. Aksi halde güvenli ortam kullanıcıları kendileriyle hiç ilgisi olmadığı halde gerçek işletim sistemindeki servislerden haberdar olacaktır. Bu durum bazen gerçekten de istenen davranıştır.

Kafesler ayrıca her müşteriye özel şekilde yapılandırılmalıdır. Bu durum her kafes için FreeBSD 'nin küçük bir sürümünün kurulmasını, genellikle */usr/jail* veya */data/jail* gibi bir alt dosya sistemi ayrılmasını içermektedir. Yapılandırmanın nasıl yapılacağını *jail(8)* kılavuz sayfasından bulabilirsiniz; ancak genelde bu süreç FreeBSD kurulum ortamında otomatikleştirilebilir.

Normal bir FreeBSD kurulumuyla kafes için yapılacak kurulum arasındaki en bariz fark kafes kurulumunda sınırlı sayıda aygıt düğümünün oluşturulmasıdır. Bunun için MAKEDEV(8) güvenli ortam için gerekli az sayıda aygıt düğümünü oluşturacak şekilde değiştirildi.

Saklama etkinliğini artırmak için sistem ağacından çok sayıda ikili (binary), kafeste kendilerine ihtiyaç duyulmayacağı için, silinebilir. Çekirdek, açılış yükleyicisi ve ilgili dosyalar, donanım ve ağ yapılandırma araçları silinebilecek ikililer arasında yer almaktadır. Kafes oluşturulduktan sonra kafesi yapılandırmanın en kolay yolu tekil kullanıcı modunda açmaktır. Her ne kadar sistem ağacının ön tanımlı bir parçası olmasa da *sysinstall* yönetim aracını yapılandırma için kullanabilirsiniz. Bu araçlar kafes ortamında çalıştırılmalıdır. Aksi

halde gerçek işletim sistemi bu yapılandırmadan etkilenecektir [çevirmenin notu: aşağıdaki komutlar FreeBSD-4.0 'a özel olup güncel bilgi için [http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/jails-build.html](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails-build.html) adresine bakınız].

```
# mkdir /data/jail/192.168.11.100/stand
# cp/stand/sysinstall /data/jail/192.168.11.100/stand
# jail /data/jail/192.168.11.100 test_alan_adi 192.168.11.100 /bin/sh
```

*jail* komutunun ardından kabuk (shell) kafes ortamına geçer ve bundan sonraki tüm komutlar kabuktan çıkılmadığı sürece güvenli ortamlarla sınırlıdır. Eğer kafesin ağ arayüzü yapılandırılmamışsa kafes ağa erişemeyecektir.

Güvenli ortamın başlangıç yapılandırması kafeste çalışmayacak servislerden gelecek olan uyarıları bastırarak şekilde yapılabilir. Ayrıca her FreeBSD sistemi için yapılması gereken ayarlar kafesler için de yapılmalıdır:

- Boş bir */etc/fstab* oluşturulması
- *portmapper* 'ın devre dışı bırakılması
- Yeni takma adların tanımlanması
- Arayüz yapılandırmasının devre dışı bırakılması
- DNS çözücüsünün ayarlanması
- *root* şifresinin ayarlanması
- Zaman ayarının yapılması
- Yerel hesapların oluşturulması
- Gerekli paketlerin kurulması

## 7.2. Kafesi Başlatmak

Kafesler normalde kendi */etc/rc* betiklerinin, daha önceki bölümde anlatılan, kendi kabuklarının çalıştırılması gibi başlatılır. Kafes başlatılmadan önce ilgili tüm ağ yapılandırılması yapılmış olmalıdır. Bu yapılandırma genelde kafese uygun IP adresinin ayrılması, IP filtreleme, yönlendirme, bant genişliği biçimi ve ağ özelliklerinin ayarlanmasından oluşur. Ayrıca kafesteki süreçleri detaylı incelemek, hataları ayıklamak istiyorsanız *proc* dosya sistemini de bağlamalısınız.

```
# ifconfig ed0 inet add 192.168.11.100 netmask 255.255.255.255
# mount -t procfs proc /data/jail/192.168.11.100/proc
# jail /data/jail/192.168.11.100 test_alan_adi 192.168.11.100 /bin/sh /etc/rc
```

sysctl ayarlarının çok az bir kısmı uyarı verecektir ama sonuçta tek bir IP adresine bağlanmış ve çalışma ortamı yalıtılmış bir kafes oluşur. FreeBSD bir makineye normal erişim süreci kafesler için de geçerlidir: *telnet*, *login*, *shell*.

```
% ps ax
PID  TT  STAT    TIME  COMMAND
228  ??  SsJ    0:18.73  syslogd
247  ??  IsJ    0:00.05  inetd -wW
249  ??  IsJ    0:28.43  cron
252  ??  SsJ    0:30.46  sendmail: accepting connections on port 25
291  ??  IsJ    0:38.53  /usr/local/sbin/sshd
```

```
93694 ?? SJ 0:01.01 sshd: rwatson@tty0 (sshd)
93695 p0 SsJ 0:00.06 -csh (csh)
93700 p0 R+J 0:00.00 ps ax
```

Yukarıdaki çıktıda göze çarpan en bariz noktalar: herhangi bir *init* süreci yok, ortada bir çekirdek seviyesi yok ve her süreçte kendisinin kafeste olduğunu gösteren bir J bayrağı var.

Tüm FreeBSD sistemlerde olduğu gibi kullanıcı hesapları açılabilir/silinebilir, e-posta gönderilebilir, günlük (log) tutulabilir, paket kurulabilir ve güvenlik açıkları olan yazılımlar kurmanız, yanlış yapılandırmalarınız nedeniyle sisteme istemediğiniz kişiler sızabilir. Fakat tüm bunlar sadece kafesin sınırları dâhilinde kalır.

### 7.3. Kafes Yönetimi

Güvenli ortam yönetimi 2 açılı ilginç bir manzaradır: kafesin içi ve hizmet veren ortam. Yukarıda anlatıldığı gibi kafesin içi her ne kadar bazı kısıtlamalar olsa da normal bir FreeBSD kurulumu gibidir. Tek fark sistemi kapatma (shut down) kısmıdır: kafesteki süreçler kendi aralarında sinyalleşebilir, tüm süreçlerin ölmesine izin verilebilir ama sistemi tekrar ayağa kaldırmak kafesin dışına taşmayı gerektirebilir.

Kafesin dışında kısıtlamalar olduğu gibi pek çok yapılabilecek iş de vardır. Kafes işin aslında gerçek sistemin bir alt kümesidir: kafes dosya sistemi gerçek dosya sisteminin bir parçasıdır ve gerçek sistemdeki süreçler tarafından doğrudan müdahale edilebilir. Kafesteki süreçler gerçek sistem süreçlerinin arasında listelenir ve diğer süreçler gibi sinyal gönderilebilir, hata ayıklaması yapılabilir. Gerçek sistemdeki *proc* dosya sistemi */proc/procnum/status* dizininde kafes ortamındaki süreçleri yönetmenizi sağlayan araçlar sunmaktadır. Ön tanımlı yapılandırma kafesteki ayrıcalıklı süreçlerin kafesin alan adını belirlemesine izin veriyor ama kafeste kötü niyetli kullanıcılar varsa bu durum sakıncalı olabilir. Kafes süreçlerinin kendi alan adlarını belirlemesini engellemek için “security.jail.set\_hostname\_allowed” *sysctl* değeri 0 olmalıdır.

Çok sayıda kafesin bulunduğu bir ortamda farklı kafeslerdeki süreçler aynı *uid* ve *gid* değerine sahip olabilir. Bu çakışma sadece gerçek işletim sisteminde fark edilebilir. Çünkü bir kafesteki süreçler kendi kafeslerinden başka bir yerden asla görülemezler. Sistem yöneticileri bu çakışmanın farkında olmalıdırlar, yoksa kafaları karışabilir ve istenmeyen sonuçlara yol açabilirler.

Güvenli ortama alınmış süreçler güvenlik duvarı kuralları da dâhil olmak üzere ağ kaynaklarına karışamazlar. Kafeslerin bant genişliği sınırlandırılabilir ve çalıştırabilecekleri servisler kısıtlanabilir.

Güvenli ortamların yönetimi FreeBSD ‘nin gelecek sürümlerinde daha da geliştirilecektir. Bazı potansiyel gelişmeler bu makalenin sonraki kısımlarında ele alınacaktır.

## 8. Gelecekle İlgili Öneriler

Çok fazla sayıda yetenekleri bulunan güvenli ortamlar geniş bir alanda gelişerek yayılmaktadır.

### 8.1. Gelişmiş Sanallaştırma

Daha önce de bahsedildiği üzere güvenli ortam kodları kafesin süreçlere, dosyalara, ağ kaynaklarına ve ayrıcalıklı servisler erişimini kısıtlamaktadır. Sanallaştırma veya kafes yapılarını tam fonksiyonlu FreeBSD sistemler haline getirme azami sayıda uygulamanın desteklenmesine ve kafeste çok sayıda servisin çalışabilmesine olanak sağlar. Ancak mevcut kodda sanallaştırmayla ilgili bazı kısıtlamalar bulunmaktadır ve bunları kaldırmak kafes yapılarını daha yetenekli kılacaktır. Çok fazla dikkat isteyen 2 bölüm bulunmaktadır: ağ ve zaman (scheduling) kaynakları.

Şu an için her kafese 1 IP adresi atanabilir ve kafesle ilgili tüm iletişim bu IP adresiyle sınırlandırılabilir. Ağ arayüzünün sanallığını geliştirip aynı kafese birden fazla IP adresinin atanabilmesi ve IPv6 desteği için çalışmalar yapılmaktadır. Ayrıca *raw* IP paketleri tüm arayüzlerle ilişkilendirildiğinden şu an için kafeslerin erişimine kapalı olan *raw* soketlerin serbestleştirilmesi gerekmektedir. *raw* soketlerin sınırlandırılması kafesleri daha güvenli hale getirip kafesler için *ping* ve diğer ağ hata ayıklama ve değerlendirme araçlarının kullanılabilmesi anlamına gelmektedir.

Diğer büyük ilgi alanı ise bir kafesin CPU kullanımının diğer kafese olan etkisinin sınırlandırılmasıdır. Bu durum özellikle kafesteki bir sürecin kendi zamanlayıcı algoritmasıyla oynadığında gereklidir. Bu alanda yapılması gereken öncelikli işler FreeBSD-2.2.X 'te yama olarak bulunmaktadır ve bunların kafesler arasındaki ayırma uygulanması gerekmektedir [LOTTERY1] [LOTTERY2]. Buna karşın mevcut zamanlayıcı algoritması zaman paylaşımı esasına dayanmaktadır ve FreeBSD çekirdeği şu an için süreçlerde gerçek zaman önceliğini desteklememektedir. Mevcut yapıyla kafeslerin tamamen ayrılması mümkün değildir.

### 8.2. Gelişmiş Yönetim

Kafes yönetimiyle ilgili bazı konular, kafes oluşturma ve başlatma gibi, iyi belgelendirilmiştir; ancak kafesin kapatılması, süreçlerin sonlandırılması iyi analiz edilmemiştir. Şu an geçerli olan çekirdek süreç yönetimi kafes merkezli süreçlerin yönetimi konusunda çok yetenekli değildir. Örneğin; kafesteki bir sürecin kafesteki tüm süreçlere sinyal göndermesi mümkündür fakat kafesin dışından sadece bir kafesteki tüm süreçleri hedef alan sinyaller göndermek olanaksızdır. Eğer güvenli ortam kodları kafesin davranışlarını etkin bir şekilde sınırlayabilirse kafesi temiz bir biçimde kapatmak basit bir iş olacaktır. Benzer şekilde, bir kafesi kendi içinden kapatmak tam olarak tanımlanmamıştır ve geleneksel kapatma işlemi akıllara gerçek sistemle birlikte gelmektedir. Bu nedenle hem çekirdekte hem kullanıcı uzayında çok sayıda aracın iyileştirilmesi gerekmektedir.

İlk olarak çekirdek merkezli yönetim mekanizmalarını gerçeklemek önemlidir. Bunun için her kafese *uid* ve *gid* 'den farklı olarak tekil bir kafes kimlik numarası verilmelidir. Yeni bir *jaillkill()* sistem çağrısı istenilen bir kafes kimlik numarasına sinyal gönderilmesine kafesteki süreçlerin etkin biçimde sonlandırılması için izin vermelidir. Ayrıca, kafes kimlik numarası

sayesinde her kafes kendi alan adını gerçek sisteme bulaşmadan atayabilir.

Kullanıcı uzayında daha dikkatli tasarlanmış açılış ve kapanış araçları da çok önemlidir. Geleneksel FreeBSD sistemlerde *init* süreci sistemi başlatır ve kapanış sürecine yardım eder. Benzer bir teknik kafesler için de kullanılabilir: *jailinit* temiz bir açılış/kapanış yapacak, kafesin */etc.rc.shutdown* ve diğer kapanış fonksiyonlarını yönetecektir. *jailinit*, gerçek sistemin tüm süreçleri sonlandırıp çekirdeği kapatması gibi, hizmet veren ortamdan kafese kapanışla ilgili yönetim mesajları gönderebilmelidir.

Hizmet veren sistem üzerinde kafesi daha kolay başlatıp sonlandıran, güvenli ortam için gerekli dosya sistemini yöneten ve çalışma zamanında daha esnek kafes yönetimi sağlayan araçların yazılması güvenli ortamları çok daha gelişmiş seviyeye getirecektir.

Güvenli ortamla ilgili hem raw işlevleri hem de yönetim araçları gelişmeye devam edecektir. Kafes konusu FreeBSD komitesinde büyük yankı uyandırdı ve FreeBSD 'nin gelecek sürümlerinde büyük ilerlemeler kaydedilmesi beklenmektedir.

## 9. Sonu

Kafes zelliđi, FreeBSD ‘yi sade ve anlamlı bir gvenlik mekanizmasına kavuřturup sanal makine blmleri arasında ynetici haklarının paylařılmasını sađlar.

Gvenli ortamın gereklenmesi hizmet veren ortam zerindeki kaynaklara eriřimin kısıtlanmasına dayanmaktadır. Bu bađlamda srelere, dosyalara, ađ kaynaklarına ve imtiyazlı iřlemlere eriřim kısıtlanmıřtır. Detaylı eriřim kontrol mekanizmasının yknden kaınılıp kafes blmleri arasındaki tutarlı arayzler kullanılmıřtır.

Kafes zelliđi zellikle sanal zel sunucu (virtual private server) servislerinde kullanılmaktadır.

Kafes yapısı ilk olarak FreeBSD-4.0-RELEASE ‘in temel sistem kodunda yer almıř olup *jail(2)* ve *jail(8)* kılavuz sayfalarıyla belgelendirilmiřtir.

## Notlar & Kaynaklar

**[BIBA]:** K. J. Biba, Integrity Considerations for Secure Computer Systems, USAF Electronic Systems Division, 1977

**[CHROOT]:** Dr. Marshall Kirk McKusick, private communication: “According to the SCCS logs, the chroot call was added by Bill Joy on March 18, 1982 approximately 1.5 years before 4.2BSD was released. That was well before we had ftp servers of anysort (ftp did not show up in the source tree until January 1983). My best guess as to its purpose was to allow Bill to chroot into the /4.2BSD build directory and build a system using only the files, include files, etc contained in that tree. That was the only use of chroot that I remember from the early days.”

**[LOTTERY1]:** David Petrou and John Milford. Proportional-Share Scheduling: Implementation and Evaluation in a Widely-Deployed Operating System, December 1997. [http://www.cs.cmu.edu/~dpetrou/papers/freebsd\\_lottery\\_writeup98.ps](http://www.cs.cmu.edu/~dpetrou/papers/freebsd_lottery_writeup98.ps)  
[http://www.cs.cmu.edu/~dpetrou/code/freebsd\\_lottery\\_code.tar.gz](http://www.cs.cmu.edu/~dpetrou/code/freebsd_lottery_code.tar.gz)

**[LOTTERY2]:** Carl A. Waldspurger and William E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Management, Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI '94), pages 1-11, Monterey, California, November 1994. <http://www.research.digital.com/SRC/personal/caw/papers.html>

**[POSIX1e]:** Draft Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment: Protection, Audit and Control Interfaces [C Language] IEEE Std 1003.1e Draft 17 Editor Casey Schauer

**[ROOT]:** Süper kullanıcı için kullanılan isimdir. Örneğin; Zilog, *root* yerine *zeus* ismini kullanmaktadır.

**[UAS]:** UAS, MVS sistemlerde RACF yapılandırmalarını denetlemek ve bunların bakımını yürütmek için uygun bir üründür. <http://www.entactinfo.com/products/uas/>

**[UF]:** Iliad 'ın kullanıcı dostu çizgi filminden alıntıdır.  
<http://www.userfriendly.org/cartoons/archives/98nov/19981111.html>