KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

JAVA PROGRAMLAMA ÇEVRELERİNDE SÜREÇ GÖZETLEME VE BİR SALDIRI TESPİT SİSTEMİNİN GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Mehmet Emin TENEKECİ

HAZİRAN 2008
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

JAVA PROGRAMLAMA ÇEVRELERİNDE SÜREÇ GÖZETLEME VE BİR SALDIRI TESPİT SİSTEMİNİN GERÇEKLENMESİ

Bilgisayar Mühendisi Mehmet Emin TENEKECİ

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsünce
"Bilgisayar Yüksek Mühendisi"
Unvanı Verilmesi İçin Kabul Edilen Tezdir.

Tezin Enstitüye Verildiği Tarih : 05.06.2008
Tezin Savunma Tarihi : 30.06.2008

Tez Danışmanı : Yrd. Doç. Dr. Hüseyin PEHLİVAN
Jüri Üyesi : Prof. Dr. Vasif V. NABİYEV
Jüri Üyesi : Yrd. Doç. Dr. Ali GANGAL

Enstitü Müdürü V. : Doç. Dr. Salih TERZİOĞLU

Trabzon 2008

ÖNSÖZ

Bilgisayar sistemlerinde, kullanıcıların davranışlarının belirlenmesi, saldırı tespit sistemleri, programların hatalarının ayıklanması için test edilmesi gibi birçok çalışmada sistem çağrılarının gözetlenmesi kullanılmaktadır. Bu maksatlar için sistem çağrılarını gözetleyebilecek birçok uygulama gerçekleştirilmiştir. Ancak bu uygulamaların tamamı işletim sistemi ile etkileşim yapabilecek C/C++ gibi dillerle geliştirilmiştir. Bu çalışmamızda sistem çağrılarının gözetlenmesini Java ortamında gerçekleştirebilmeyi sağlayacak bir çatı oluşturulmuştur. Bu çatı Java dili ile C/C++ dilleri arasında iletişimi sağlayacak JNI teknolojisinden yararlanılarak geliştirilmiştir.

Bu çalışmamda danışmanlığımı üstlenen değerli hocam Yrd. Doç. Dr. Hüseyin Pehlivan' a ilgi, alaka ve yardımlarından dolayı teşekkürü bir borç bilirim. Ayrıca her türlü destekleriyle her zaman yanımda olan aileme teşekkür ederim.

Mehmet Emin TENEKECİ
Trabzon 2008

İÇİNDEKİLER

Sayfa No

ÖZET

UNIX ve türevi işletim sistemlerinde bulunan süreç izleme mekanizmaları sistem saldırılarının tespiti ve programların davranışının modellenmesi gibi çok çeşitli amaçlar için yaygın olarak kullanılırlar. Birçok bilimsel çalışmada süreç eylemlerinin belirlenmesi izleme mekanizmaları yardımıyla yakalanan sistem çağrıları üzerine dayandırılır. Gerçekten sistem çağrıları dizisi ve onların verileri analiz edilerek izlenen sürecin gerçekleştireceği eylemlerin ortaya çıkarılması mümkündür. Süreçlerin yönetimi doğrudan işletim sistemleri tarafından yapıldığı için bir süreç izleme faaliyeti işletim sistemi ile işbirliğine ihtiyaç duyar. Bununla birlikte, günümüz UNIX sistemleri bu işbirliğinin sadece C/C++ programlama çevrelerinden yapılmasını desteklemektedirler.

Bu çalışmanın amacı süreç izleme işleminin Java programlama çevresi ile gerçekleştirilebilmesidir. UNIX işletim sistemlerinde süreç izleme gereksinimleri için kullanılan /proc dosya sisteminin Java programlama çevresine entegrasyonunu sağlayan bir çatı geliştirilmiştir. Bir sürecin izlenebilmesi, işletim sistemiyle etkileşimini, yani yerel sistem fonksiyonlarının kullanımını kapsadığı için süreç izleyen programların yazımı ileri düzeyde sistem programlama deneyimi gerektirmektedir. Diğer yandan, modern UNIX sistemlerinde süreç izleme gereksinimleri sadece C/C++ programlama çevrelerinden karşılanabilmektedir. Geliştirilen çatıda Java tarafından tedarik edilen JNI yardımıyla C/C++ çevrelerine erişim sağlanarak süreç izleme mekanizmasının kolay bir kullanımı gerçekleştirilmiştir. Bunun sonucu olarak, her bir programcı kendi ihtiyacına göre UNIX sistem süreçlerinin farklı denetimlerini icra eden programları Java dilinde kodlayabilecektir.

Sistem çağrılarının kullanıldığı en önemli alanlardan birisi saldırı tespit sistemleridir. Geliştirilen çatıyı kullanarak basit bir saldırı tespit sistemi Java ortamında gerçekleştirilmiştir.

Anahtar Kelimeler: Süreç İzleme, Sistem Çağrıları, Dillerin Birlikte Kullanımı, Saldırı Tespit Sistemi

.

SUMMARY

Process Tracing with Java Programming Language and Develop an Intrusion Detection System with Java

Process tracing mechanisms which are provided by UNIX and its derivative operating systems are extensively used for many various purposes such as attack&intrusion detection and model behavior of programs. In many academic researches, the determination of process activities are based on system calls which can be catched through the tracing mechanisms. Indeed, it is possible that the activities the traced process is to carry out is determined by analyzing system calls sequences and their data. Since the processes execute under the control of the operating system, a process monitoring task requires collaboration with the operating system. However, modern UNIX operating systems support this collaboration only through C/C++ programming environments.

The aim of this study is to ensure that process tracing tasks can be accomplished through Java programming environment. A framework is developed, in order to integrate the /proc file system used for process tracing requirements in UNIX operating systems into Java programming environment. The task of process tracing involves the experience of advanced system programming because it needs an interaction with the operating system, that is, the usage of the local system primitives. On the other hand, in modern UNIX systems, the process tracing requirements are provided only through C/C++ environments. In the developed framework, an easy use of the process tracing mechanism is made by accessing C/C++ environments via the JNI provided by Java language. Thus, each programmer will be able to code programs in Java language, which evaluate the activities of UNIX processes from different perspectives.

One of the most important study fields in which system calls are traced and evaluated is intrusion detection systems. Using the developed framework, a simple intrusion detection system is implemented in Java environment.

Key Words: Process Tracing, System call, Language Interoperability, Attack and Intrusion Detection System.

ŞEKİLLER DİZİNİ

SEMBOLLER DİZİNİ

API	Uygulanabilir Programlama Arayüzü (Application Programming Interface)
BSM	Temel Güvenlik Modülü (Basic Security Modul)
DARPA	Defense Advanced Research Projects Agency
JNI	Java Yerel Arayüzü (Java Native Interface)
JVM	Java Sanal Makinesi (Java Virtual Machine)
JUSİ	Java' dan UNIX Süreçlerini İzleme
PID	Süreç ID (Process ID)
tf-idf	Terim Sıklığı – Ters Döküman Sıklığı (Term Frequency-Inverse Document Frequency)

TABLOLAR DİZİNİ

Sayfa No

1. GENEL BİLGİLER

1.1. Giriş

Günümüzde sistem güvenliği konusunda yapılan çalışmalarda, sistem çağrılarının izlenmesi, kaydedilmesi ve analiz edilmesi çok değişik amaçlar için kullanılmaktadır. Özellikle sistemde çalışan programların davranışlarının belirlenmesi konusunda birçok önemli çalışma mevcuttur. Programcı açısından yazılan programdaki mantıksal hataların tespit edilebilmesi için program davranışının belirlenmesine ihtiyaç duyulur. Ayrıca, virüsler gibi bir sisteme zarar verebilecek saldırıların tespitinde de davranış belirleme konusu önemli bir rol oynar. Bu konu sistem güvenliği açısından yerel saldırı tespit sistemlerinin temelini oluşturmaktadır.

Sistem çağrılarının izlenmesi, işletim sisteminin çekirdek veri yapılarında değişiklik yapılmasına ve bazı sisteme özel verilerin okunmasına dayandığı için ileri seviyede sistem programlama deneyimi gerektirmektedir. Ayrıca bu işlemler UNIX işletim sisteminde sadece C/C++ dilleri ile yapılabilmektedir.

Bu çalışmamızda C/C++ ile yapılabilen süreçlerin izlenip, yaptığı sistem çağrılarının kaydedilmesi işlemini daha basit bir şekilde Java programlama dili ile gerçekleştirilebilecek bir programlama çatısı oluşturulmuştur. Bu amaçla Java dili ile C/C++ dilleri beraber kullanılarak UNIX işletim sisteminin çekirdek veri yapılarına erişimi mümkün hale getirilmiştir. Bu iki dilin etkileşimi sağlamak için JNI teknolojisinden yararlanılmıştır. Çalışmanın son kısmında oluşturulan çatı kullanılarak basit bir saldırı tespit sistemi geliştirilmiştir. Bu sistem, metin sınıflandırma algoritması ile süreçlerin yaptığı sistem çağrılarına göre bu süreçlerin normal veya anormal olarak çalıştığını belirlemektedir. Oluşturulan çatı bu saldırı tespit sisteminin işletim sisteminden yapılan sistem çağrılarını alabilmesi için kullanılmıştır.

1.2. Süreç İzleme

1.2.1. Süreç Kavramı

Süreçler sistem çekirdeği tarafından oluşturulan ve programların icradaki örneklerinin yerleştirildiği ortamlardır. Her süreç tek bir programı koşar ve bir program tarafından

birden fazla süreç oluşturulabilir.UNIX sistemlerinde programlar normal dosyalarda saklanan emirler ve veriler topluluğundan oluşurlar. Bunlar "çalışabilir" özellikleri setlenmiş dosyalardır. Programlar bir derleyici yardımıyla da oluşturulabilir.

Kullanıcı tarafından girilen bir program, sistemde onun için oluşturulacak bir sürecin içeriğine alınır ve o içerikte çalışmaya başlar. Süreç içeriği, emir segmenti (alanı), kullanıcı veri alanı, ve sistem veri alanı olmak üzere 3 parçaya ayrılır. Program içerikleri, sürecin emir ve kullanıcı veri segmentlerinin başlatımını yapabilmek için kullanılırlar. Süreç başlatıldıktan sonra programın kendisi ile bir bağlantısı kalmaz [1].

Süreç çekirdek tarafından, o an çalışmakta olan süreç adına yaratılır. Çalışmakta olan süreç ana süreç, oluşturulan süreç çocuk süreç olur. Çocuk süreç ana sürecin birçok sistem veri özelliklerini (attributes) miras alır. Örneğin, eğer ana süreç herhangi bir açık dosyaya sahipse, *çocuk* süreç için de o dosyalar açık olacaktır.

Süreçlerin her biri sistemde bir PID (Süreç ID) değerine sahiptir. PID değerleri her süreç için farklı, 0' dan büyük tamsayı değerlerdir. İşletim sisteminde çalışan süreçler, bu PID' ler dahil olmak üzere bir çok özellikleri ile birlikte süreç tablosu adı verilen özel bir veri yapısının elemanlarıdırlar. Süreçle ilgili birçok işlemde (süreç gözetleme, süreçler arası haberleşme vb..) PID değerleri önem kazanmaktadır.

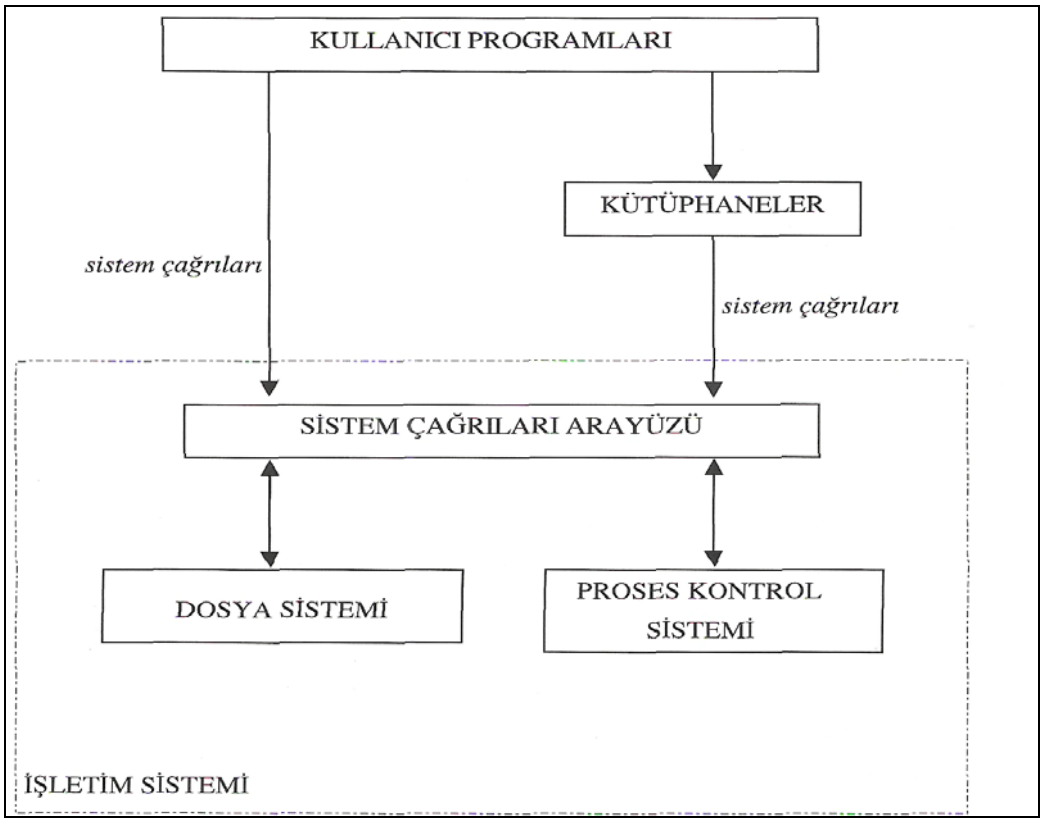
UNIX sistemlerinde her sürecin oluşturulduğu andan itibaren sahip olduğu 3 özel dosya tanımlayıcısı vardır: 0, 1 ve 2 numaralı tanımlayıcılar. 0, standart giriş (stdin) anlamına gelir. Süreçler girilen argümanları standart girişten alırlar. 1 ve 2 numaralı tanımlayıcılar sırasıyla standart çıkış (stdout) ve standart hata (stderr) tanımlayıcılarıdır.

Standart çıkış sürecin ürettiği çıktının yönlendirildiği ve genellikle terminal penceresi olarak belirlenen sistem bileşenidir. Standart hata ise süreçlerin ürettikleri hata mesajlarının gönderildiği ve standart çıkış gibi terminal penceresine bağlanan bir bileşendir.

Süreçler işletim sistemlerinden sistem çağrıları ile istekte bulunurlar. Sürecin sistem kaynakları üzerinde yapmak istediği işlemler, sistem çekirdeğine iletilen sistem çağrılarına dönüştürülür. Dolayısıyla bir sürecin sistemle etkileşimi, sistem çağrılarının bir dizisi olarak da düşünülebilir. Kullanıcı süreçlerinin yaptığı çağrılarla işletim sistemi çekirdeğinin kullandığı çağrılar birbirlerinden farklıdır. Çekirdek kullanıcı süreçlerinin yaptığı her bir sistem çağrısına karşılık özel bir yordamı çalıştırmaktadır.

Sistem çağrıları doğrudan program içinden yapılabileceği gibi, kütüphane fonksiyonları tarafından da oluşturulabilirler [2].

Şekil 1.1'de programların işletim sistemi çekirdeği ile ilişkisi gösterilmiştir.



Şekil 1.1. Program-işletim sistemi ilişkisi

1.2.2. Kullanıcı Modu ve Çekirdek Modu

UNIX' te süreçlerin kullanıcı ve çekirdek modu olmak üzere iki çalışma modu vardır [3].

Bir sürecin sistem çağrısı yapmasıyla veya bir kesme geldiğinde modlar arası geçiş yapılır (kullanıcı moddan → çekirdek moda). Kullanıcı moddaki süreçler kendi emir ve veri alanlarına erişebilirler. Çekirdeğin veya başka süreçlerin veri alanlarına erişemezler. Ayrıca bazı ayrıcalıklı emirlerin kullanıcı modda çalışmaları hata üretir.

Çekirdek moddaki süreçler çekirdek veri alanlarına erişebilirler. Sürecin çekirdek modda koşması, çekirdeğin o süreç adına istenen işi yapıyor olması gibi düşünülebilir. Kaynaklara çekirdek erişir, çeşitli işlemleri yapar, süreç tekrar kullanıcı modda çalışmaya bırakılır. Daha sonra süreç çeşitli sistem çağrılarıyla tekrar çekirdek moda geçmeyi isteyebilir.

İki ayrı mod kaynaklara erişim konusunda bir karışıklık çıkmasını engellemektedir. Bir süreç her zaman kaynaklara erişim isteğinde bulunmayacağı için farklı bir modda beklemesi uygundur. Kaynaklara erişimi kontrol etmek, işletim sistemlerinin temel görevlerindendir.

1.2.3. İçerik Değişimi

Sürecin emir alanı ile tanımlanan durumu, genel kullanıcı değişkenlerinin değeri, kullandığı makine kaydedicilerinin değerleri, kullanıcı ve çekirdek yığınlarının içeriği, süreç tablosundaki alanları sürecin içeriğini oluşturur. Bir süreci çalıştırmak onun içeriğini çalıştırmak anlamına gelir. Çekirdek, başka bir süreci çalıştırması gerektiğine karar verirse içerik değişimi yapar. Bu sayede sistem başka bir sürecin içeriğini çalıştırır. Çekirdek, sadece bazı özel durumlarda içerik değişimine izin verir. İçerik değişimi yapılırken, önceki sürecin içeriğine geri dönebilmek ve çalışmasını devam ettirebilmek için yeterli miktarda bilgi tutulur. Benzer şekilde kullanıcı moddan çekirdek moda geçerken de yeterli miktarda bilgi tutulur. Kullanıcı moddan, çekirdek moda geçiş bir mod değişimidir, içerik değişimi değildir [3]. Çekirdek, içeriği bir süreçten diğer sürece geçerken değiştirir. Mod değişiminde ise çekirdek yine aynı sürecin içeriğini çalıştırır durumdadır.

Çekirdek, bir süreç kendini uykuya aldığında, öldüğünde, sürecin kendine ayrılan koşum süresi dolduğunda içerik değişimi yapar. Çekirdek, kesmelere, kesmenin sebebi o süreç olmasa bile, durdurulan sürecin içeriğinde cevap verir. Yarıda kesilen süreç kullanıcı veya çekirdek modda çalışıyor olabilir. Çekirdek, süreci devam ettirebilmek için yeterli bilgiyi tutar ve kesmeye çekirdek modda hizmet verir. Kesmeler için ayrıca bir süreç üretilmez veya tarifelenmez.

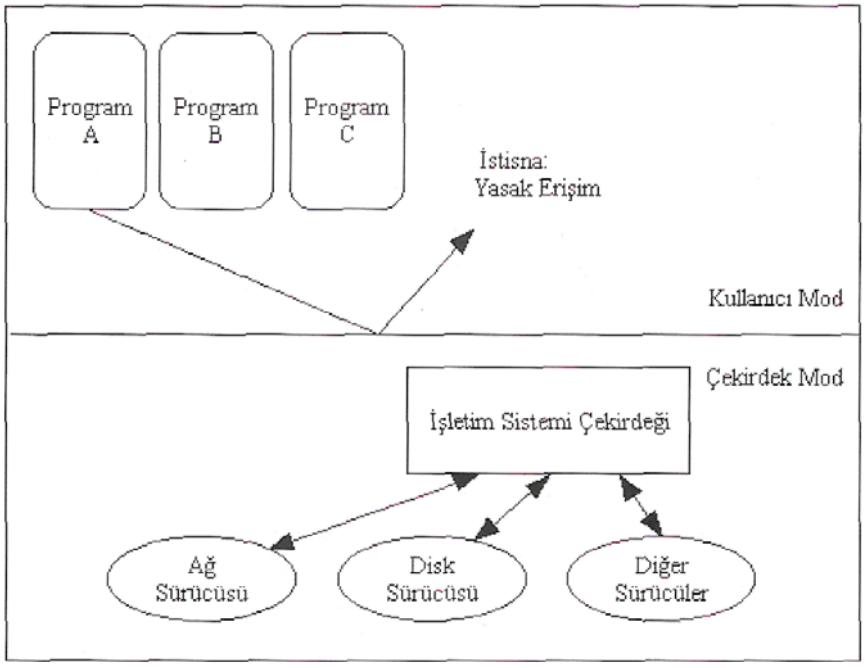
Çekirdeğin içerik değişim işlemini gerçekleştirme adımları aşağıda verilmiştir.

1. İçerik değişimi yapılıp yapılmayacağına ve buna izin olup olmadığına karar ver.
2. Önceki sürecin içeriğini sakla.
3. Tarifeleme algoritmasına bakarak çalışması gereken süreci seç.
4. sürecin içeriğini yükle.

1.2.4. Sistem Çağrıları

İşletim sisteminin çekirdeğini hatalı veya kötü niyetli programlardan gelebilecek zararlardan korumak amacıyla, modern işletim sistem mimarileri, kullanıcılar tarafından

çalıştırılan kod ile işletim sisteminin kendisinin çalıştırdığı kodu birbirinden ayırırlar. Bunu sağlamak için modern işlemciler sürecin hangi modda (kullanıcı veya çekirdek modu) çalıştığını belirlemek için bir mod biti bulundururlar. Bu bit setlenmiş ise süreç kullanıcı modda demektir ve işlemci donanımı çekirdeğin bellek alanına yapılan tüm erişimleri engeller. Şekil 1.2'de gösterildiği gibi eğer bir kullanıcı programı çekirdeğin bellek alanına erişmeye çalışırsa işlemci, yasak erişim istisnası üretir. Dolayısıyla kullanıcı modda çalışan hiçbir süreç çekirdek belleğine doğrudan erişim yapamaz.[4]



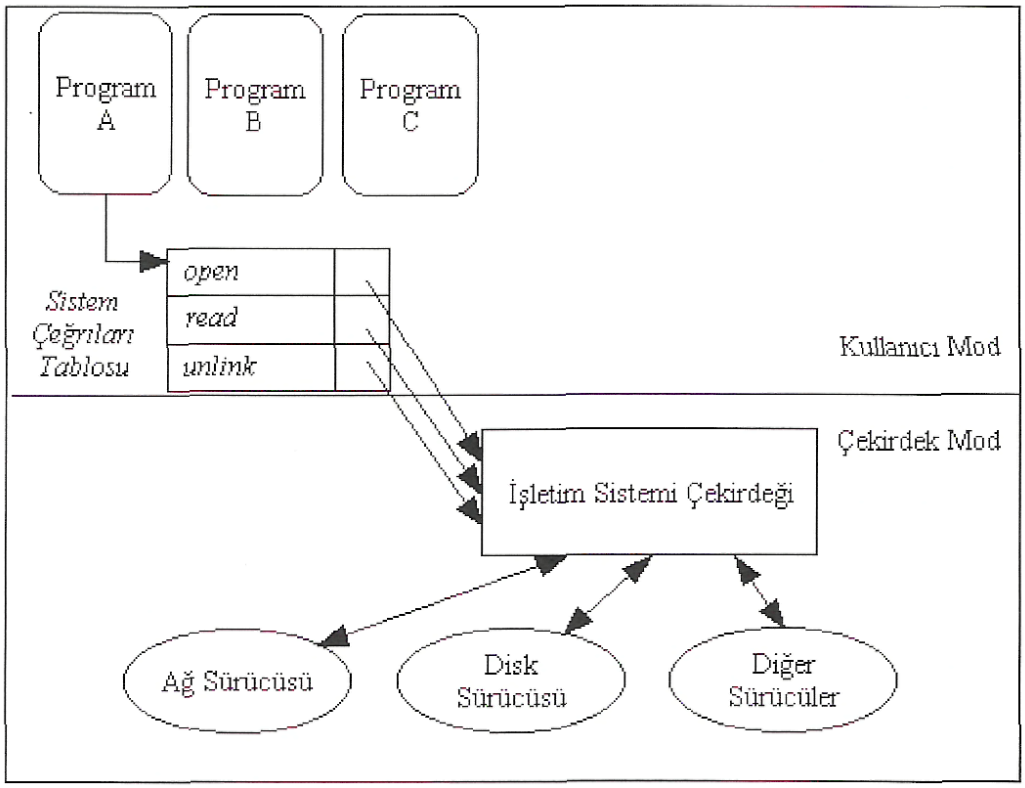
Şekil 1.2. Kullanıcı moddaki süreçlerin çekirdek alanlarına erişmeye çalışması

Ağ sürücüleri, disk sürücüleri ve paylaşılmış bellek gibi birimlere erişmek çekirdek moda ait eylemlerdir. Kullanıcı moddaki süreçler bu eylemleri gerçekleştirebilmek için sistem çağrılarını kullanırlar. Örneğin, kullanıcı seviyesindeki bir süreç diske doğrudan erişemez, çekirdekten bunu kendisi için yapmasını sistem çağrıları vasıtasıyla ister. Sistem çağrıları arayüzü kullanıcı mod ve çekirdek mod arasında izin verilen tek arayüzdür. İşletim sistemi tarafından gerçekleştirilen tüm hizmetlere sadece sistem çağrıları yoluyla erişilebilir. Sistem çağrıları işletim sisteminde sistem çağrıları tablosu kullanılarak gerçeklenir. Sistem çağrıları tablosu her bir sistem çağrısını işletim sistemi çekirdeği içinde belli fonksiyon adresi ile ilişkilendirir.

Tablo1.1. Sistem çağrıları ve bellek adresleri

Sistem Çağrısı	**Çekirdek Fonksiyon Adresi**
Open	0x00009CA1
Read	0x00..
Unlink	0x00..

Şekil 1.3' de gösterildiği gibi bir sistem çağrısına rastlandığında, örneğin *open,* sistem çağrıları tablosundan *open'*ı bulur ve 0x00009CAl adresindeki çekirdek mod fonksiyonuna kontrolü devreder. Tablo 1.1' de gösterildiği gibi tüm sistem çağrılarının sistem çağrıları tablosunda işletim sistemi çekirdeğinde uygun fonksiyona karşılık düşürülen bir girişi (entry) vardır.



Şekil 1.3. Sistem çağrılarının çalışması

Sistem çağrıları görevlerine göre 4 sınıfa ayrılabilir. Bunlar süreç yönetimi sistem çağrılan, bellek yönetimi sistem çağrıları, dosya sistemi çağrıları ve diğer sistem çağrılarıdır. Tablo 1.2' de bu sınıflara giren sistem çağrılarından bazıları gösterilmiştir.

Tablo1.2. Sistem çağrılarının sınıflandırılması

Süreç Yönetimi	Bellek Yönetimi	Dosya Sistemi	Diğer
exec, fork, vfork, pipe, dup, dup2, exit, signal, ptrace, setuid, kill, poll	mmap, brk, swaporı, swapoff, mlock, sync, mprotect,	open, creat, read, write, link, symlink, unlink, chdir, rmdir, chmod, rename, chown, mknod, mount, mkdir, truncatc, flock, close, lseek, fcntl, readdir, stat	ioctl,time,sync, plock

1.2.5. Süreç Gözetleme Mekanizmaları

Süreç gözetleme mekanizmaları, bir sürecin diğer süreçlerin çalışmasını izlemesine izin verir. Gözetlenen süreç adım adım koşturulabilir ve bellek alanı okunup değiştirilebilir. Veri ayrıca süreç tablosundan da okunabilir. gdb ve dbx gibi hata ayıklayıcı *(debugger)* programların eski versiyonları *ptrace* [3] sistem çağrışma dayandırılmıştı. Bu programların yeni versiyonları ile ps ve top gibi programlar ise koşmakta olan süreçleri incelemek ve kontrol etmek için */proc* [5] dosya sistemini kullanırlar.

1.2.5.1. ptrace ile Gözetleme

ptrace sistem çağrısı çekirdekten süreç hakkında bilgi edinmek için kullanılır. Bir ayıklayıcı(debugger) süreç gözetlenecek bir süreç üretir ve *ptrace* sistem çağrısıyla durma noktalarını (breakpoints) setleyip resetleyerek ve sürecin adres alanına yazarak veya orayı okuyarak çalışmasını kontrol eder. Süreç gözetleme işlemi ayıklayıcı ile gözetlenen sürecin senkronizasyonundan ve gözetlenen sürecin çalışmasının kontrolünden oluşur.

ptrace sistem çağrısının dizilimi aşağıdaki gibidir.

ptrace (cmd, pid, adres, veri);

cmd - veri okuma-yazma, süreci çalışmaya devam ettirme gibi çeşitli komutlar olabilir.

pid - gözetlenecek sürecin süreç id'si.

adres- çocuk süreçteki yazılıp okunacak sanal adres

veri- yazılacak tamsayı değer

Çekirdek *ptrace* sistem çağrısını koşarken ayıklayıcı sürecin gözetlenmekte olan bir çocuk süreci olduğunu doğrular ve iki süreç arasında veri transferini gerçekleştirmek için genel *trace* veri yapısı kullanır. Diğer gözetleme yapan süreçlerin *trace* veri yapısını bozmamaları için bu yapıyı kilitler. *cmd* , *adres* ve *veri* 'yi *trace* veri yapısına kopyalar, çocuk süreci uyandırır, koşmaya hazır duruma getirir, çocuk süreçten cevap gelinceye kadar uyuma moduna geçer. Çocuk, çalışmaya devam edince (çekirdek modda) uygun *trace* komutunu koşar, cevabını *trace* veri yapısına yazar ve ayıklayıcıyı uyandırır. Komutun türüne göre çocuk süreç tekrar gözetlenme durumuna geçip yeni komut bekleyebilir veya sinyallerin kontrol edilmesi işleminden dönüp çalışmasını sürdürebilir. Ayıklayıcı çalışmasına devam ettiğinde çekirdek takibe alınan süreç tarafından sağlanan dönüş değerini kaydeder, *trace* veri yapısını serbest bırakır, kullanıcıya döner.

1.2.5.2. */proc* Dosya Sistemi

/proc dosya sistemi, sistemdeki her sürecin görüntüsüne erişim sağlar. Sistemde yaratılan tüm süreçler */proc* dizini altında temsili dosyalar şeklinde tutulurlar. */proc* dizinindeki her bir dosya, süreç ID' ye denk düşen sayılardır ve her bir dosyanın sahibi sürecin gerçek kullanıcı ID' si ile belirlenir. Dosyayı açma izinleri genel dosya sistemi izinlerinden daha kısıtlıdır.

Standart sistem çağrısı arayüzü */proc* dosyalarına erişim için kullanılabilir. *read* ve *write* kullanılarak, *Iseek* ile erişilen süreç adres alanına veri yazılıp, oradan okuma yapılabilir.

İlgilenilecek olaylar */proc* arayüzüne bayrak kümeleri kullanılarak belirtilir. Gözetlenen süreç ilgilenilen (sisteme daha önce bildirilen) olaylardan birine rastlayınca durur. Sistem çağrısı girişindeki bir durma, sistem çağrısından gerekli argümanlar alınmadan önce gerçekleşir. Sistem çağrısı çıkışındaki durma ise sistem tüm geri dönüş

değerlerini, gözetlenen sürecin veri alanına yazdıktan sonra gerçekleşir. Bu durma noktaları, sistem çağrısı argümanlarını ve geri dönüş değerlerini elde etmeye fırsat verir.

Kontrol edilen sürecin yarattığı yeni süreçleri kontrol altına almak için ayıklayıcı, orijinal sürecin *inherit-on-fork* bayrağını setleyebilir ve *fork, vfork* çağrılarının dönüş değerlerini gözetleyebilir. Kontrol edilen süreç *fork* yapınca, çocuk süreç ana sürecin tüm gözetleme bayraklarını miras alır ve iki süreç de (ana ve çocuk) *fork* çağrısının dönüşünde durur. Ayıklayıcı program ana sürecin durduğunu görür, çocuk süreçten dönen değeri (çocuk sürecin süreç ID'si) alır ve bu değeri çocuğun */proc* dosyasını açmak için kullanır. Kullanıcı seviyesinde herhangi bir komut koşulmadan çocuk süreç durduğu için ayıklayıcı program tam kontrolü sağlayabilmektedir.

1.2.5.3. *ptrace* ve */proc'un* Karşılaştırması

Gözetleme mekanizmaları, bir programın sistem servislerini kullanımını kolay bir şekilde takip edebilmemizi sağlar. *ptrace* mekanizması */proc'a* göre daha ilkel bir mekanizmadır. Aşağıda */proc*'un *ptrace*' e göre üstünlükleri verilmiştir.

1. Süreç gözetleme daha hızlıdır ve performansı etkilemez,
2. Süreçler eş zamanlı olarak gözetlenebilir,
3. Çalışır durumda olan süreçleri gözetlemek mümkündür,
4. Üzerinde durulan sistem çağrıları bireysel olarak gözetlenebilir [3].
5. Verilen bu özellikleriyle /proc mekanizması süreç gözetlemenin en etkin yoludur.

1.2.6. Süreç İzleme Mekanizmasının Kullanıldığı Uygulamalar

Süreç izleme mekanizmaları çeşitli bilimsel çalışmaların ihtiyaç duyduğu süreçlere ait durum bilgilerinin temin edilmesinde bir araç olarak görev yaparlar.

Süreçlerin davranışlarını sergiledikleri çalışma ortamları işletim sistemi ile süreçler arasında yer alan ve yazılımla inşa edilmiş çeşitli arayüzler tarafından oluşturulabilirler [6]. Kullanıcıların çalıştırdığı UNIX süreçleri genellikle komut yorumlayıcıların (shell) kontrol ettiği çevreler içerisinden aktivitelerini yürütürler. Bu aktivitelere örnek olarak aktif dizinin değiştirilmesi, dosya sistemi üzerinde yapılan okuma/yazma gibi işlemler, yeni süreç oluşturma ve süreçler arası haberleşme verilebilir. Bütün aktiviteler işletim sistemi çekirdeğine yapılan sistem çağrıları ile çekirdek seviyesinde (modunda) gerçekleştirilir.

Dolayısıyla sistem çağrılarını izleyerek süreçlerin çalışma ortamları üzerinde yürüttükleri aktiviteleri belirlemek mümkündür.

Literatürde sistem çağrılarının izlenmesi, kaydedilmesi ve de analiz edilmesi çok değişik amaçlar için kullanılmıştır. Bu amaçlardan bazıları aşağıda özetlenmiştir.

1. Hata Ayıklama: Bir programın yaptığı sistem çağrıları incelenerek programın davranışı hakkında bilgi edinilebilmektedir. Bunun sonucunda ilgili programın davranışında gözlemlenen anormallikler yardımıyla kod içinde hatalı alan tespit edilebilir ve gerekli düzeltmeler yapılabilir [7,8].
2. Sistem Kötü Kullanımlarının ve Yetkisiz Girişlerin Algılanması: Sistemi kullanan kimselerin yaptıkları işlemler kaydedilerek bu kişilerin sistem kaynaklarını nasıl kullandıkları hakkında bilgi sahibi olunabilmektedir. Sistemde yapılan işlemler sonucu meydana gelen sistem çağrıları dizileri üzerinde sınıflandırma, tanıma algoritmaları ve veri madenciliği yöntemlerinden yararlanarak kişilerin sistemi kullanım amaçları yada niyetleri belirlenebilir. [9-13].
3. Program Doğrulama: Programların sistem üzerinde gerçekleştirdiği işlemler kaydedilerek bir programın beklenen davranışı gösterip göstermediği belirlenebilir. Ayrıca programın davranışına göre bir virüsün yada casus yazılımın (spyware) varlığı tespit edilebilir [14-16].
4. Ortam Modelleme ve Yeniden Oluşturma: Programların derlenmesi esnasında yapılan sistem çağrılarının izlenmesi ile programların hangi ortamda hangi kütüphane dosyalarını kullandığı belirlenebilir [17, 18].
5. Arayüzlerin Anlaşılması: Programların icrası esnasında yaptığı sistem çağrıları incelenerek bir program arayüzünün kullanım biçimi ve davranışı belirlenebilir. Ayrıca belgelenmemiş sistem fonksiyonlarının veya çağrılarının nasıl çalıştığı hakkında bilgi alınabilir [19].
6. Güvenli Dosya Sistemi İşlemleri: Programların dosya sistemi üzerinde yaptığı sistem çağrıları izlenerek, dosya sistemini etkileyen aktiviteler ile karşılaşıldığında dosyaların birer kopyaları alınabilir. Bu şekilde virüsler veya kullanıcı hatalarından meydana gelen hasarlar telafi edilebilir [20-21].

1.3. Programlama Dillerinin Birlikte Kullanılması

1.3.1. Dillerarası İhtiyaçlar

Günümüzde, bütün ihtiyaçlarımızı eksiksiz bir şekilde sadece bir programlama dilini kullanarak gerçekleştirmemiz mümkün değildir. Bu nedenden dolayı bir problemin çözümünde birkaç programlama dilinin kullanılması gerekmektedir. Dillerin birlikte kullanılması işlemi bir dilin etkin özelliklerinin diğer dil tarafından miras alınması veya ihtiyaç anında çeşitli dil mekanizmaları yardımıyla ihtiyaç duyulan işlevlerin çağrılması olarak gerçekleşir. Örneğin, düşük seviyeli (bit seviyesinde) işlemlerin gerçekleştirilmesi veya işletim sistemi kaynaklarına erişim için C veya C++ gibi orta seviyeli diller, veri tipi güvenliği, otomatik atık toplama gibi yüksek seviye işlemler için de Java gibi yüksek seviyeli diller kullanılabilir.

Programlama dillerinin birlikte kullanılabilirliği her ihtiyaca uygun bir programlama dili oluşturmadan daha fazla desteklenir. Bu şekilde daha önceden farklı bir programlama dilinde yazılmış olan programların yeniden kullanılması sağlanır.

Dillerin birbirlerini desteklemeleri bazı mekanizmaları diğerine kazandırmak için kod çağırma şeklinde gerçekleşir. Ancak bu işlem fazladan zaman ve bellek ihtiyacını beraberinde getirir. İletişim esnasında bir taraftan diğer tarafa gönderilen parametre ve geri dönüş değerlerinin paketlenmesi ve uygun veri tipi dönüşümlerinin yapılması gerekmektedir. Bu işlemler yukarıda bahsedildiği gibi zaman ve bellek harcayacağından sistem performansını etkilemektedir. Ancak yüksek hesaplama gerektiren işlemleri düşük seviyeli programlama dillerini kullanarak işlemci seviyesine getirip performans artışı sağlanabilir [22].

1.3.2. Birlikte Kullanım Teknolojileri

Son yıllarda programlama dillerinin ortak kullanımını sağlamak için birçok yeni teknoloji geliştirilmiştir. Bu teknolojiler yerel veya dağıtık olarak çalışabilmektedir. Bu şekilde diller arası iletişimi sağlarken, sistemler arası iletişimi de sağlamakta kullanılmış olurlar.

Bu teknoloji veya yapılara çok çeşitli örnekler verilebilir: CORBA [23] (Ortak Nesne İstem Aracı Mimarisi), RPC [24] (Uzak Yordam Çağırma), COM [25] (Parçacıklı Nesne Modeli), JNI (Java Yerel Arayüzü). Bu teknolojiler aşağıda özetlenmiştir:

CORBA (Ortak Nesne İstem Aracı Mimarisi): CORBA Nesne Yönetim Grubu'nun (OMG) Nesne Yönetim Mimarisi'nin (OMA) ana bileşenlerinden birisidir. Nesne Yönetim Mimarisi Nesne Modeli ve Referans Modelinden oluşur. Nesne Modeli heterojen bir ortamda dağılmış nesnelerin nasıl tanımlanabileceğini belirler. Referans Modeli ise nesneler arası etkileşimleri tanımlar. Dolayısıyla Nesne Yönetim Mimarisi heterojen ortamlara dağılmış beraber işleyebilen dağıtık nesnelerin geliştirilmesine ve konuşlandırılmasına yardımcı olur. CORBA sayesinde programcılar kullandıkları nesnelerin hangi dilde yazıldığına, dağıtık olup olmadıklarına, işletim sistemlerine ve iletişim protokollerine bakmaksızın programları geliştirebilirler.

RPC (Uzak Yordam Çağırma): RPC teknolojisi bir bilgisayar programının başka bir bilgisayar veya ağ üzerindeki adres alanında kod çalıştırmasını sağlar. Ancak uzaktaki sistemle olan etkileşimin detaylarından haberi yoktur.

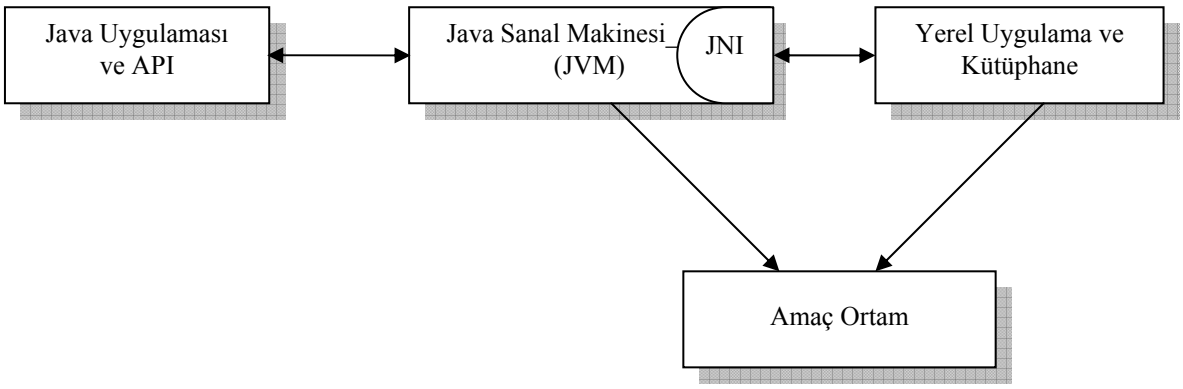
COM (Parçacıklı Nesne Modeli): Microsoft tarafından bilgisayarlar arasında dağıtılmış uygulamaların daha rahat yaratılabilmesi ve çalışabilmesi için yazılmıştır. Bu teknoloji programların parçacıklı olarak oluşturulmasına imkan verir. Bu şekilde oluşturulan parçacıklar değişik uygulamalar tarafından kullanılır. Bu teknoloji herhangi bir dilde süreçler arası iletişimi ve dinamik nesne oluşturulmasını sağlamak için kullanılır.

JNI (Java Yerel Arayüzü): Java programlarının başka dillerde yazılmış fonksiyonlarını kullanabilmesi veya başka dillerde yazılan programların Java ile yazılmış fonksiyonları kullanabilmesi için yazılmış bir programlama çatısıdır. Burada çalıştırılan kod JVM tarafından yorumlanmaktadır.

Bu teknolojiler dillerin yetersiz kaldığı noktalarda başka programlama dillerinden yararlanmaya ilaveten kullanılan dilden bağımsız olarak programlama yapma amacına yönelik de gelişmektedir. Bu şekilde birçok değişik dil kullanılarak ortak bir proje geliştirme imkanı vermiştir. Son yıllarda Microsoft'un çıkarmış olduğu .NET teknolojisi bu yapıyı destekleyecek altyapıda olup, birçok değişik programlama dili kullanılarak program geliştirebilmeyi sağlamıştır. .NET ortamında kodlama yapılan dillerden bağımsız bir ara dile dönüşüm yapılabilmektedir. Bu yol içinde değişik dillerde yazılmış kodlar birleştirilerek programcı gereksinimini karşılayan bir program kolaylıkla üretilebilir.

1.3.3. JNI Teknolojisinin Mimarisi ve Kullanıldığı Yerler

JNI, java ortamının en önemli özelliklerinden birisi olarak karşımıza çıkmaktadır. JNI yapısını kullanan programlar, C ve C++ ile yazılmış olan yerel uygulamalarla ve kütüphanelerle birlikte çalışabilme olanağına sahip olurlar. JNI, programcılara Java'nın avantajlarını kullanma imkanı sunarken, aynı zamanda önceden yazılıp derlenmiş kodlardan da vazgeçmek gibi bir durumu ortadan kaldırmaktadır.



Şekil 1.4. JNI' nın java ortamındaki rolü

JNI, Java sanal makinesinin (JVM) bir parçası olmakla beraber Java uygulaması ve yerel kütüphane ile iki yönlü iletişimi sağlamaktadır. Standart Java uygulamaları yazılarak Java byte koda dönüştürülür. Çalıştırılmak istenildiği zaman Java Sanal Makinesi tarafından çalıştırılan ortama uygun makine kodları üretilerek sistemde çalıştırılması sağlanır. Yerel fonksiyonları Java programımızda kullanmak istediğimiz zaman Şekil 1.4'de gösterildiği gibi JNI kullanılarak bu fonksiyon çağırılır ve JVM ile etkileşimli olarak yazılan Java programı ile beraber sistemde çalıştırılır.

JNI, Java uygulamalarından yerel kütüphanelerin işlevlerini çağırmak için yerel yöntemler tanımlamak için kullanılabilir. Buna ilaveten yerel uygulamalara Java sanal makinesini (JVM) gömebilmeyi sağlayan bir arayüzü destekler.

JNI, Java ile nispeten daha alçak seviyeli dillerin birlikte kullanılabilmesini sağlar. Bu sayede Java ile C/C++ dilerinin birlikte kullanılarak kodlama yapılabilmesi güçlü bir programlama çevresi oluşturacaktır. Çünkü, bu dillerin birbirlerine göre çeşitli üstünlükleri ya da zenginlikleri bulunmaktadır. Örneğin, C++ dili yüksek hesaplama gerektiren

uygulamalar ve bellek optimizasyonu için oldukça elverişli iken Java mümkün olduğunca taşınabilirliği destekler ve çok zengin sınıf kütüphanelerine sahiptir [22].

Bununla birlikte Java'nın temel olarak iki amaç için JNI kullanarak yerel fonksiyonlarına eriştiğini söyleyebiliriz. Bunlar;

Java uygulama geliştirme arabirimi (Application Programming Interface- API) bazı ortam-bağımlı özellikleri desteklemiyor olabilir. Bir uygulamada, Java API' nin desteklemediği bir işlemi gerçekleştirmek zorunda kaldığımızda JNI gerekebilir. Süreçler arası iletişimin (Interprocess Communication) getireceği performans kaybını engellemek için bir yerel kütüphanenin fonksiyonlarının kullanılması gerekebilir. Kütüphaneyi, çalışan programın adres uzayına yüklemek bellek ve zaman kazancı sağlayacaktır.

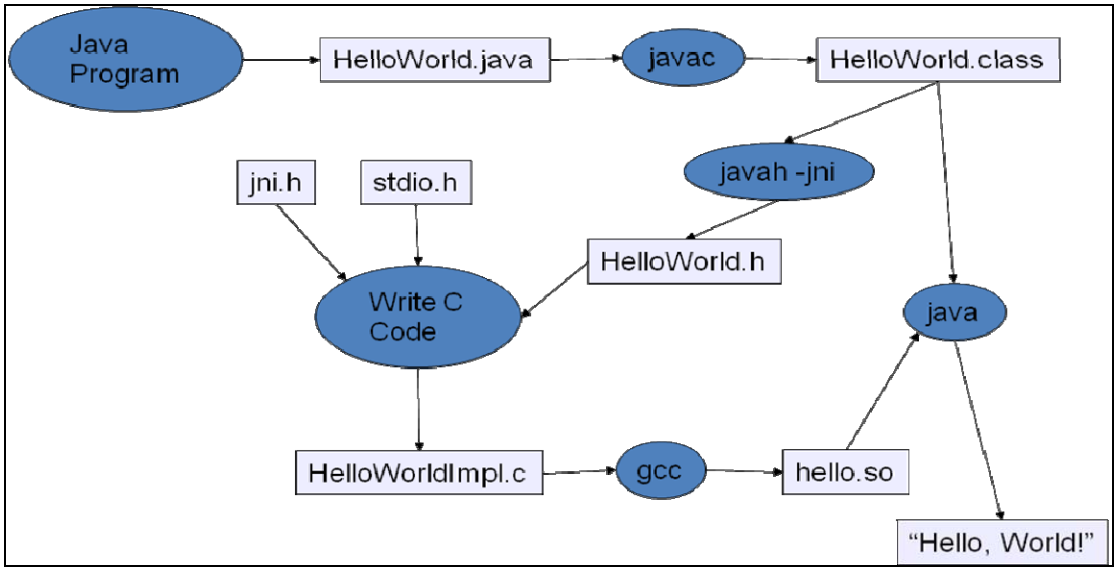
JNI kullanılarak geliştirilen birçok uygulama mevcuttur. Bu uygulamalara birkaç tane örnek verecek olursak: JTUX [26] (Java-To-UNIX Package), POSIX/SUS sistem çağrılarını Java ortamında kullanabilmek için JNI ile oluşturulan bir arayüzdür. Bu arayüz sayesinde Java paketleri ile desteklenmeyen UNIX fonksiyonlarına Java ortamından erişilebilmektedir. Unix sistem çağrılarının büyük bir çoğunluğuna Java ortamından erişimin sağlanması için gerekli fonksiyonlar tanımlanmıştır. Bu fonksiyonlar kullanılarak Java çevresinden Unix sisteminde kernel seviyesinde programlama imkanı sağlanmıştır. Diğer bir JNI uygulaması da sayısal işaret işlemede Java ile C/C++ fonksiyonlarının birlikte kullanılmasıdır. Burada C/C++ fonksiyonları yüksek performans elde etmek için tercih edilmiştir [22]. Bu uygulama ile C/C++ dilinin düşük seviyeli olarak direk işlemci üzerinde hızlı aritmetik işlem yapma yeteneğinden faydalanılmıştır. Bu şekilde çok fazla aritmetik işlem gerektiren sayısal işaret işleme uygulamalarının Java kullanılarak hızlı bir şekilde yapılması sağlanmıştır. Buna benzer bir çalışmada da Java çevresinden MPEG 4 sahneleri JNI kullanılarak oluşturulmuştur [28]. Burada biraz önce bahsettiğimiz gibi performans artışı çok ciddi bir şekilde görülmektedir. Diğer bir örnek olan JNIMarshall [27], Java ile C/C++ fonksiyonları arasında etkileşimi sağlayacak JNI fonksiyonlarını otomatik olarak üreten bir uygulamadır. Bu çalışmayla acemi programcıların üst düzey programlama gerektiren JNI ile yerel fonksiyonları kullanabilmeleri otomatikleştirilmiştir.

1.3.4. JNI Teknolojisinin Kullanımı ve Veri Tiplerinin Dönüşümü

Java uygulamalarında, JNI kullanılarak C fonksiyonlarının kullanımı birkaç aşama ile gerçekleştirilmektedir. *Native* fonksiyonları içeren java programı derlendikten sonra,

programda kullanılan C fonksiyonlarının prototiplerini içeren başlık dosyası *javah –jni* komutu ile üretilir. Bu başlık dosyasına uygun C kaynak dosyası gerçeklenerek, *gcc* programı ile Java programına eklenecek kütüphane dosyası oluşturulur. Java uygulaması bu kütüphane dosyası çalışması esnasında kullanır.

Burada bahsettiğimiz işlemlerin nasıl gerçekleştiği basit olarak Şekil 1.5' de gösterilmektedir.



Şekil 1.5. JNI kullanılan programların temel yapısı

Şimdi genel olarak bu adımları inceleyelim.

Adım 1: İçerisinde native metot deklarasyonu olan "HelloWorld.java" dosyası oluşturulur.

Adım 2: "HelloWorld.java" kaynak dosyası javac komutu aracılığı ile derlenerek "HelloWorld.class" dosyası oluşturulur.

Adım 3: Native metot gerçeklemesi için fonksiyon prototipini içeren C header dosyası (HelloWorld.h) elde etmek için javah –jni komutu kullanılır.

Adım 4: Native metodun gövdesini içeren C kodu (HelloWorld.c) yazılır.

Adım 5: Kullandığımız işletim sistemine göre C kodu derlenerek native kütüphane dosyası olan hello.so oluşturulur. Bu dosya sistemler arasında farklılık içerir.

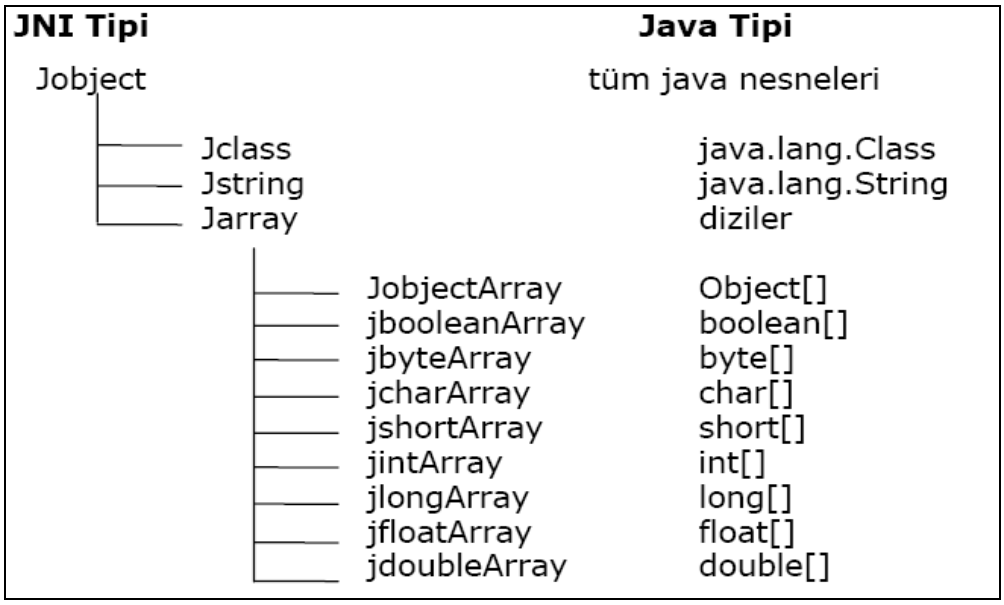
Adım 6: java komutu aracılığıyla HelloWorld programı çalıştırılır. Burada oluşturduğumuz HelloWorld.class dosyası ile native kütüphane (hello.so) çalışma zamanında yüklenir.

Yerel fonksiyonların kullanımı esnasında Java çevresinden gelen parametrelerin geldikleri biçimde kullanılmaları sistemler arası farklılıklardan dolayı mümkün olmamaktadır. Bu parametrelerin bazı dönüşümlerden geçmeleri gerekmektedir. Bununla ilgili olarak JNI'de, Java veri türlerinin C++ karşılığı olan veri türleri tanımlanmıştır. Temel veri türleri ve referans veri türleri farklı işlem görürler. Tablo 1.3'de veri türlerinin eşleştirilmesi ve jni.h dosyasında yapılan tanımı verilmiştir.

Tablo 1.3. C++' daki temel veritiplerinin JNI karşılıkları

Java Tipi	**JNI Tipi**	**Açıklama**
boolean	jboolean	işaretsiz 8-bit
byte	jbyte	işaretli 8-bit
char	jchar	işaretsiz 16-bit
short	jshort	işaretli 16-bit
int	jint	işaretli 32-bit
long	jlong	işaretli 64-bit
float	jfloat	32-bit kayan noktalı
double	jdouble	64-bit kayan noktalı

Tablo 1.3'de belirtilen şekilde gelen temel veri tipleri kullanılırken nesneleri açıkça belirtilmemiş bir referansla (*opaque references*) yerel metotlara geçirir. Bu referanslar, nesnenin Java sanal makinesi tarafından oluşturulan bellek alanına bir C++ pointer' idir. Ancak nesnenin yapısı programcıya tamamen kapalıdır. Bu nedenle nesne, JNIEnv pointeri aracılığı ile uygun JNI fonksiyonları kullanılarak ele alınmalıdır. Örnek olarak, java.lang.String türünün JNI karşılığı jstring'dir. jtring'in gerçek değerine yerel kodda doğrudan ulaşılamaz. Tüm JNI referansları, JNI'de *jobject* türünden miras almışlardır.



Şekil 1.6. Java – JNI Referans Veri Türü Eşleştirmesi

Şekil 1.6'da referans veri türlerinin eşleştirilmesi verilmiştir. JNI fonksiyonları, JVM veri türleri ve C++ veri türleri arasında iki taraflı dönüşümü sağlarlar. Temel veri türleri doğrudan kullanılabilir, ancak referans veri türlerinin kullanımı doğrudan yapılamaz. Bu işlem programcıya bırakılmıştır. Gerekli bellek bölgesi programcı tarafından oluşturulur ve gerekli veriler okunur veya yazılır. Referans veri tipi dönüşümlerinde dikkat edilmesi gereken nokta verilerin kullanılması için oluşturulan bellek bölgesinin kapatılmasının unutulmamasıdır.

Referans veri tiplerinden kullanılan veriye göre erişim farklılık göstermektedir. Bu nedenle farklı referans veri tiplerine aynı şekilde erişim mümkün olmamaktadır. Bunlara örnek olarak birkaç tanesinden bahsedelim.

Stringlere erişim uygun JNI fonksiyonunu kullanarak, jstring objelerini C++ karakter dizilerine dönüştürmesi gerekir. JNI, Unicode karakterlerden UTF-8 karakterlere iki yönlü dönüşümü de destekler. GetStringUTFChars fonksiyonu, Unicode karakter dizisini, UTF-8 karakter dizisine dönüştürür ve bu diziye bir pointer döndürür. Bu pointer, artık normal bir C++ karakter pointeri gibi kullanılabilir. Bu işlem aşağıda gösterilmiştir.

```
JNIEXPORT jstring JNICALL Java_Class_deneme(JNIEnv *env,jobject obj, jstring s)
{
char buf[128];
const jbyte *str;
```

```
str = (*env)->GetStringUTFChars(env, s, NULL);//UTF-8 karakterine dönüştürülür
if (str == NULL) {
return NULL;
}
printf("%s", str);
(*env)->ReleaseStringUTFChars(env, s, str);// bellek alanı serbest bırakılıyor
scanf("%s", buf); // Kullanıcının 127 karakterden fazla yazmadığını düşünelim.
return (*env)->NewStringUTF(env, buf); //Unicode dönüşümü yapılıyor.
}
```

JNI 'de temel veri tipi dizileri ve referans dizilerinin ele alınması farklı şekillerde gerçekleştirilmektedir. Temel veri tipi dizilerinin yönetimi, String veri türünde olduğu gibi bazı JNI fonksiyonlarının kullanımını gerektirir. Bir java kayan noktalı sayı dizi referansının JNI karşılığı jfloatArray veri tipidir. jstring'in bir C++ veri türü gibi kullanılamayacağı gibi, jfloatArray da bir C++ dizi veri türü gibi doğrudan kullanılamaz.

Native metotta temel veri tipi dizilere erişim için JNI fonksiyonların kullanımına gereksinim duyulur.

```
JNIEXPORT jint JNICALL Java_Int_dizi(JNIEnv *env, jobject obj, jintArray dizi1 )
{
    jint *buf;
    jint i, toplam=0;
    (*env)->GetIntArrayRegion(env,dizi1,0,10,buf); //dizinin dönüşümü
    for(i=0;i<10;i++) {
        toplam+=buf[i];
    }
    (*env)->ReleaseIntArrayElements(env, dizi1, buf, 0);//bellek serbest bırakılıyor.
    return (toplam);
}
```

JNI, Java nesnelerinin veri üyelerine erişimine olanak sağlar. Bir nesne olgusu jobject türünde bir C++ değişkeni olarak yerel metoda geçer. Metot içerisinde bu nesneye ait veri

üyelerinin değerinin alınması ya da bir değere setlenmesi uygun JNI fonksiyonları ile sağlanabilir. Bir Java sınıfı, temel ya da referans veri türlerinden oluşabilir.

Yerel yöntemin gerçekleştiriminde, bir sınıfın nesnesine ait veri üyelerine erişmek için atılacak ilk adım, nesneye ait sınıf tanımına erişmektir.

jclass cls = env->GetObjectClass(nesnem);//nesnenin sınıfı belirleniyor.

Daha sonra GetFieldID JNI fonksiyonu ile sınıf tanımını ve erişilmek istenen veri üyesinin adını ve türünü kullanarak, bu veri türüne ait bir field ID değeri elde edilir.

jFieldID fid = env->GetFieldID(cls,"s","Ljava/lang/String;");

Yukarıda verilen örnekte, GetFieldID fonksiyonuna parametre olarak erişilecek alanın türüyle ilgili "Ljava/lang/String;" değeri gönderilmiştir. Bu string değerleri, JNI alan tanımlayıcılarıdır. Tablo 1.4'de temel veri türlerinin ve nesne veri türlerinin alan tanımlayıcı değerleri verilmiştir.

Tablo 1.4. Sınıfların Elemanlarına Erişmek için Kullanılan Tanımlayıcılar

Veri Türü	**Alan Tanımlayıcısı**
byte	B
char	C
double	D
float	F
int	I
long	J
class tipi	Lclassadı;
short	S
void	V
boolean	Z

Tablo 1.5'de class veri türünün kullanımını biraz daha detaylandıralım. Veri türü dizi ve nesne referansı türünde ise alan tanımlayıcıları aşağıda verilen tablodaki gibi olacaktır. Buradaki L ifadesinin sonundaki ";" daha önceden parametreleri ayırmak için kullandığımız virgül(,) gibi değil, tip ifadesinin sonlandırıcısı olarak kullanılmaktadır.

Tablo 1.5. Sınıfların Referens Elemanlarına Erişmek için Kullanılan Tanımlayıcılar

Veri Türü	Alan Tanımlayıcısı
String	“Ljava/lang/String;”
Object[]	“Ljava/lang/Object”
Int[]	“[I”

Temel veri türlerinin dizileri türündeki alanların alan tanımlayıcıları, o türün alan tanımlayıcı değerinin başına “[“ karakteri eklenerek elde edilmektedir.

Elde edilen bu fieldID değeri kullanılarak veri üyesine erişim GetObjectField fonksiyonu ile sağlanmaktadır.

jstring jstr1 = env->GetObjectField(nesnem,fid);

1.3.5. JNI Teknolojisinin Dezavantajları

JNI kullanımı ileri seviye programcılık deneyimi gerektirmektedir. Bunun yanında Java ile yerel fonksiyonların kullanıldığı programlarda bazı sınırlandırmalar ortadan kalktığından bazı problemler ortaya çıkabilmektedir. Bu nedenle JNI kullanımında aşağıdaki durumlar göz önünde bulundurulmalıdır [29] .

- Bahsedilen JNI API fonksiyonlarının öğrenilmesi ve kullanılması kolay değildir.
- JNI kullanımında karşılaşılan hataların JVM içerisinde kolayca bulunup düzeltilmesi mümkün değildir. Çünkü debug yapmak kolay değildir.
- Sadece belirli programlar ve işaretli appletler tarafından kullanılabilir.
- Programın platform bağımsızlık özelliği kaybedilmiş olur. Çünkü yerel fonksiyonların kullanımı ile çalıştırılan platformdaki fonksiyonlara artık bağımlılık söz konusudur.
- JNI tarafında artık temizleme işlemi programcıya bırakılmıştır. Çünkü JNI otomatik artık temizleme işlemini desteklemez.
- Hata kontrolünü kullanıcı dikkatli bir şekilde yapmalıdır. Çünkü JNI tarafında JVM’nin hata verme olasılığı yüksektir.

1.4. Saldırı Tespit Sistemleri

1.4.1. Saldırı Türleri

Bilgisayar sistemlerinde en genel anlamda sızma ya da saldırı makine konsolundan yapılacak izinsiz erişimlerden başlar ve çok geniş bir spektruma yayılır. Bilgisayar ağları söz konusu olduğunda ise saldırılar sadece bu tip kullanıcı ve erişim temelli saldırılar ile sınırlı kalmaz.

Ağ üzerinden yapılan saldırılar günümüzde en sık karşılaşılan problemlerdir. Bu tip saldırılar 4 temel kategoride incelenebilirler [33].

- Bilgi Tarama (Probe ya da scan): Bu saldırılar bir sunucunun ya da herhangi makinenin, geçerli ip adreslerini, aktif portlarını veya işletim sistemini öğrenmek için yapılan saldırılardır.
- Hizmet Engelleme (Denial of Service - DoS): Bu saldırılar genelde TCP/IP protokol yapısındaki açıklardan faydalanarak veya bir sunucuya çok sayıda istek yönelterek onu tıkamaya sebep olan saldırılardır.
- Yönetici Hesabı ile Yerel Oturum Açma (Remote to Local - R2L): Kullanıcı haklarına sahip olunmadığı durumda misafir ya da başka bir kullanıcı olarak izinsiz erişim yapılmasıdır.
- Kullanıcı Hesabının Yönetici Hesabına Yükseltilmesi (User to root - U2R): Bu tip saldırılarda sisteme girme izni olan fakat yönetici olmayan bir kullanıcının yönetici izni gerektirecek işler yapmaya çalışmasıdır.

1.4.2. Saldırı Tespit Sistemlerinin Sınıflandırılması

Saldırı tespit sistemleri çok değişik parametrelere göre sınıflandırılabilir. Ancak burada en çok kullanılan birkaç parametreyi dikkate alacağız [30]. Özellikle kullanılan yöntem üzerinde duracağız. Mevcut sistemlerden bazılarına dikkati çekeceğiz.

1.4.2.1. Veri İşleme Zamanına Göre

Burada saldırı tespit sistemleri “gerçek zamanlı” ve “gerçek zamanlı olmayan” şeklinde ikiye ayrılırlar. Gerçek zamanlı olmayan sistemlerde veri önce depolanır, sonra analiz için ilgili saldırı tespit sistemlerine gönderilir. Gerçek zamanlı sistemlerde veri o

anda analiz edilir. Bu tip saldırı tespit sistemleri yoğun bilgi akışı olan ağlarda uygulanması zor olan ama aktif olarak cevap üretilmesi gereken durumlarda da tek çözüm olan sistemlerdir. Ticari uygulamaların gerçek zamanlı olması tercih edilir.

1.4.2.2. Mimari Yapıya Göre

Saldırı tespit sistemleri ağ üzerinden birçok noktadan veri toplayıp bir merkezde işleyebilirler. Bu tip mimarilere sahip saldırı tespit sistemlerine dağıtık saldırı tespit sistemleri denir. Bunun karşısında ağ üzerinden tek bir yerden veri toplayan saldırı tespit sistemleri de vardır. Bilinen saldırı tespit sistemlerin çoğu bu kategoriye girer.

1.4.2.3. Bilgi Kaynaklarına Göre

Saldırı tespit sistemleri bilgi kaynaklarına göre iki sınıfa ayrılılar: ağ temelli saldırı tespit sistemleri ve sunucu-temelli saldırı tespit sistemleri.

Ağ temelli saldırı tespit sistemlerin temel amacı ağ üzerinden yapılan saldırıları ağ trafiğini gözetleyerek tespit etmektir. Ağ paketlerini yakalayıp bunları analiz ederek saldırı tespiti yaparlar.

Sunucu temelli saldırı tespit sistemleri bir bilgisayar sistemi içerisinde toplanan veriler üzerinde çalışırlar. Bu şekilde işletim sistemine yönelen saldırılar için sorumluluğun hangi sistem çağrılarında ve hangi kullanıcılarda olduğu tespit edilebilir. Sunucu-temelli saldırı tespit sistemleri iki tür bilgi kaynağı kullanır: işletim sistemi izleme seçenekleri ve sistem günlük dosyaları. Bunların ağ temelli sistemlere göre bir avantajı, olayların olduğu yerel sunucuyu izleme yetenekleri sayesinde ağ-temelli saldırı tespit sistemlerinin yakalayamayacağı saldırıları tespit edebilmesidir. Bu saldırılar çoğu zaman fazla trafik yaratmayan R2L veya U2R saldırılarını içerir

1.4.2.4. Saldırı Tespit Yöntemlerine Göre

Saldırı tespit sistemleri saldırı tespit yöntemlerine göre sınıflandırılırsa iki temel yöntem söz konusudur. Bunlardan ilki anormallik tespitine dayanır. Diğeri kötüye kullanım tespiti ya da bazı kaynaklarda tanımlandığı üzere imza-tanıma üzerinden gider. Aşağıda iki yaklaşım da ayrıntılı olarak incelenecektir.

1.4.2.4.1. Anormallik Tespiti

Anormallik (anomaly) normal davranıştan sapma anlamına gelir. Burada normal davranıştan farklılık gösteren davranışların saldırı olarak işaretlenmesi söz konusudur. Normal bir sistemde kullanıcı istekleri tahmin edilebilir istatistiksel değerlerle uyuşur. Burada normal davranışın bilinmesi ve modellenmesi esastır. Ancak bundan sonra bir anormallik varsa tespit edilebilir. Belirli kurallar zinciri ile tanımlanabilir veya başka yöntemler uygulanabilir ama sonuçta bu tanımlı ya da öğrenilen davranıştan sapma anormallik olarak değerlendirilir ve sapmanın şiddetine göre saldırı olarak sınıflandırılabilir. Bu yöntemin avantajı daha önceden tanınmayan saldırıların keşfedilebilmesi olasılığıdır. Dezavantajı ise yanlış alarmların (false alarm/positive) sayısının yüksek olmasıdır [31]. Yanlış alarm tanımı kabaca "var olmayan" bir durumu "var" olarak tanımaktır.

Anormallik tespitinde istatistiksel yöntemler, yapay sinir ağları, veri madenciliği bilgisayar bağışıklık sistemi (computer immunology) gibi birçok yaklaşım uygulanabilir [32]. Bunları kendi içinde sınıflandırmak gerekirse [30]:

1. Kendi-kendine öğrenen sistemler: Belli bir veri seti ile eğitilerek normal davranışı öğrenirler. Bunlar zaman serilerini kullanmayan kural tabanlı sistemler, açıklayıcı istatistikleri kullanan sistemler ve zaman serilerini kullanan yapay sinir ağları temelli teknikler olarak üçe ayrılırlar.
    - 1.1. Zaman serisi kullanmayan sistemler:
        - 1.1.1. Kural Tabanlı (Rule Based) Sistemler: Sistem kendisi trafiği inceleyip kurallar oluşturuyor ve saldırı tespiti sırasında bu kararlara göre davranıyorsa bu sınıfa girer.
        - 1.1.2. Açıklayıcı İstatistikleri (Descriptive Statistics) Kullanan Sistemler: Kullanıcı profilinin basit istatistiklerle oluşturulup, buradan uzaklık vektörlerini (distance vector) kullanarak karar alan sistemlerdir.
    - 1.2. Zaman serisi kullanan sistemler:
        - 1.2.1. Yapay Sinir ağları (Artificial Neural Networks – ANN) yaklaşımı: Burada sistem önceden bir zaman serisi ile eğitilir ve çalışmaya başladığında, buradan öğrendikleri ile karar verir.
2. Programlanan Sistemler:

2.1. Açıklayıcı İstatistikleri kullanan sistemler: Burada yukarıdakinden farklı olarak profiller daha önceden tanımlanmıştır. Profilleri sınırlayan belli bir eşik değeri, bir istatistiksel büyüklük veya basit bir kural olabilir.

2.2. Bilinmeyeni reddet (Default Deny) yaklaşımı: Tek başına çok sık kullanılan bir yaklaşım olmasa da, sistem davranışındaki durum-geçiş değerlerini kontrol ederek, farklı bir geçiş (transition) görüldüğünde izin vermemek şeklinde bir yaklaşıma sahiptir.

2.3. Sinyal İşleme Tekniklerini Kullanan yaklaşım: Burada sistem parametreleri sinyal işleme teknikleriyle belirlenir ve saldırı içermeyen normal veri setinden çıkartılan eşik değerleri ile anormallik tespiti yapılır. Bu yaklaşım diğerlerine göre yenidir ve çok geniş bir alanı kapsamaktadır. Henüz sınırlı sayıda sinyal işleme tekniği saldırı içeren zaman serilerine uygulanmıştır. Özellikle son yıllarda popüler olan temel bileşen analizi (Principle Component Analysis - PCA) yaklaşımı ile elde edilen öz değerlerin sistem parametresi olarak kullanılması bu alandaki ilginç çalışmalardan biridir.

1.4.2.4.2. İmza Tanıma Temelli Saldırı Tespiti

Kötüye kullanım tespiti (Misuse detection) ya da imza-tanımaya dayalı sistemlerde her davranışın bir imzası-karakteri vardır. Bunlar daha önce görülen davranış şablonlarıdır. Eğer gözlemlenen davranış daha önceden bilinen bir saldırı imzası ile eşleşiyorsa saldırı olarak sınıflandırılır. Daha önce karşılaşılmadıysa saldırı olarak nitelenmez. Avantajı saldırıyı kesin olarak tanıyabilmesidir. Yani yanlış alarm vermezler fakat yeni bir saldırı gelirse bunu sezemezler. İmza temelli sistemler daha çok ticari sistemlerde kullanılırlar.

İmza-tanımaya dayalı sistemlere ait yaklaşımlar sınıflandırılırsa [30]:

1. Programlanan Sistemler: Kendi kendine öğrenen hibrid bir teknik olan Ripper dışında tüm imza temelli teknikler bu sınıfa girerler.
2. Durum modellemesi: Burada tüm davranışlar durumlara karşı düşer. Eğer bir davranış daha önceden tanımlı durumlara ve durum geçişlerine denk düşen hareketler yapıyorsa saldırı olarak tanınır.
3. Uzman sistemler: Sızma belirleme sistemlerinin ilkleri kural-tabanlı (rule-based) uzman sistemlerdir.

4. Örüntü eşleme (Pattern Matching): Sistemde daha önceden tanımlanmış, olmaması gereken bazı sözcüklerin tanınmasına yardım eder. Esnek değildir fakat basittir. Örneğin "parola dosyasını kopyala" komutu görüldüğünde bunun bir saldırı olduğunu en basit şekilde bu yöntem tespit eder.

Aşağıda Tablo1.5'de saldırı tespit sistemlerinin sınıflandırma şeması ve bu sınıflarda yer alan örnek sistemler listelenmiştir.

Tablo 1.6. Saldırı tespit yöntemlerinin ve araçlarının sınıflandırılması

Anormallik Tabanlı	Kendi-kendine Öğrenen Sistemler	Zaman serisi kullanmayan	Kural tabanlı	**W&S**
			Açıklayıcı İstatistikler	**IDES NIDES EMERALD Haystack**
		Zaman serisi kullanan	Yapay sinir ağları	**Hyperview**
	Programlanan Sistemler	Açıklayıcı İstatistikler	Basit istatistikler	**MIDAS NADIR**
				Haystack
			Basit kural	**NSM**
			Eşik değeri	**Computer Watch**
		Bilinmeyeni Reddet	Durum Modeli Serileri	**JANUS Bro**
İmza Tanıma tabanlı	Programlanan Sistemler	Durum Modellemesi	Durum Geçiş	**USTAT**
			Petri-net	**IDIOT**
		Uzman sistem	**NIDES EMERALD MIDAS DIDS**	
		Örüntü eşleme Basit kural	**NSM NADIR Bro HayStack**	
İmza esinlenmeli	Kendi kendine öğrenen	Otomatik özellik seçme	**Ripper**	

Tablo 1.6'de bazı sistemlerin birden fazla kategoride yer alması, ilgili yaklaşımların tümünü içerdiğini gösterir. Burada amacımız varolan bu sistemlerin incelenmesi olmadığından dolayı sistemler hakkında sadece kısa bilgi vermekle yetineceğiz.

Haystack: Amerikan Hava Kuvvetlerinde kullanılan Unisys 1100/60 ana bilgisayarları için tasarlanmış bir saldırı tespit sistemidir. 1988 yılında geliştirilmiştir [34].

MIDAS: 1988'de National Computer Security Centre (Ulusal Bilgisayar Güvelik Merkezi) için geliştirilmiştir. Uzman sistem tabanlıdır [35].

IDES (Intrusion Detection Expert System): IDES 1988-1992 yılları arasında üzerinde çalışılan birçok sistemi içerir. Bu arada birçok versiyonu çıkmıştır ve son olarak NIDES (Next-Generation Intrusion Detection Expert System) adını almıştır [36].

W&S (Wisdom and Sense – Zeka ve His): Geliştirilmesine 1984 yılında başlanmasına rağmen ilk makalesi 1989'da çıkmıştır. Zeka kısmı geçmişteki toplanan verileri inceleyerek normal davranışı oluşturması anlamına gelir. His kısmı ise bunların kural haline getirilip uzman sisteme verildikten sonra anormal davranışların yakalanması anlamına gelmektedir [37].

ComputerWatch: AT&T Bell Laboratuarları tarafından ticari bir ürün olarak 1990 yılında geliştirilmiştir [38].

NSM (Network Security Monitor): 1990-94 arasında geliştirilmiş olup IDES gibi çeşitli revizyonlardan geçmiştir. NSM ağı dinleyerek, ağın kullanımıyla ilgili bir profil geliştirir ve geçerli kullanımı onunla karşılaştırır. Elde edilen veri beklenen bağlantı verisiyle karşılaştırılır ve beklenen aralıkta çıkmayan her veri anormal olarak işaretlenir. Biz çalışmamızda NSM tipinde bir STS gerçeklenmesi üzerinde duracağız [39].

NADIR: 1991-93 yılları arasında Los Alamos Laboratuarlarında geliştirilmiştir. NADIR kullanıcılar hakkında haftalık istatistikler tutar. Daha sonra bu istatistikleri uzman sistem kurallarıyla karşılaştırır [40].

Hyperview: 1992 yılında geliştirilmiş diğer sistemlerden oldukça farklı bir sistemdir. İki ayrı parçadan oluşur. İlki davranışları izleyen ve sınıflayan bir uzman sistem, ikincisi buradan öğrendikleriyle eğitilen yapay sinir ağlarını içeren parçadır [41].

DIDS: 1992 yıllarında geliştirilmiş dağıtık mimarili sistemleri kapsar. Bu tip sistemlerde ağın değişik noktalarından veriler toplanır ve bir merkezde incelenir [42].

USTAT: 1993-95 yılları arasında geliştirilmiştir. Durum geçiş analizi yapar. Eğer bir davranış saldırı için tanımlı durum geçişlerini yapıyorsa saldırı olarak sınıflandırılır [43].

IDIOT: 1994-96 yılları arasında CERIAS (Center for Education and Research in Information Assurance and Security) tarafından geliştirilmiştir. Örüntü tanıma için petri-net'lerin kullanımına dayanır [44].

JANUS: 1996'da Berkley'de geliştirilmiş bir sistemdir [45].

EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances): 1997-98 yıllarında geliştirilmiş ölçeklenebilir, dağıtık bir sistemdir[46].

Bro: 1988'de Vern Paxson tarafından geliştirilmiş bir sistemdir. Gerçek zamanda ağ trafiğini pasif olarak gözlemleyerek saldırı tespiti yapmaktadır. Birçok değişik özelliği kendinde barındırması açısından sıkça söz edilen bir sistemdir [47].

Ripper: 1999 yılında geliştirilen bu sistem veri madenciliği yaklaşımını kullanır. Ripper DARPA değerlendirmesine katılmış olan sistemlerden bir tanesidir [48].

Burada adı geçen sistemlerden başkaca birçok STS mevcuttur. Özellikle günümüzde anormallik tespitinde en popüler yaklaşımlar zaman serilerini kullanmak veya veri setinden çıkartılan özellikler (feature) ile uzaklık hesabı yapmaktır. Zaman serisi kullanılacaksa sinyal işleme teknikleri de rahatça kullanılabilme olanağı bulurlar. Verinin herhangi bir zaman serisi olarak elde edilebilmesi uygulama alanını genişletir. Belirli aralıklarla olaylar zamanda dizilirse, buradan en basit istatistiksel testler (ortalama ve diğer momentler) başlayarak, spektrum kestirimi, özdeğer analizi, dalgacık analizi gibi birçok karmaşık sinyal işleme tekniğine de yol açılmış olur. Sinyal işleme ile saldırı tespit sistemlerinin birleştirilmesi çok sayıda çalışmaya konu olmuştur [49,50]. Halen bu konu üzerinde birçok çalışma devam etmektedir.

1.4.3. Tespit Edilen Saldırılara Karşı Alınacak Önlemler

Saldırı olduktan sonra alınabilecek önlemler saldırının tekrar olmasını engellemeye yarar ve o andaki saldırı için hiçbir şey yapamazlar. Bu önlemler biri, örneğin, yazılım açıklarından kaynaklanan saldırıları önlemek için yazılım yaması yapmak olabilir. Ama asıl önemli olan saldırı anında önlem alabilmektir. Saldırılara karşı iki farklı önlem gerçeklenebilir; pasif gözetleme ve erişim engelleme.

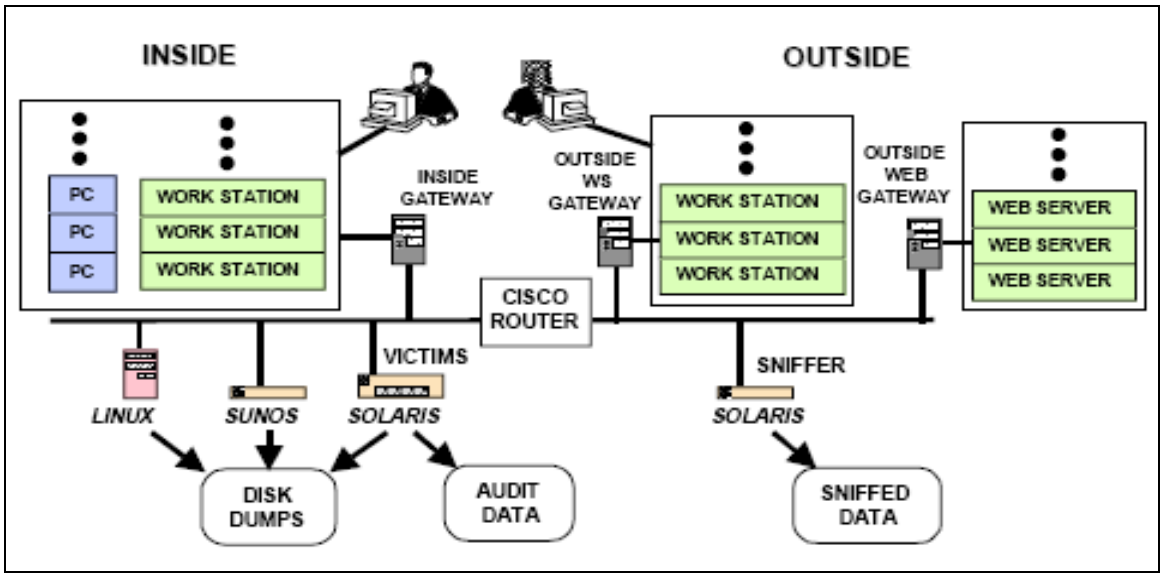
a) Pasif Gözetleme: Bu yaklaşımda saldırganın davranışları kaydedilir, saldırıda araya girilmez. Amaç, saldırganın istediği şeyi ve kullandığı yöntemleri öğrenmektir. Burada en büyük sorun, sistemin savunmasız bırakılmasıdır. Saldırgan bu savunmasız durumda diğer sistemlere de saldırabilir.

b) Erişimi Engelleme: Burada amaç saldırganın erişebileceği sistem kaynaklarını sınırlamaktır.

Saldırıya karşı yukarıdaki önlemler alındıktan sonra, saldırıya karışmış sistemlerin erişimi kaldırılır veya süreçleri sonlandırılır.

1.4.4. Saldırı Tespit Sisteminde Kullanılan Veri Seti

Saldırı tespit sistemlerinin test edilmesinde en zorlu kısım güvenilir veri setleri elde edilmesidir. İnternet ortamından elde edilmiş verilerin çoğu saldırı var ya da yok bilgisi içermez. Herhangi bir ağda veya bilgisayarda bu bilginin edinilmesi için ağ veya bilgisayar yöneticisinin sistemi gözlüyor olması ve sistem kayıtlarını depoluyor olması gerekir. Ancak bu işlem normal sistemler için oldukça masraflı ve zahmetli bir iş olduğundan gerçekleştirilmemektedir. Bu zorluklara rağmen saldırı tespit sistemlerinin test edilmesi için bu şekilde depolanmış veri setlerine ihtiyaç vardır. Bu amaç ile saldırıları içeren bir simülasyon yapılarak istenilen veri seti oluşturulmalıdır. Bu şekilde, DARPA'nın sponsorluğunda MIT Lincoln Laboratuarlarında saldırı tespit sistemleri için bir karşılaştırma ortamı sunan IDEVAL veri setlerini oluşturulmuştur [55]. Özellikle saldırı tespit sistemlerinin sınanması amacıyla saldırıların hedefi olacak bir ağ ve saldırıları gerçekleştiren başka bir ağ dizayn edilmiştir. İç ağda Amerikan Hava Kuvvetlerindeki bir yerel ağın simülasyonu gerçeklenmiştir. 1998' de kullanılan ağ aşağıdaki gibidir.



Şekil 1.7: DARPA 1998 veri setini oluşturmak için kullanılan ağın yapısı

Şekil 1.7' de iç ağda hava kuvvetlerinin ağının bir simülasyonu olarak 100 farklı host ve kullanıcı otomatik olarak oluşturulmuştur. Dış ağ ise interneti simule edecek şekilde 1000 farklı web sayfası veya kullanıcıyı temsil edecek şekilde oluşturulmuştur. Sistemde kurban olarak yerel ağdaki Linux, SunOS ve Solaris makineları seçilmiştir. Ağ ortamında iç ve dış ağdaki paketler toplanmıştır. Solaris makinesi üzerinde koşan süreçleri izlenerek BSM (Basic Security Module) verileri depolanmıştır. Ayrıca her gece kurban olan 3 makinenın disk yedekleri tutulmuştur. Bu veri depola işlemi 9 hafta devam etmektedir. Bu 9 haftanın ilk 7 haftası eğitim verisi olarak düşünülmüştür. Geriye kalan 2 hafta ise test işlemi için kullanılması düşünülmüştür[55].

Tablo 1.7. Darpa 1998' de veri setleri oluşturulurken kullanılan ataklar [55].

	Solaris	**SunOS**	**Linux**	**Cisco Router**
Hizmet Engelleme	apache2 back mailbomb neptune ping of death process table smurf syslogd udp-storm	apache2 back land mailbomb neptune ping of death process table smurf udp-storm	apache2 back mailbomb neptune ping of death process table smurf teardrop udp-storm	
Yönetici Hesabı ile Yerel Oturum Açma	dictionary ftp-write guest http-tunnel phf xlock xsnoop	Dictionary ftp-write guest phf xlock xsnoop	dictionary ftp-write guest imap named phf sendmail xlock xsnoop	snmp-get
Kullanıcı Hesabının Yönetici Hesabına Yükseltilmesi	at eject ffbconfig fdformat ps	loadmodule	perl xterm	
Bilgi Tarama	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan

Tablo 1.7' de gösterilen ataklar belirtilen cihazlara otomatik olarak gerçekleştirilmiştir. Bu atakların hangi oturumları ve başarı durumları depolanan veriler içerisinde belirtilmiştir.

Bir günlük veri içerisinde ağdan toplanılan TCP paketleri, TCP bağlantılarının listesi, Solaris makineden toplanan BSM verilerinin ikili ve ASCII dosyaları, Solaris makineye yapılan bağlantıların listesi ve düzenli olarak alınmış solaris makinede koşan süreçlerin listesi bulunmaktadır. Bu çalışmada sistem çağrıları ile ilgilenildiğinden solaris makineden alınan BSM verileri bizim için önemlidir. Aşağıda gerçekleştirilen *ioctl* sistem çağrısı için kaydedilen BSM verisi görülmektedir.

header,181,2,ioctl(2),,Fri Jun 05 07:47:43 1998, + 580503971 msec
path,/devices/pseudo/clone@0:udp
attribute,20000,root,root,8388608,51464,10747906
argument,2,0x5308,cmd
argument,3,0xeffddb28,arg
argument,2,0x501cc90c,strioctl:vnode
subject,-2,root,root,root,root,96,0,0 0 0.0.0.0
return,success,0
trailer,181

Bu BSM verileri içerisinde *header* ile başlayan satırda altı çizili olarak gösterilen kısım yapılan sistem çağrısını göstermektedir. s*ubject* ile başlayan satırda altı çizili olan kısım hangi sürecin gerçekleştirdiğini göstermektedir[56]. Diğer kısımlarda yapılan sistem çağrısının parametreleri, geri dönüş değerleri, ne zaman yapıldığı gibi verileri içermektedir.

Bu DARPA veri setlerinin kullanıldığı çalışmalara örnek olarak [57-59] verilebilir. Bu çalışmalarda sistemin eğitimi ve test işlemleri DARPA veri setleri kullanılarak gerçekleştirilmiştir.

1.5. K En Yakın Komşuluk Algoritması

Bu çalışmamızda saldırı tespit yöntemi olarak anormallik tespitini kullanması düşünüldü. Sistemde çalışan süreçlerin davranışlarının sınıflandırılması için metinleri sınıflandırmak amacı ile kullanılan K en yakın komşuluk algoritması kullanılabilir. Çalışmamız da Y. Liao tarafından geliştirilen K en yakın komşu algoritması ile anormallik tespiti [53] çalışmasından esinlenerek yapılmıştır.

Bu çalışma için sistemde yapılan sistem çağrılarını kelimeler ve bu sistem çağrıları dizisini de metin olarak düşünüp bu algoritmayı kullanabiliriz. Aslında saldırı tespit sistemlerinde K en yakın komşuluk algoritmasının kullanılmasında metin sınıflandırmaya göre daha fazla avantaj vardır. Çünkü herbiri bir kelime olarak kabul edilen sistem çağrılarının sayısı normal metinlerde kullanılan kelime miktarından çok daha fazladır. UNIX sistemleri için sistem çağrısı sayısı 255'i geçmezken tipik bir dilde kullanılan kelime miktarı 15000 kelimeyi aşmaktadır [54]. Bu süreçler için oluşturulan vektörlerin boyutunu da oldukça düşürmektedir.

K en yakın komşuluk algoritması sorgu vektörünün en yakın k komşuluktaki vektör ile sınıflandırılmasının bir sonucu olan denetlemeli öğrenme algoritmasıdır. Bu algoritma ile yeni bir vektörü sınıflandırabilmek için doküman vektörü ve eğitim dokümanları vektörleri kullanılır. Bir sorgu örneği verilir, bu sorgu noktasına en yakın k tane eğitim noktası bulunur. Sınıflandırma ise bu k tane nesnenin en fazla olanı ile yapılır. K en yakın komşuluk uygulaması yeni sorgu örneğinin sınıflandırmak için kullanılan bir komşuluk sınıflandırma algoritmasıdır.

K en yakın komşuluk algoritması çok kolaydır. K en yakın komşulukları bulmak için sorgu örneği ile eğitim dokümanları arasındaki en küçük uzaklıklar dikkate alınır. En yakın komşuları bulduktan sonra bu komşulardan kategorisi en çok olanın kategorisi dokümanın kategorisini tahmin etmekte kullanılır [51].

Avantajları;

- Uygulanabilirliği basit bir algoritmadır.
- Gürültülü eğitim dokümanlarına karşı dirençlidir.
- Eğitim dokümanları sayısı fazla ise etkilidir.

Dezavantajları;

- K parametreye ihtiyaç duyar.
- Uzaklık bazlı öğrenme algoritması, en iyi sonuçları elde etmek için, hangi uzaklık tipinin ve hangi niteliğin kullanılacağı konusunda açık değildir.
- Hesaplama maliyeti gerçekten çok yüksektir çünkü her bir sorgu örneğinin tüm eğitim örneklerine olan uzaklığını hesaplamak gerekmektedir.
- En yakın komşuluk prensibine dayanır. Tüm dokümanlar vektörel olarak temsil edilir. Sorgu dokümanı ile diğer dokümanlar arasındaki kosinüs benzerliği hesaplanır. Benzerlik oranı 1'e en yakın olan n tane vektörün kategorisinden çok olanı dokümana atanır.

Ayrıca eşik değerinin seçimi de önemli bir konudur. Yanlış seçilmesi durumunda hata oranı artmaktadır. Eşik değeri yüksek olduğu zaman bazı saldırılar normal davranış göstermekte ve tespit edilmeleri mümkün olamamaktadır.

A= a_{ij} şeklinde matris olarak verilebilir, a_{ij}, j numaralı dokümanda bulunan i numaralı kelimenin ağırlığıdır.Sınıflandırma işleminde A matrisinin oluşturulması eğitim işleminin gerçekleştirilmesi anlamına gelmektedir. A matrisinde eğitim işleminde kullanılan doküman sayısı kadar satır vardır. Her bir satır bir eğitim vektörünün ağırlıklandırılmış a_{ij} değerlerinden oluşur. A matrisi için sütunların sayısı dokümanların içerisindeki kelimelerin adedine bağlıdır. Bu kelimelerin adedi çok olduğu zaman azaltmak gerekir. Bu da özellik seçimi gibi bazı yöntemlerle yerine getirilir.

Bu a_{ij} ağırlıklandırılmış değerlerin hesabı için bazı değişik yöntemler bulunur. Bunlardan birisi (1.1)' de görüleceği gibi doküman içerisinde gelen kelimelerin frekansıdır. (1.1)' de f_{ij}, j dokümanı içerisinde i numaralı kelimenin meydana gelme sıklığını verir. Diğer bir ağılıklandırma yöntemi ise *tf*idf* (term frequency - inverse document frequency) olarak bilinir (1.2). Bu yöntemde bütün eğitim dokümanları göz önüne alınarak ağırlıklandırma gerçekleştirilir. (1.2)' de N, bulunan doküman sayısını, n_i ise bütün test dokümanlarında i numaralı kelimelerin meydana gelme sıklığını verir. M, birbirinden farklı olan kelimelerin sayısını vermektedir. En basit yaklaşım boolean ağırlık bulma yaklaşımıdır. Bu yaklaşımda eğer kelime doküman içerisinde kullanılmışsa, kaç defa geldiğine bakmaksızın 1 değerine setlenir diğer durumda 0 değerine setlenir. Diğer basit bir yaklaşımda sıklık ağırlığı yöntemidir. Buna göre doküman içerisinde kelimelerin kaç kez tekrar ettiği bulunur [52].

$$a_{ij} = f_{ij} \tag{1.1}$$

$$a_{ij} = \frac{f_{ij}}{\sqrt{\sum_{l=1}^{m} f^2{}_{ij}}} * \log(\frac{N}{n_i}) \tag{1.2}$$

Eğitim işleminin tamamlanmasından sonra yeni dokümanların sınıfının belirlenmesi için (1.3)' de gösterilen sim(X, D_j) fonksiyonu kullanılır. Burada X test dokümanıdır. D_j, j numaralı eğitim dokümanıdır. $(X \cap D_j)$ ifadesine t_i olarak kabul edersek, t_i X ve D_j tarafından paylaşılan bir kelimedir. X_i, X içerisindeki t_i kelimesinin bulunma sayısıdır

(ağırlığıdır). d_{ij}, D_j dokümanında bulunan t_i kelimelerinin sıklığını verir. (1.4)' de ise test işlemi sırasında dokümanların değerlerinin normunun hesabı gösterilmektedir. x_i değeri, herbir kelimenin ağırlık değerini belirtir.

$$sim(X, D_j) = \frac{\sum \in (X \cap D_j) X_i * d_{ij}}{\|X\|_2 * \|D_j\|_2} \tag{1.3}$$

$$\|X\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots} \tag{1.4}$$

2. YAPILAN ÇALIŞMALAR

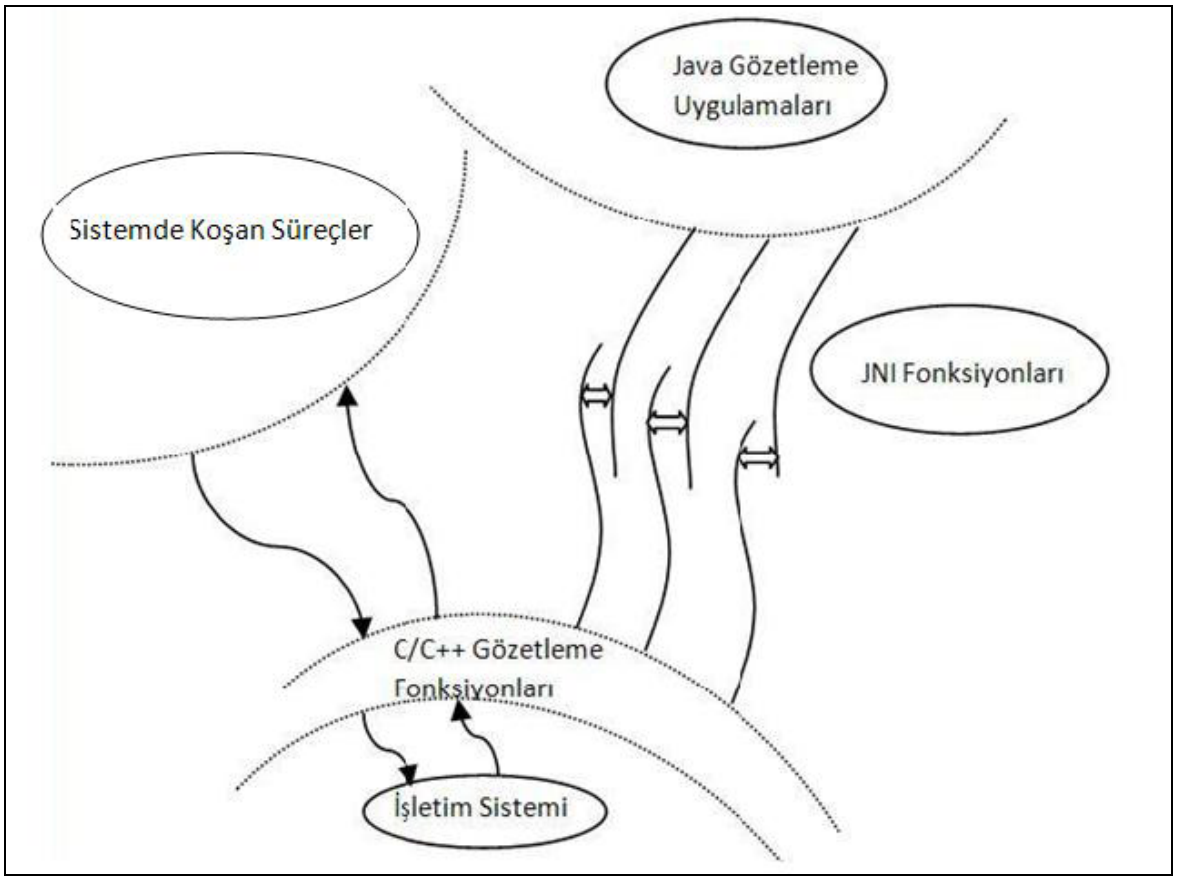
2.1. Giriş

Bu çalışmada, UNIX işletim sistemlerinde süreç izleme gereksinimleri için kullanılan /proc dosya sisteminin Java programlama çevresine entegrasyonunu sağlayan bir çatı (framework) geliştirilmiştir. Bu şekilde Java programlama çevresinden çalışan süreçlerin izlenmesi mümkün hale getirilmiştir.

2.2. JUSİ Çatısının Tasarımı ve Gerçeklenmesi

2.2.1. JUSİ Çatısının Yapısı

Java uygulamalarının UNIX süreçlerini gözetleyebilmeleri için geliştirilen bu mekanizma Java'dan UNIX Süreçlerini İzleme (JUSİ) olarak isimlendirilmiştir. Öncelikli olarak bu çatı ile Java programlama dilinin erişimine izin verilmeyen işletim sistemi veri yapılarına erişimi sağlanmıştır. Ayrıca programcıların Java ortamında yapacakları kodlamanın mümkün olduğu kadar işletim sistemin karmaşık alt yapısından soyutlanması amaçlanmıştır. Genel bir ilke olarak, süreçlerin gözetlenmesinde gerekli olan C++ fonksiyonlarının daha kolay bir kullanımı sağlanacak şekilde Java çevresi eşdeğerleri yazılmıştır. Şekil 2.1'de Java uygulamalarından işletim sistemine giden yol üzerinde yer alan ortamlar gösterilmiştir.



Şekil 2.1. Java ortamında süreç gözetleme mimarisi

Şekil 2.1'de de görüleceği gibi sistemde çalışan süreçleri izleme işlemi C/C++ fonksiyonları tarafından yapılmaktadır. JUSI çatısı JNI fonksiyonları sayesinde izlemeyi gerçekleştiren C/C++ fonksiyonları ile iletişime geçip veri alış verişinde bulunmaktadır. Yani Java' nın direk olarak sistemdeki veri yapılarına erişimi söz konusu değildir

2.2.2. Yerel Gözetleyicinin Yapısı

Bu kısımda temel olarak Java'da süreçleri gözetlemek için C++ tarafından kullanacağı bazı önemli kısımlarından bahsedeceğiz. Süreç gözetlemede doğrudan erişilen işletim sistemi verileri ve izleme işleminde kullanılan sistem bileşenleri önemlidir.

Gözetleyici */proc* dosya sistemi yardımıyla kullanıcı süreçlerini gözetler. */proc* dosya sistemi birçok sürecin gözetlenmesine izin verir. İzlenilmek istenen sürece ait dosya üzerinde gerekli ayarlamaları yaparak izleme işlemi gerçekleştirilir.

Gözetleyici, süreç tanımlayıcısı */proc* dizininde bulunan söz konusu sürece tutunur. Burada tutunmaktan maksat izlenilmek istenen sürece ait dosyanın açılması ve üzerinde gerekli olan işlemlerin yapılmasıdır. Bu dosyayı açma işlemi aşağıdaki gibi yapılmaktadır.

```
sprintf( proses_adı, "/proc/%d", pid);
procfd = open (proses_adı, 0_RDWR | 0_EXCL);
```

Sürece bir kere tutunduktan sonra süreci kontrol etmek için açık dosya tanımlayıcısı üzerinde *ioctl* sistem çağrılarını kullanılır. İşletim sistemine, ilgilenilen olaylardan birini gerçekleştirdiğinde sürecin durdurulması isteğini bildirilir. İlgilenilmesi gereken iki durum mevcuttur. Bunlardan biri sistem çağrılarının çekirdeğe girişi diğeri de sistem çağrılarının çekirdekten çıkışıdır. *open, creat* gibi çağrıların sadece çekirdek girişinde meydana gelen durma sırasında kontrol edilmesi yeterlidir. Aşağıdaki kod parçası gözetleyicinin işletim sistemine hangi sistem çağrılarının çekirdeğe girişte durdurulması gerektiğini nasıl bildirdiğini göstermektedir.

```
sysset_t SysCalls;
premptyset (&SysCalls);
praddset (&SysCalls, SYS_open);
praddset (&SysCalls, SYS_creat);
ioctl (procfd, PIOCSENTRY, &SysCalls);
```

Diğer bazı sistem çağrıları (fork, vfork, vs.) gözetleyicinin işletim sisteminden dönen değerleri almasını sağlamak için çekirdekten çıkış noktasında durdurulurlar. Örneğin, fork sistem çağrısının döndürdüğü sonuç, çocuk sürecin PID değeridir ve gözetleyici bu değeri o çocuk sürecin /proc dosyasına tutunmak için kullanır. Bu sayede söz konusu sürecin tüm yarattığı süreçler de takibe alınabilir.

```
preemptyset (&SysCalls);
praddset (&SysCalls, SYS_fork);
praddset (&SysCalls, SYS_vfork);
ioctl (procfd, PIOCSEXIT, &SysCalls);
```

Gözetlenen sürecin çocuk süreçlerinin de gözetleyen süreç tarafından takip edilebilmesi için gözetleyici, her çocuk sürecin gözetleme bayraklarını bulmalıdır. Çocuk süreçlerin ana süreçten tüm gözetleme bayraklarını miras almalarını temin etmek için, gözetleyicinin gözetlenen sürecin *inherit-onfork* bayrağını setlemesi gerekmektedir.

```
longflags = PR_FORK;
ioctl (procfd, PIOCSET, &flags);
```

İlgilenilen olaylar işletim sistemine bildirildikten sonra, yeni bir *ioctl* komutu gözetleyicinin sürecin durmasını beklemesini sağlamak için kullanılır.

```
prstatus_t *pstatus;
ioctl (procfd, PIOCWSTOP, &pstatus);
```

Bu aşamada, durma gerçekleşir gerçekleşmez gözetleyici, sürecin kaydedicilerini ve durum bilgisini okur. Bu verileri sistem çağrılarının parametrelerini ve dönüş değerlerini değerlendirmek için kullanır. Aslında gözetleme dediğimiz işlem burada okunan değerlerin kendisidir. Buradaki sistem çağrıları, parametreleri ve geri dönüş değeri sürecin sistemde yaptığı işlemi temsil etmektedir.

Gerekli değerler okunduktan sonra gözetleyici durmuş olan sürecin çalışmasını yeniden başlatır.

```
ioctl (procfd, PIOCRUN, 0);
```

Süreç ilgilenilen yeni bir sistem çağrısı gerçekleştirene kadar çalışmasına devam eder. Gözetlenen süreç sonlanır veya öldürülürse, gözetleyici bunu algılar ve gözetleme işlemini durdurur.

Süreçleri gözetlemek, süreçlerle ilgili gereken bilgiyi tutmak ve eşzamanh süreçlerle ilgilenebilmek için gözetleyici bir süreç tablosu kullanır. Aşağıdaki yapı süreç tablosunu ifade etmektedir.

```
struct processTable {
    int cmd_time;
```

```
    pid_t PID;
    int procfd;
    prstatus_t *pstatus;
    prstatus_t pstatus_buf;
} procT[MAXPROCS]
```

Bir grup süreç içerisinde herhangi bir durma ya da sonlanma olayının meydana gelmesini gözetleyici, *poll* sistem çağrısı sayesinde algılar.

2.2.3. Gözetleme İçin İhtiyaç Duyulacak Fonksiyonların ve Sınıfların Belirlenmesi

JUSİ çatısı üç sınıftan meydana getirilmiştir: Trace, Handler, Process Sınıfları. Bu sınıflar gözetleme faaliyetini yürütmede kullanılacak JNI fonksiyonları içerirler. JNI fonksiyonları ile C++ çevresine ait işlevler çağrılarak süreçlerin durum bilgilerini içeren veri yapılarına erişilir. Bununla beraber izleme işlemi için gerekli setlemeler ve bildirimler yapılabilir veya izleme esnasında süreçlerin aktivite bilgileri Java ortamına aktarılabilir.

Trace Sınıfı

```
private native int executeProgram(String cmd);
private native int initEntryTraceFlags(int n, int s_entry[]);
private native int initExitTraceFlags(int n, int s_exit[]);
private native int handleStoppedProcess (int pindex);
private native int waitForSomeoneToStop ();
private native int waitForProcessToStop (int n);
private native int restartProcess (int procfd);
private native int releaseProcess (intlprocfd);
private native int shutdownTrace (int procfd);
private native int attachToProcess(int pid);
```

Process Sınıfı

```
private native int getNumOfProcs ();
private native int getProcfd (int pindex);
```

private native void closeProcess (int pindex);
private native int getProcWhy (int pindex);
private native int getPid (int pindex);
private native int getSysnum (int pindex);

Handler Sınıfı
private native string getSyscallParameters(pindex)
private native string getSyscallName (int pindex);
private native string getSyscallReturnValue (int pindex);
private native int changeSysCallArg(int n, int arg, String str);

Trace sınıfı içinde tanımlanan metotlar yardımıyla süreçleri gözetleme faaliyetleri yönetilir. Gözetleme faaliyetlerinin sadece kullanıcının kendisine ait süreçler üzerinde yapılabileceğine dikkat edilmelidir. UNIX servis programlarını ve diğer kullanıcı aktivitelerini koşan süreçler sadece sistem yöneticisi tarafından gözetlenebilir. Metotlardan *executeProgram* aktiviteleri izlemek istenen UNIX programının icrasını başlatmak için kullanılır. Gözetleme başlamadan önce *initEntryTraceFlags* ve *initExitTraceFlags* metotları ile izlenmek istenen sürecin yapacağı sistem çağrılarının hangilerinin çekirdek moduna girişte ve hangilerinin çıkışta yakalanmak istendiği işletim sistemine bildirilir. *attachToProcess* metodu halihazırda koşmakta olan UNIX süreçlerinin gözetimlerini gerçekleştirir. *waitForSomeoneToStop* durdurulan sürecin indeks numarasını belirlemek için kullanılır. w*aitForProcessToStop* duran sürecin hangi sebepten durduğunu belirlemek için kullanılır. *handleStoppedProcess* metodu durdurulan sürecin özel bir işlem yapıp yapmadığını kontrol etmek için kullanılır. *restartProcess* metodu durdurulan sürecin yeniden başlatılması için kullanılır. *releaseProcess* metodu sürecin izlenme işlemini sona erdirmek zaman. *shutdownTrace* metodu ise takip etme işlemini sonlandırmak için kullanılır.

Process sınıfı, izlenen süreçler arasında durmuş olanlarının yapmakta oldukları aktiviteleri öğrenmek için kullanılan metotları içerir. Örneğin, bu sınıf içinde, hangi sistem çağrısının durduğu *getSysnum* metodu ile, bu durmanın çekirdeğe girişte mi çıkışta mı gerçekleştiği *getProcWhy* metodu ile belirlenir. Süreç bir fork çağrısı yaparken durmuş ise bu çağrının geri dönüş değeri (child sürecin PID'si) *getPid* metodu ile alınır. *closeProcess* metodu ile takip edilmesi vazgeçilen sürecin süreç tablosundan çıkarılması için kullanılır.

getProcfd metodu süreçleri temsil eden dosya tanımlayıcısını almak için kullanılır. *getNumOfProcs* metodu takip edilen süreç sayısını öğrenmek için kullanılır.

Handler sınıfı ise sistem çağrılarının ismini öğrenmek için *getSyscallName,* argümanlarını okumak için *getSyscallParameters,* fork dışında kalan çağrıların geri dönüş değerlerini almak için *getSyscallReturnValue* ve durdurulan sistem çağrısının parametrelerini değiştirmek için *changeSysCallArg* metotlarından oluşur.

2.2.4. Java İle Yerel Fonksiyonlarla İletişimi Sağlayacak Olan JNI Fonksiyonlarının Gerçeklenmesi ve Örnek Bir Kullanım

JUSİ çatısını oluşturan sınıfların içerdiği metotların hem JNI hem de C++ karşılıklarının gerçeklenmesi gerekir. Burada yazılan JNI fonksiyonları C++ karşılıklarının çağırılması için kullanılmaktadır. JNI fonksiyonların gerçeklenmesinde dikkat edilmesi gereken kısım veri tipleri dönüşümüdür. Örnek olması açısından iki fonksiyonun gerçeklemesini aşağıda gösterilmiştir.

```
int initEntryTraceFlagsC(int n, int *s_entry)
 {
      sysset_t SysCalls;
      premptyset(&SysCalls);
      for(int i = 0; s_entry[i]; i++)
            praddset (&SysCalls, s_entry[i]);
      int rc=ioctl(p->getProcfd(n),PIOCSENTRY, &SysCalls);
      long flags = PR_FORK;
      int rc2 = ioctl(p->getProcfd(n), PIOCSET, &flags);
      return (rc || rc2);
}

int initExitTraceFlagsC(int n, int *s_exit)
{
      sysset_t SysCalls;
      premptyset(&SysCalls);
      for(int i = 0; s_exit[i]; i++)
      praddset (&SysCalls, s_exit[i]);
```

```
    return (ioctl(p->getProcfd(n), PIOCSEXIT, &SysCalls));
}
```

Seçilen *initEntryTraceFlagsC ve initExitTraceFlagsC* fonksiyonlarının C++ gerçeklemesi yukarıda görülmektedir. Bu fonksiyonlar izlenilen sürecin çekirdeğe giriş ve çıkışta hangi sistem çağrıları için durdurulacağı belirtilmiştir.

```
JNIEXPORT jint JNICALL Java_trace_initExitTraceFlags (JNIEnv *env, jclass obj,
jint index, jintArray s_entry)
{
    jint *s_ent;
    jint res;
    if ((s_ent = (*env)->GetIntArrayElements(env, s_entry, NULL);) == NULL)
        return -1;
    JTHROW_neg1(res = initEntryTraceFlagsC(index, s_ent));
    (*env)->ReleaseIntArrayElements(env, s_exit, s_ext,JNI_ABORT);
    return res;
 }
```

```
JNIEXPORT jint JNICALL Java_trace_initExitTraceFlags(JNIEnv *env, jclass obj,
jint index, jintArray s_exit)
{
    jint *s_ext;
    jint res;
    if ((s_ext = (*env)->GetIntArrayElements(env, s_exit, NULL);) == NULL)
        return -1;
    JTHROW_neg1(res = initExitTraceFlagsC(index, s_ext));
    (*env)->ReleaseIntArrayElements(env, s_exit, s_ext,JNI_ABORT);
    return res;
 }
```

initEntryTraceFlags ve *initExitTraceFlags* fonksiyonlarını *initEntryTraceFlagsC ve initExitTraceFlagsC* fonksiyonlarının Java içerisinde kullanılabilmesi için

gerçekleştirilmiştir. Görüldüğü gibi JNI fonksiyonları içerisinden C++ fonksiyonları çağırılmaktadır. Diğer metotların JNI ve C++ karşılıkları benzer olarak gerçeklenmiştir.

Süreçleri bu metotlarla izlerken ilgilenilmesi gereken önemli konulardan biri de 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 C++ dilinde tanımlanan aşağıdaki süreç tablosundan (*procT*) yararlanılır.

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

Pollfd yapısı, süreç eylemlerinin etkin kontrolünü eşzamanlı olarak gerçekleştirmek için *poll* çağrısı tarafından kullanılır. Bu şekilde tek bir program ile çalışmakta olan birçok sürecin gözetleme faaliyeti gerçekleştirilebilir. Bu yapı içerisinde sistemin kendi veri yapıları da kullanıldığı için Java tarafında oluşturulmamıştır.

Tanımlanan metotlarla basit bir uygulama yazalım. Bu uygulamada kullanıcının ismini girdiği programlar Java içerisinden çalıştırılır ve programlarla ilgilenen süreçlerin yaptığı sistem çağrıları izlenir. Uygulamanın kaynak kodu incelendiğinde programcının süreç gözetlemeyi istediği gibi yapılandırabileceği görülmektedir. Bu yapılandırmalar işletim sistemi çekirdeğine ait yapılarla doğrudan ilgilenilmeden gerçekleştirilebilmektedir.

```
public class Tracer {
    private Trace Tr, Process Ps, Handler Hnd;
    private int s_entry[254],s_exit[254];
    private int pindex, res, fd, pid, status;
    private String sysname;
    private void setAllSysCalls () {
```

```
        int i;
        for (i=2; i< N_SYSCALLS; i++) {
            s_entry [i-2]=i; s_exit [i-2]=i;
        }
        s_entry [i-2]=1; s_entry [i-1]=0; s_exit [i-2]=0;
}
private void handleProcess () {
        fd=Ps.getProcfd (pindex);
        pid=Ps. getPid (pindex);
        sysname = Hnd. getSyscallName(pindex);
        if (sysname !="") {
            if (Ps.getProcPrWhy(pindex)==PR_SYSENTRY) {
                System.out.println(sysname + "(");
                System.out.println(
                Hnd.getSyscallParameters(pindex)+"(");
            } //end if
            else
            System.out.println(Hnd.getSyscallReturnValue (pindex) + "(");
        } //end if
}
public static void main(String args[]) throws Exception {
        BufferedReader jinput = new BufferedReader (
        new InputStreamReader (System.in) );
        String program;
        Tracer tracer=new Tracer();
        Trace Tr = new Trace();
        Process Ps = new Process();
        Handler Hnd = new Handler();
        tracer.setAllcalls();
        while (true) {
            System.out.print("Enter Program Name>> ");
            program = jinput.readLine();
            if (program.length()!=0) {
```

```
                pindex = Tr. executeProgram(program);
                if (pindex < 0)
                    printf("Could not attach to child.\n");
                if (Tr. initEntryTraceFlags (pindex, s_entry)< 0)
                    printf("Could not set entry trace options.\n");
                if (Tr. initExitTraceFlags (pindex, s_exit)< 0)
                    printf("Could not set exit trace options.\n");
                while ( Ps.getNumOfProcs() > 0 ) {
                    pindex = Tr.waitForSomeoneToStop();
                    res = Tr.waitForProcessToStop(pindex);
                    switch (res) {
                        case WPS_STOPPED:
                            tracer.handleprocess();
                            status = Tr.handleStoppedProcess(pindex);
                            if (status == 0)
                                Tr.restartProcess(Ps.getProcfd(pindex))
                            Ps.closeProcess(pindex);
                            break;
                        case WPS_DIED:
                            Ps.closeProcess(pindex);
                            break;
                        case -1:
                            System.out.println("ERR: WPSTOP failed");
                            break;
                    } //end switch
                } //end while
            }//if
        } //while
        System.exit(0);
    }
}
```

2.2.5. JUSİ' nin Sistem Performansına Etkisi

JUSİ' nin sisteme getirdiği yükü belirlemek için, gözetleme altında sistem programlarının çalışma sürelerinin ölçülmesi gerekmektedir. Bu ölçüm işlemi gözetlemeli ve gözetlemesiz olarak çalışma sürelerinin farkı olarak düşünülebilir. Süreç gözetleme işlemi süreçlerin mod değişimi anında durdurulup, sürecin veri alanından istenilen verilerin okunmasına dayandırıldığından icra süresinin artmasına neden olmaktadır. Çünkü gözetlemeli olarak programın icrasında gözetlenen sürecin icrasından önce gözetleyen süreç, gözetlenen süreci durdurup, istediği verileri aldığından gözetlenen sürecin çekirdekteki sırasını almış olur. Bu sürede gözetlenen süreç gözetleyen süreci tekrar başlatmasını bekler. Sürecin gerçekleştirdiği sistem çağrılarının fazla olması durumunda veya sürecin izlenilen sistem çağrılarının fazla olması durumunda performans kaybı daha fazla olacaktır. Bu durum, her bir sistem çağrısı için gözetlenen süreç ile gözetleyen süreç arasında içerik değişimleri ortaya çıkmasından dolayı kaçınılmazdır.

JNI kullanımından kaynaklanan performans kayıpları da vardır. Java ile C++ arasında fonksiyon çağrımlarında, veri tiplerinin paketlenmesi ve açılması ve fonksiyonlar arası iletişimden kaynaklanan zaman kayıpları da dikkate değerdir.

Sistemin performansının ölçülmesi amacı ile Tablo 2.1'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 süreler için bağlantı süreleri dikkate alınmayıp, süreçlerin harcadığı işlemci zamanları kaydedilmiştir.

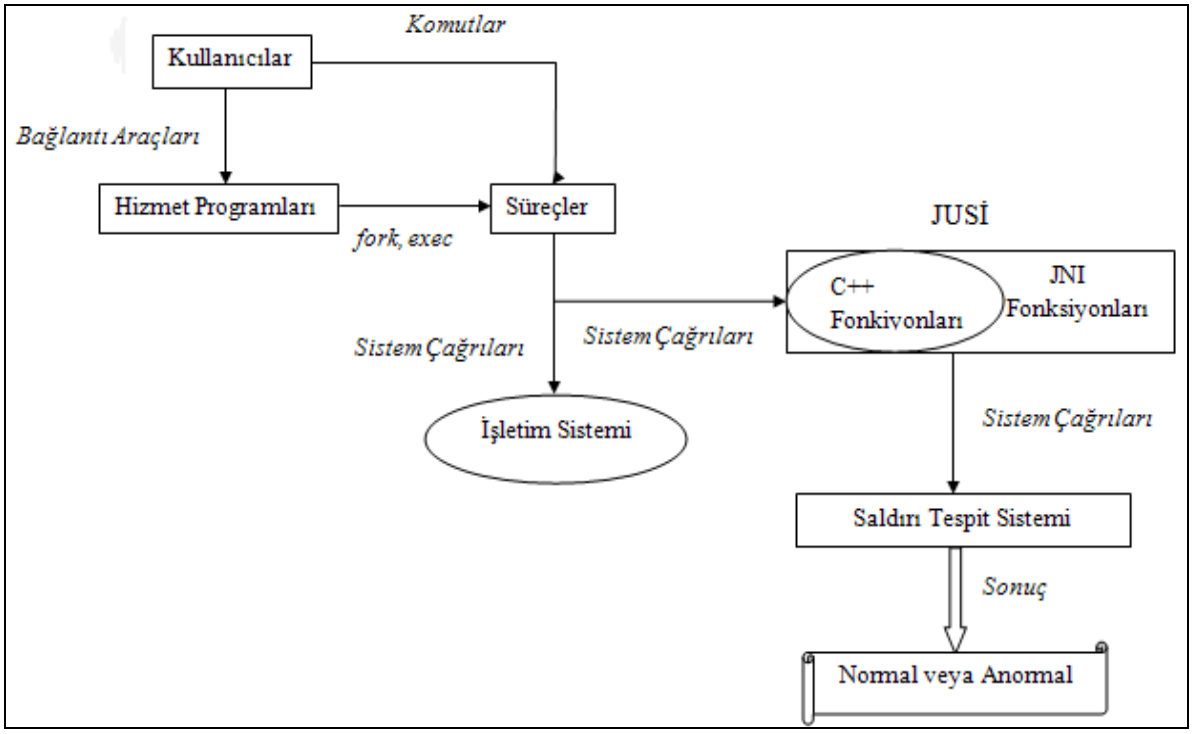
Tablo 2.1. Süreç gözetlemenin program icra süresine etkisi

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

2.3. Saldırıları Tespit Sisteminin Tasarımı ve Gerçeklenmesi

2.3.1. Saldırı Tespit Sisteminde JUSİ 'nin Rolü

Geliştirilen JUSİ çatısı, Solaris sisteminde çalışan süreçlerin yaptığı ve saldırı tespit sistemlerinin bir parçası olan sistem çağrılarının Java ortamına getirilmesi için kullanılır. K en yakın komşuluk sistemi Java dili kullanılarak oluşturulmuştur. Offline olan veri setleri kullanılarak eğitim aşaması tamamlanmıştır. Bu aşamada JUSI kullanılarak, Şekil 2.2' de gösterildiği gibi Solaris ile etkileşimli olarak, çalışan süreçlerin yapmış oldukları sistem çağrıları yakalanmış ve Java ortamına aktarılmıştır. Bu sistem çağrıları üzerinde yukarıda anlatılmış olan işlemler uygulanarak çalışmakta olan sürecin davranışı belirlenmiştir. Bu şekilde, normal veya anormal olarak etiketlenmiştir.



Şekil 2.2. JUSİ çatısının saldırı tespit sisteminde üslendiği rol

2.3.2. Sistem Çağrılarının En Yakın Komşuluk Yöntemi Uygulanacak Şekilde Yorumlanması

Bu çalışmada sistem çağrıları değişik bir şekilde değerlendirilmiştir. Sistem çağrılarının belli bir sıra ile gelmesine rağmen programların davranışlarının sınıflandırılmasında bu sıraya bakmaksızın frekansları kullanılmaktadır. Metinleri sınıflandırmak için kullanılan K en yakın komşuluk yöntemini programların davranışlarının sınıflandırılmasında da kullanmak amacıyla programların yapmış olduğu sistem çağrılarını değişik şekilde yorumlamak gereklidir. Süreçlerin yaptığı sistem çağrılarının her birisi bir kelime ve toplamı ise metin olarak alındığında metin sınıflandırma işleminde kullanabileceğimiz bir yapıya ulaşılmış olacaktır.

Tablo 2.2. En yakın komşuluk yönteminin sınıflandırma yönteminde metin sınıflandırma ile saldırı tespit sisteminin benzerliği

Terimler	**Metin Sınıflandırma**	**Saldırı Tespit**
N	Toplam Doküman Sayısı	Toplam Süreç Sayısı
M	Toplam Kelime Sayısı	Toplam Sistem Çağrısı Sayısı
n_i	i.Kelimenin Tekrarlanma Miktarı	i.Sistem Çağrısının Tekrarlanma Miktarı
f_{ij}	j Dokümanındaki i. Kelimenin Frekansı	j Sürecindeki i. Sistem Çağrısının Frekansı
D_j	j. Eğitim Dokümanı	j. Eğitim Süreci
X	Test Dokümanı	Test Süreci

Tablo 2.2'de görüldüğü gibi metinlerin sınıflandırılması ile saldırı tespit için sistem çağrıları dizisinin sınıflandırması birbirine çok yakın bir ilişki içerisindedir. Bu nedenden dolay K en yakın komşuluk algoritmasının uygulanmasına çok elverişlidir.

Metinlerin sınıflandırmasına benzer bir şekilde her sürecin yaptığı sistem çağrılarını bir vektör olarak düşünürsek frekansa göre ağırlıklandırma (1.1) denklemine göre ve tf-idf ağırlıklandırması (1.2) denklemine göre yapılarak vektör kayıtlarının değerlerini hesaplayabiliriz. Yeni bir sürecin normal veya saldırgan olduğu belirlenmek istendiği zaman k en yakın komşuluk algoritmasına göre herbir eğitim sürecine olan uzaklığı (1.3)

denklemine göre hesaplanır. Kendisine en yakın K tane komşu vektöre uzaklığı hesaplanır. Bu uzaklık vektörlerinin ortalaması belirli bir eşik değerinden büyük ise normal, küçük ise anormal olduğu varsayılır.

2.3.3. Kullanılacak Veri Kümesi

Bu çalışmada, K en yakın komşuluk sınıflandırıcı için eğitim verileri olarak DARPA 1998 veri setlerinden Temel Güvenlik Modülü (BSM) denetleme verileri kullanılmıştır. Bu veri setleri ayrıntılı olarak genel bilgiler kısmında anlatılmaktadır. BSM verileri kurban olarak seçilen Solaris makinesinde çalışan süreçlerin yaptığı sistem çağrılarından oluşmaktadır. Bu veriler yapılan sistem çağrılarının isimlerini, çalışan dosyaların özelliklerini, tam yolunu, parametrelerini, geri dönüş değerini, oturum numarasını ve buna benzer birçok özellikleri içermektedir. Bu çalışmada sadece sistem çağrılarının isimleri kullanılmaktadır. DARPA veri setleri her oturumda birden fazla sürecin yaptığı sistem çağrılarının programlı bir şekilde sistemde sıralı olarak kaydedilmesi ile oluşturulmuştur. Tablo 2.3' de örnek olarak bir sürecin yaptığı sistem çağrıları verilmiştir.

Tablo 2.3. Örnek *sh* sürecinin yapmış olduğu sistem çağrıları

execve	*close*	*mmap*	*close*	*munmap*	*ioctl*
open	*open*	*mmap*	*open*	*setpgrp*	*ioctl*
mmap	*mmap*	*munmap*	*mmap*	*setpgrp*	*close*
open	*mmap*	*mmap*	*mmap*	*ioctl*	*close*
mmap	*munmap*	*mmap*	*munmap*	*ioctl*	*close*
mmap	*mmap*	*close*	*mmap*	*stat*	*close*
munmap	*close*	*open*	*close*	*access*	*close*
mmap	*open*	*mmap*	*close*	*fork*	*exit*

Çalışmamızda bu gelen sistem çağrılarının her birisinin kaç defa geldiği ile ilgilenilmektedir. Bu değerler frekans ve tf-idf yöntemleri ile ağırlıklandırılarak vektöre dönüştürülür.

2.3.4. Anormallik Tespiti

Gerçekleştirilecek saldırı tespit sistemi anormallik tespitine dayalı olduğu için programların normal davranışlarının belirlenmesi gerekmektedir. Bütün olası normal davranışların belirlenmesi kolay değildir. Ancak çok büyük veri setleri kullanımı ile belirlenen davranışlar çok daha etkili olmaktadır. Bu ise programların davranışlarının belirlenmesinde çok fazla işlem gerektirir. Davranışların belirlenmesi için DARPA veri setleri içerisinde 12 günlük saldırısız olan veriler kullanılmıştır. Bu süre içerisinde 3668 ayrı sürecin yaptığı sistem çağrıları izlenilmiştir. Ayrıca bu izlenilen süreçlerin yaptığı bütün sistem çağrılarının izlenmesine ihtiyaç yoktur. Sadece 50 sistem çağrısı kullanmak yeterli olur. İzlenilen sistem çağrıları Tablo 2.4'de gösterilmiştir.

Tablo 2.4. Saldırı tespiti için kullanılacak sistem çağrıları

access	*audit*	*auditon*	*chdir*	*chmod*	*chown*	*close*	*creat*
execve	*exit*	*fchdir*	*fchown*	*fcntl*	*fork*	*fork1*	*getaudit*
getmsg	*ioctl*	*kill*	*link*	*login*	*logout*	*lstat*	*memcntl*
mkdir	*mmap*	*munmap*	*nice*	*open*	*pathdonf*	*pipe*	*putmsg*
readlink	*rename*	*rmdir*	*setaudit*	*setegid*	*seteuid*	*setgid*	*setgroups*
setpgrp	*setrlimit*	*setuid*	*stat*	*statvfs*	*su*	*sysinfo*	*unlink*
utime	*vfork*						

Sistemdeki süreçlerin izlenmesi sonucu elde edilen sistem çağrılarının her birisinden ne kadar geldiği hesaplanarak ağırlıklandırma fonksiyonlarına göre vektörler oluşturulur. Bütün süreçlerin kendilerine ait vektörleri oluşturması eğitim aşamasını meydana getirir. Eğitim aşamasında iki değişik ağırlıklandırma fonksiyonu kullanılır. Bu fonksiyonlara göre süreçlerin yaptığı sistem çağrılarının ağırlıklandırılması vektörlerin oluşturulması anlamına gelir.

Süreçlerin yaptığı sistem çağrılarını bir vektör olarak düşünürsek frekansa göre ağırlıklandırma (1.1) denklemine göre eğitim vektörleri oluşturulur. Bu eğitim vektörlerine bir örnek olarak Tablo 2.5' deki vektör verilebilir. Bu vektör izlenilen programın Tablo 2.3 de olan sistem çağrılarının kaç defa geldiği beliklenilerek oluşturulmuştur.

Tablo 2.5. Herhangi bir programın yapmış olduğu sistem çağrılarının gelme miktarları

2	1	0	0	0	1	0	0	1	4	18	0	0	6	0	0	0
6	0	0	0	0	0	0	0	7	0	7	0	48	0	0	0	6
0	0	0	0	324	0	0	6	0	0	1	0	0	0	0	0	

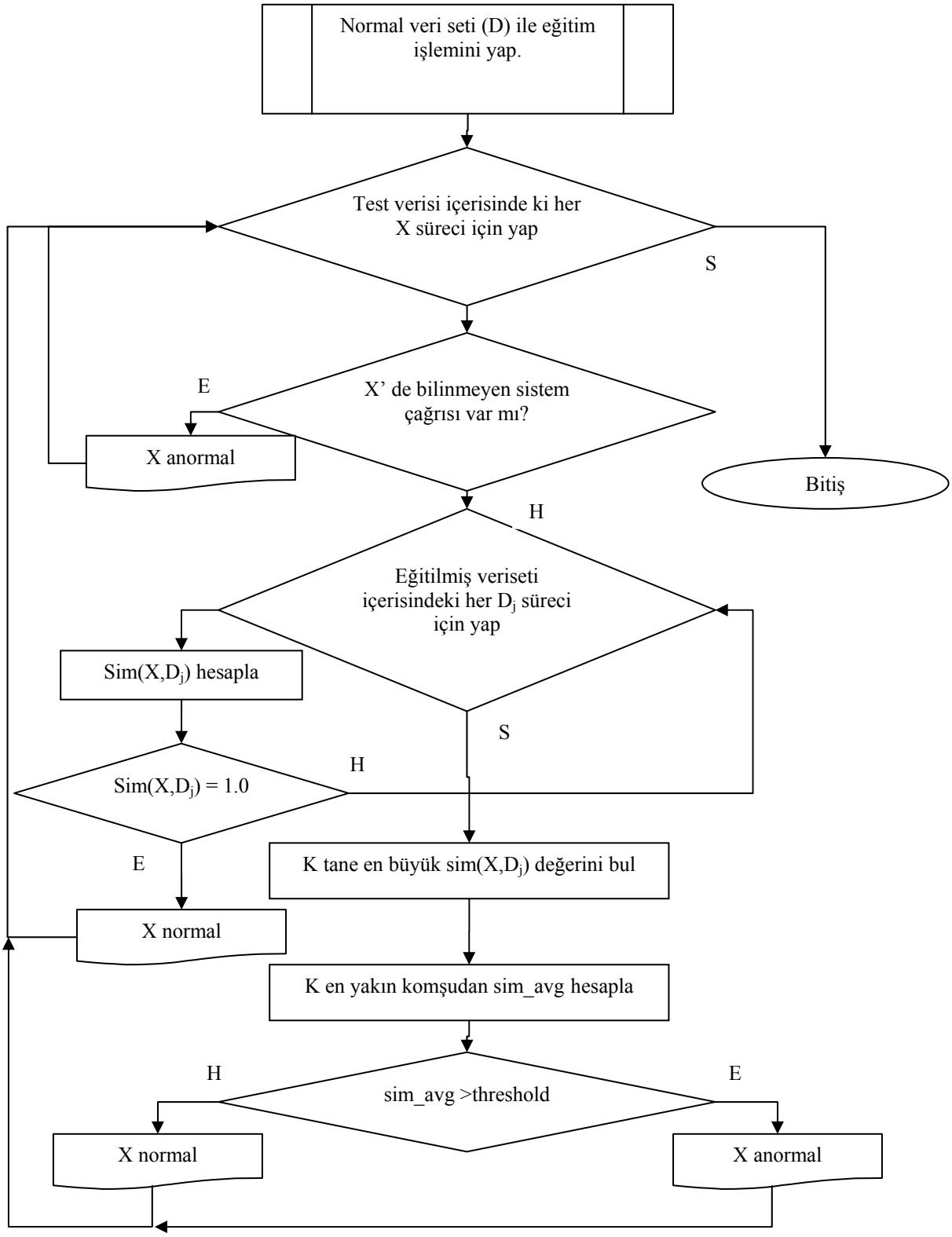
Ağırlıklandırma işlemini tf-idf ağırlıklandırması (1.2) denklemine göre yapılarak da vektörler oluşturulur. (1.2) denkleminde görülebileceği gibi her bir değerin ağırlıklandırma işlemi için bütün eğitim verilerinin frekans değerlerinin bilinmesi gerekir. Bu yöntemle oluşturulmuş vektörlere bir örnek Tablo 2.6' de gösterilmektedir.

Tablo 2.6. Herhangi bir programın tf-idf ağırlıklandırma fonksiyonuna göre hesaplanmış değerleri

0.004	0	0	0	0	0	0	0	0	0.017	0.158	0	0	0.033	0	0	0
0.033	0	0	0	0	0	0	0	0.041	0	0.041	0	0.566	0	0	0	0.033
0	0	0	0	5.703	0	0	0.033	0	0	0	0	0	0	0	0	

Eğitim aşamasının tamamlanması ile birlikte online olarak sistem izlenir. Süreçlerin yapmış oldukları sistem çağrıları daha önceki eğitim aşamasındaki verilere uygulanan ağırlıklandırma işlemine tabi tutularak kendi ağırlıklandırılmış vektörü elde edilir. Bu vektörün her bir eğitim vektörüne uzaklığı (1.3) denklemine göre tespit edilir. Belirlenen k değerine göre k tane komşu vektöre benzerliğinin ortalaması hesaplanır. Bu ortalama benzerlik belirli bir eşik değerinden fazla olması durumunda normal, az olma durumunda ise anormal olarak kabul edilir. Benzerlik (uzaklık) belirleme işleminde 1 sonucu elde edilirse k değere bakmadan kesin benzerlik bulunduğundan normal olarak kabul edilir.

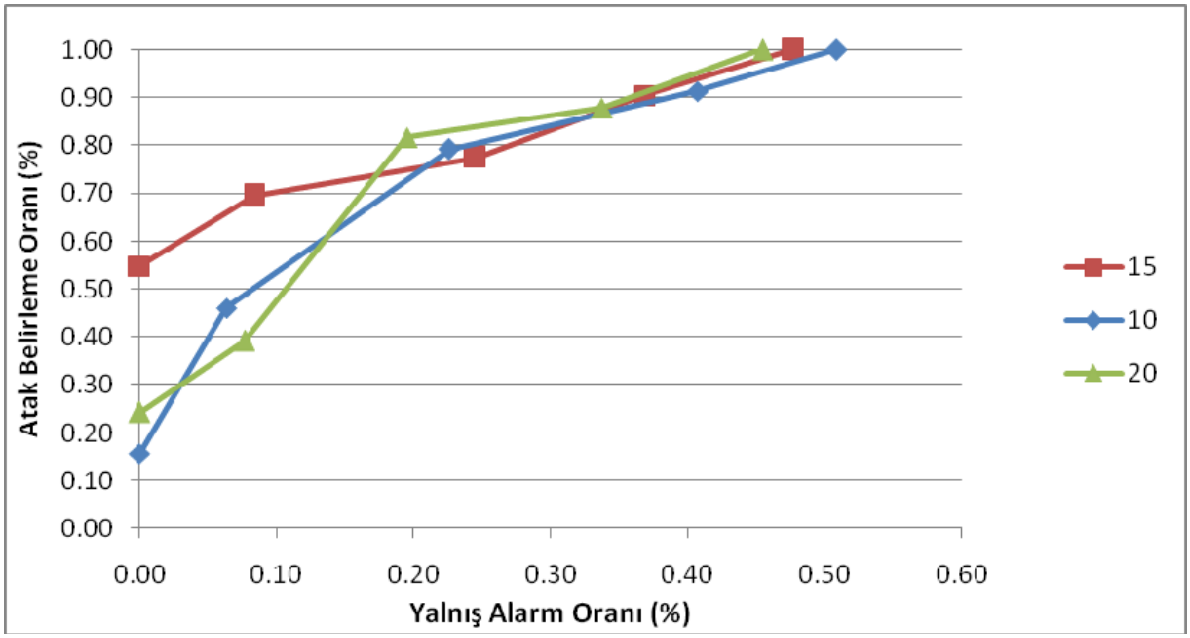
Çalışmamızda kullanılacak K en yakın komşu algoritmasının anormallik tespiti için kullanımına ait algoritma Şekil 2.3'de verilmiştir. Ayrıca yazılan programdaki bazı kodlar Ek 1' de mevcuttur. Özellikler hesaplamalar ve karşılaştırma işlemlerinin nasıl yapıldığı Ek 1' de görülebilir.



Şekil 2.3. Anormallik tespitinde K en yakın komşuluk algoritması akış diyagramı

K en yakın komşuluk sınıflandırma sisteminin performansı k değerinin seçimine bağlıdır. Burada k değeri test edilen sürecin kaç komşu değerinin dikkate alınacağını belirler. Genel olarak en uygun k değeri testler sonucu elde edilebilir. Üç faklı k değeri için sistemin çalışması test edilmiştir. Anormallik tespitine dayalı saldırı tespit sistemlerinde en önemli parametre olan yanlış alarm yani saldırı olmadığı halde saldırı olduğunu tespit etme oranı dikkate alınmıştır.

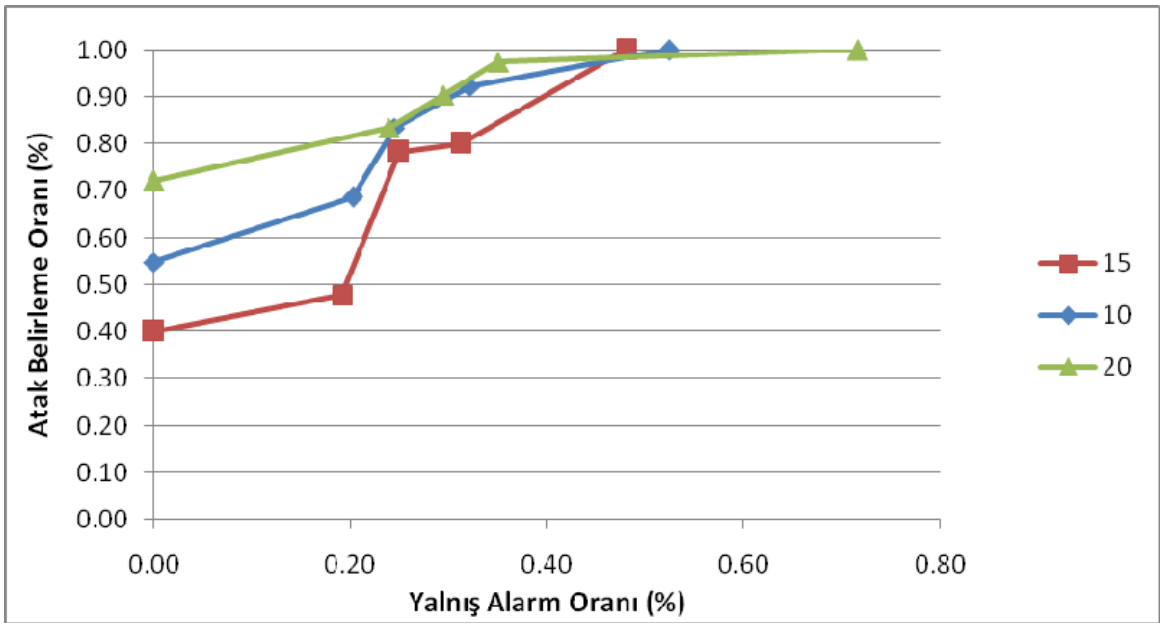
K değeri olarak 10,15,20 alınarak, tf-idf ağırlıklandırma fonksiyonu ile sistem test işleminden geçirilmiştir. Bu k değerleri içerisinde en iyi sonuca Şekil 2.4'de görülebileceği gibi 15 değeri ile ulaşılmıştır. Test veri seti içerisindeki yanlış alarm vermeksizin 21 ataklı oturum içerisinden 115 süreçten 63 tanesini yanlış alarm vermeden tespit edebilmiştir. Ayrıca test verilerindeki tüm atakları 0,92 eşik değeri tespit edilmiş. Ancak izlenilen 38604 normal süreç içerisinde sadece 1843 yanlış alarm vermiştir.



Şekil 2.4. tf-idf ağırlıklandırma fonksiyonuna ile K=10,15,20 değerleri için yanlış alarm ve atak tespit oranları

K değeri olarak 10,15,20 alınarak, frekans ağırlıklandırma fonksiyonu ile sistem test işleminden geçirilmiştir. Bu k değerleri içerisinde atakları yakalama konusunda en iyi sonuca Şekil 2.5'de görülebileceği gibi 20 değeri ile ulaşılmıştır. Ancak bu değer ile tüm atakları yakalamak istenmesi durumunda yanlış alarm verme oranı oldukça

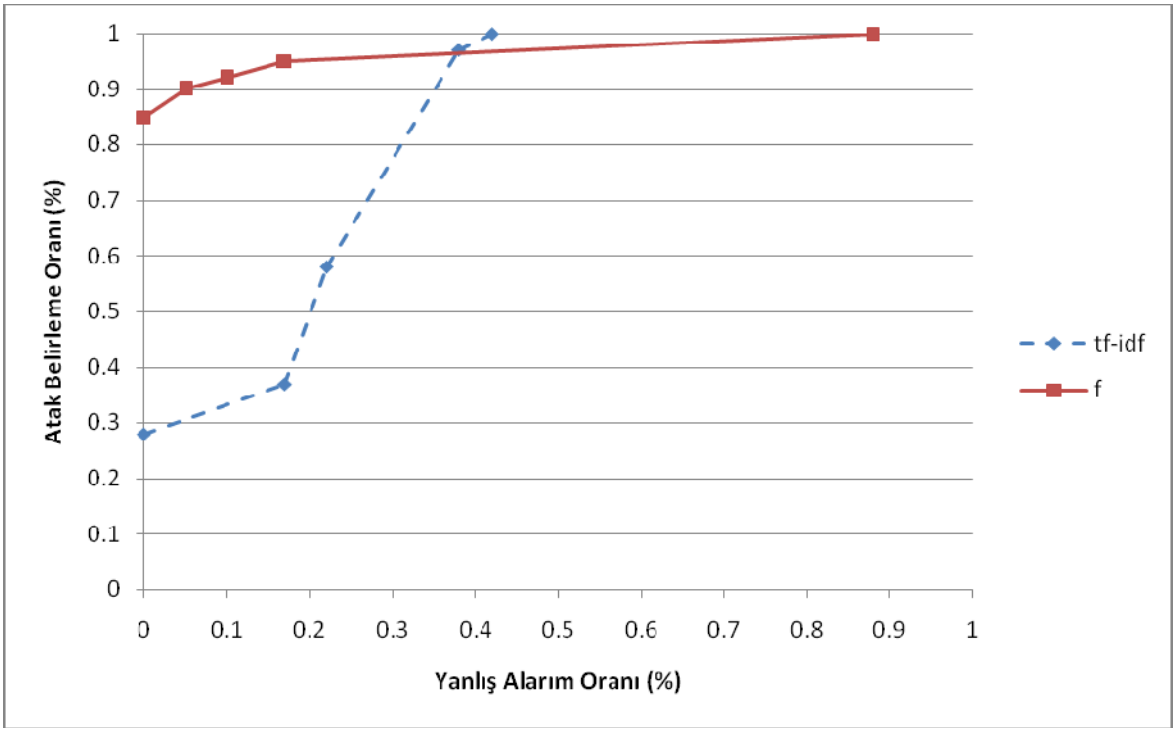
yükselmektedir. Test veri seti içerisindeki yalnış alarm vermeksizin 21 ataklı oturum içerisinden 115 süreçten 83 tanesini yanlış alarm vermeden tespit edebilmiştir. Ayrıca test verilerindeki tüm atakları 0,92 eşik değeri tespit edilmiş. Ancak izlenilen 38604 normal süreç içerisinde sadece 2765 yanlış alarm vermiştir.



Şekil 2.5. Frekans ağılıklandırma fonksiyonuna ile K=10,15,20 değerleri için yanlış alarm ve atak tespit oranları

Şekil 2.6'de bu iki ağırlıklandırma fonksiyonunun iyi olduğu durumlar gösterilmektedir. Frekans ağırlıklandırma için 20, tf-idf ağırlıklandırma için 10 eşik değeri ile sistemin performansını gösterecek eğriler grafiğe eklenmiştir. Bu iki farklı ağırlıklandırma fonksiyonu kıyaslandığında frekans ağırlıklandırma fonksiyonu atak tespitinde, tf-idf ağırlıklandırma fonksiyonu yanlış alarm vermede istenilen seviyededir. Bu iki fonksiyonun kendine göre avantaj ve dezavantajları vardır.

Bu grafiklerde en ilginç olan kısımlar eğrilerin X=0 ve Y=1 olduğu noktalardır. X=0 olduğu nokta sistemde hiçbir yanlış alarm verilebilecek şekilde eşik değeri seçildiği zaman atak belirleme oranını belirtir. Y=1 olduğu nokta ise yapılan bütün atakların belirlenebileceği şekilde eşik değerinin belirlenmesi durumunda gerçekleşebilecek yanlış alarm oranını belirtmektedir. Eğrinin karakteristiğinin anlaşılabilmesi için bu iki nokta arasında değişik eşik değerleri belirlenerek eğriler oluşturulur.



Şekil 2.6. tf-idf ağırlıklandırma (k=10) ve frekans ağırlıklandırma (k=20) için yanlış alarm ve atak tespit oranları

Bu çalışmanın temel amacı saldırı tespit sistemleri olmayıp Java programlama dili ile süreçlerin izlenebilmesini sağlayan JUSİ çatısının oluşturulması olduğundan, burada saldırı tespitlerinin k en yakın komşuluk algoritması tespiti Java programlama dili ile gerçeklenmiştir.

3. TARTIŞMA

Geliştirilen JUSİ çatısı, JNI ile sistemin yerel fonksiyonları kullanılarak geliştirildiği için Java'nın en önemli özelliklerinden birisi olan platform bağımsızlık özelliğini ortadan kaldırır. Bu nedenle bu çatının farklı sistemlerde kullanılması mümkün değildir. Bununla beraber kodlama yapılacağı esnada genel bilgiler kısmında bahsedilen JNI kullanımının dezavantajları da göz ardı edilmemelidir.

Java ile yerel fonksiyonların beraber kullanılması bu çalışmada Java'nın sistemin çekirdek veri yapılarına erişimi içindir. Bununla beraber böyle bir kullanım performans açısından oldukça kötü sonuçlar verir. Çünkü her an Java ile JNI fonksiyonları arasında yapılan fonksiyon çağırma işlemi ve geri dönüş değerinin beklenmesi, bu çatı kullanılarak geliştirilecek programların yavaş bir şekilde çalışmasına neden olacaktır.

Sistemde çalışan programların takip edilmek istenmesi durumunda bilinmesi gereken en önemli noktalardan birisi de her kullanıcı kendine ait programları izleyebilmesidir. Sistemde çalışan hizmet programlarının izlenmesini ancak sistem yöneticisi gerçekleştirebilir. Bu gerçekleştirilen saldırı tespit sistemini sadece sistem yöneticisinin kontrolünde çalışabilir yapar.. Dolayısıyla, saldırı tespit sistemi gibi servis şeklinde çalışacak programlar dikkatli bir şekilde yazılmalıdır. Çünkü JNI kullanımından dolayı Java dilinin güvenlik mekanizmalarının bir kısmı devreden çıkmaktadır.

Bu tezde saldırı tespit sistemleri üzerinde detaylı bir çalışma yapılmamıştır. Mevcut bir sistem, yani K en yakın komşuluk sınıflandırması Java ortamında gerçekleştirilmeye çalışılmıştır. JUSİ çatısı ile çalışan süreçlerin yapmış oldukları sistem çağrıları yakalanıp saldırı sisteminde kullanılmıştır.

4. SONUÇLAR

Bu çalışmamızda C/C++ ile yapılabilen süreç izleme, yaptığı sistem çağrılarının kaydedilmesi işlemini daha basit bir şekilde Java programlama dili ile gerçekleştirilebilecek bir programlama çatısı oluşturulmuştur. Bu amaçla Java dili ile C/C++ dilleri beraber kullanılarak UNIX işletim sisteminin çekirdek veri yapılarına erişimi mümkün hale getirilmiştir. Bu iki dilin etkileşimi sağlamak için JNI teknolojisinden yararlanılmıştır.

Çalışmanın son kısmında oluşturulan çatı kullanılarak basit bir saldırı tespit sistemi geliştirilmiştir. Bu sistem, metin sınıflandırma algoritması ile süreçlerin yaptığı sistem çağrılarına göre bu süreçlerin normal veya anormal olarak çalıştığını belirlemektedir. Oluşturulan çatı bu saldırı tespit sisteminin işletim sisteminden yapılan sistem çağrılarını alabilmesi için kullanılmıştır.

5. ÖNERİLER

Java ile C/C++ fonksiyonlarının birlikte kullanımı performans olarak çok iyi netice vermemektedir. Çünkü JNI kullanılarak yapılan diller arası iletişim fonksiyon çağırımları ile gerçekleştirilir. Çok fazla fonksiyon çağırma işlermi, verilerin paketlenmesi ve veritipleri arası dönüşümlerden dolayı çok fazla zaman harcamaktadır. Bu nedenle JUSİ çatısına, bloklar halinde sistem çağrılarının Java ortamına aktarabilecek fonksiyonlar eklenmelidir.

Ayrıca JUSİ çatısını kullanan programcıların mümkün olduğu kadar az sistem çağrısı izlemesi de performansı arttırabilir. Kullanıcının ihtiyacına göre belirlediği sistem çağrılarını Java ortamına aktarma gerçekleşir. Bu şekilde fonksiyon çağırma sayısı azaltılmış olacaktır.

Ayrıca Java ile C/C++ fonksiyonlarının birlikte kullanılabilmesi için değişik yöntemler de ileriki çalışmalarda denenebilir. Bu şekilde farklı yöntemlerle kıyaslanarak daha etkili bir yöntem bulunabilir.

Saldırı tespit sisteminin daha hızlı çalışması için programların davranışlarını yansıtan az sayıda sistem çağrısı kullanılabilir. Bu şekilde Java ile yerel fonksiyonlar arasındaki iletişim azaltılırken, en yakın komşuluk sistemi de daha az işlem yaparak daha hızlı çalışabilecektir.

Saldırı tespit sisteminde, anormallik belirleme ile birlikte imza tanıma temelli saldırı belirleme yöntemi kullanılması yanlış alarm oranını düşürebilmektedir. Ayrıca bu şekilde yapılan saldırının ne tür bir saldırı olduğu da belirlenebilir.

6. KAYNAKLAR

1. Rochkind, M.J. , Advanced UNIX Programming, Prentice-Hall, New Jersey, 1985.

2. Marshall, A. D., Programming in C UNIX System Calls and Subroutines using C, Wiley Publishing, New Jersey, 1994.

3. Bach, M.J., The Design of the UNIX Operating System, Prentice-Hall, New Jersey, 1986.

4. Çelik, S., UNIX Sistemleri için Dosya Sistemi Erişimlerinin Güvenilir Hale Getirilmesi, Yüksek Lisans Tezi, KTU, Trabzon, Fen Bilimleri, 2004.

5. Faulkner, R., Gomes R., The Process file system and process model in UNIX system V., 1991Winter USENIX Conference, 1991, Dallas, Bildiriler Kitabı, 243-252.

6. Spinellis, D., Trace: A Tool for Logging Operating System Call Transactions, Operating Systems Review, 28, 4(1994) 56-63.

7. Stallman, R., The GNU source-level debugger. Distributed by the Free Software Foundation, Boston, 1989.

8. Beander, B., VAX DEBUG: An interactive, symbolic, multilingual debugger, Proceedings of the Software Engineering Symposium on High-Level Debugging, ACM SIGSOFT/SIGPLAN, 1983, Encio, Bildiriler Kitabı, 173–179.

9. Bernaschi, M., Gabrielli, E. ve Mancini, L., Operating System Enhancements to Prevent the Misuse of System Calls, Proc. ACM Conf. Computer and Comm. Security, 2000, Athens, Bildiriler Kitabı, 174-183.

10. Garfinkel, T., Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools, Proc. Network and Distributed Systems Security Symp., 2003, Bildiriler Kitabı, 163-176.

11. Garfinkel, T., Pfaff, B. ve Rosenblum, M., Ostia: A Delegating Architecture for Secure System Call Interposition, Proc. Network and Distributed Systems Security Symp., 2004, Fortaleza, Bildiriler Kitabı, 187-201.

12. Provos, N., Improving Host Security with System Call Policies, Proc. 12th USENIX Security Symp., 2003, Washington, Bildiriler Kitabı, 257- 272,.

13. Eskin, E., Lee, W. ve Stolfo, S.J., Modeling system calls for intrusion detection with dynamic window sizes. Proceedings of the 2001 DARPA Information Survivability Conference & Exposition II. DISCEX '01, 2001, Arizona, Bildiriler Kitabı, 165 -17.

14. Sekar, R., Bendre, M., Dhurjati, D., ve Bollineni, P., A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors, Proc. IEEE Symp. Security and Privacy, 2001, California, Bildiriler Kitabı, 144-155.

15. Peterson, D. S., Bishop, M., ve Pandey, R., A flexible containment mechanism for executing untrusted code, 11. USENIX Security Symposium, 2002, San Francisco.

16. Kruegel, C., Mutz, D., Valeur, F., ve Vigna, G., On the Detection of Anomalous System Call Arguments, Proc. 8. European Symp. Research in Computer Security (ESORICS '03), 2003, Norveç, 326-343,.

17. Graham, S. L., Kessler, P. B., and Marshall K. McKusick. An execution profiler for modular programs, Software Practice and Experience, Wiley Publishing, California, 1983.

18. Spinellis, D., A C execution profiler for MS-DOS, http://usenet.newsgroup.comp.sources.misc, 8, 8, 1989.

19. Ignatin, G. R., Let the hackers hack: Allowing the reverse engineering of copyrighted computer programs to achieve compatibility, University of Pennsylvania Law Review, 140 (1992) 1999–2050.

20. Holyer, I., ve Pehlivan, H., A recovery mechanism for shells, The Computer Journal, 3, 4 (2000) 1-9.

21. Pehlivan, H., ve Tenekeci, M. E., Unix İşletim Sistemleri İçin Akıllı Geri Dönüşüm Kutusu Tasarımı Ve Gerçekleştirilmesi, Eleco2006, Bursa, Bildiriler Kitabı, 178-182.

22. Shah, V. P., ve Younan, N. H., T. Alford and A. skjellum, An Advanced Signal Processing Toolkit For Java Applications, proceedings of the 2nd international conference on Principles and practice of programming in Java PPPJ '03, 2003, İrlanda, Bildiriler Kitabı, 112-119.

23. Scallan, T., A corba primer. http://www.omg.org/news/whitepapers/seguecorba.pdf, 3 Haziran 2002.

24. Birrell, A. D. ve Nelson, B. J., Implementing remote procedure calls. ACM Transactions on Computer Systems, 2, 1(1984) 39-59.

25. Microsoft Inc Com: Component object model technologies. http://www.microsoft.com/com/default.mspx., 2 Şubat 2000.

26. Rochind, M. J., Jtux – Java to Unix Package, , http://www.basepath.com/aup/jtux/, 25 Kasım 2005.

27. Dueck, G., JniMarshall A Java Native Interface Generator, Brandon University Revision, 2, Ekim 2006.

28. Shin, Y., Sangwook K.,: An MPEG-4 Scene Rendering Technology Using JNI Functions, International Conference on Parallel and Distributed Processing Techniques and Applications, 2002, Las Vegas, Bildiriler Kitabı, 1216-1222.

29. http://en.wikipedia.org/wiki/Java_Native_Interface, Java Native Interface, 22 Ekim 2007.

30. Axelsson, S., Intrusion Detection Systems: A Survey and Taxonomy, Bilgisayar Mühendisliği Teknik Raporu, Chalmers University, Göteborg, 2000.

31. Kemmerer, R.A. ve Vigna, G., Intrusion Detection: A Brief History and Overview, IEEE Computer Special Issue on Security and Privacy, 35, 4 (2002) 27-30.

32. Wu, N. ve Zhang, J., Factor Analysis Based Anomaly Detection, Proc. IEEE Workshop on Information Assurance, 42, 1 (2003) 375 - 389.

33. Stallings, W., Cryptography and Network Security: Principles and Practice, Prentice-Hall, New Jersey, 1998.

34. Smaha, S. E., Haystack: An intrusion detection system, in Proc. of the IEEE 4th Aerospace Computer Security Applications Conference, 1988, Orlando, Bildiriler Kitabı, 37 – 44.

35. Arlowe, H. D. ve Coleman, D. E., The Mobile Intrusion Detection and Assessment System (MIDAS), in Proceedings of the Security Technology Conference, 1990, New York, Bildiriler Kitabı, 54-61.

36. Lunt, T., Tamaru, A., Gilham, F.,Jagannathan, R. ve Jalali, C.; Neumann, P. G.; Javitz, H. S.; Valdes, A.; Garvey, T. D., A real time Intrusion Detection Expert System (IDES), IEEE Symposium on Security and Privacy, 1992, California, Bildiriler Kitabı, 1-13

37. Liepins, G. E., Vaccaro, H. S.: Intrusion Detection: Its role and validation, Computers & Security 11, (1992) 347 – 355.

38. Dowell, C. ve Ramstedt, P., The ComputerWatch data reduction tool, in Proc. of the 13th National Computer Security Conference, 1990, Washington, Bildiriler Kitabı, 99-108.

39. Heberlein, L. T., Network Security Monitor: Final Report, Computer Science Department, California, 1995.

40. Hochberg, J., Jackson, K., Stallings, C., McClary, J., DuBois, D. ve Ford, J., NADIR: An automated system for detecting network intrusions and misuse, Computers and Security, 12, 3(1993) 253 – 248.

41. Debar, H. ve Dorizzi, B., An application of a recurrent network to an intrusion detection system, in Proc. of the International Joint Conference on Neural Networks, 1992, Baltimore, Bildiriler Kitabı, 478 – 483.

42. Snapp, S. R., Smaha, S. E., Grance, T. ve Teal, D. M., The DIDS (Distributed Intrusion Detection System) Prototype, USENIX, Summer 1992 Technical Conference, 1992, San Antonio, Bildiriler Kitabı, 227 – 233.

43. Ilgun, K. ve Kemmerer, R. A., Porras, Ph. A.: State transition analysis: A rule-based intrusion detection approach, IEEE Transactions on Software Engineering 21, 3 (1995) 181 – 199.

44. Crosbie, M., Dole, B.; Ellis, T., Krsul, I. ve Spafford, E., IDIOT - Users Guide, COAST Laboratory Publications, Indiana, 1996.

45. Goldberg L, Wagner D., Thomas R. ve Brewer E.A., A secure environment for untrusted helper applications-Confining the wily hacker, 1996 USENIX Security Symposium. USENIX Assoc, 1996, California, Bildiriler Kitabı, 1-12.

46. Neumann, P. G. ve Porras, A. Ph., Experience with EMERALD to Date, in Proc. of First USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, 1999, California, Bildiriler Kitabı, 73 – 80.

47. Paxson, V., Bro: A System for Detecting Network Intruders in Real-Time, in Proceedings of the 7th USENIX Security Symposium, 1998, San Antonio, Bildiriler Kitabı, 132-142.

48. Miller M., Learning Cost-Sensitive Classification Rules for Network Intrusion Detection using RIPPER, Columbia University Computer Science Technical Report CUCS-035, Colombia, 1999.

49. Lakhina, A. ve Crovella, M. ve Diot, C., Diagnosing Network-Wide Traffic Anomalies, Proc. of ACM SIGCOMM,34, 4(2004) 219-230.

50. Thottan, M. ve Ji, C., Anomaly detection in IP Networks, IEEE Trans on Signal Processing, 51, 8(2003) 234-242.

51. http://www.csharpnedir.com/makalegoster.asp?Mid=811, Metin Madenciliği ile Metin Sınıflandırma, 12 Aralık 2007.

52. http://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html, KNN Tutorial, 22 Şubat 2008.

53. Y. Liao ve V. R. Vemuri. Use of K-Nearest Neighbor Classifier for Intrusion Detection, Computers & Security, 21, 5(2002) 439-448.

54. Ye, N., Li, X., Q., Emran C.S.M. and Xu., M,. Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data, IEEE Trans. SMC-A, 31, 4(2001) 266-274.

55. Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K. ve Zissman, M. A., Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion

Detection Evaluation, Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, 2000, New York, Bildiriler Kitabı, 1012-1023.

56. http://docs.sun.com/app/docs/doc/806-1789/6jb25l4c2?a=view, Audit Token Structure, 15 Mayıs 2008

57. Vigna G. ve Kemmerer R.A., NetSTAT: A Network-Based Intrusion Detection Approach, Computer Security Applications Conference, 1998, Washington, Bildiriler Kitabı, 25 – 34

58. Wenke L., Salvatore J. S. ve Kui W. M., A Data Mining Framework for Building Intrusion Detection Models, IEEE Symposium on Security and Privacy, 1999, Califonia, Bildiriler Kitabı, 120-132

59. Sekar, R., and Uppuluri, P., Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications, Proceedings 8th Usenix Security Symposium, 1999, Washington, Bildiriler Kitabı, 63-78

7. EKLER

Ek 1. Saldırı Tespit Program Kodundan Fonksiyonlar

```
private static double knn_evaluate(double []test, double [][]trained_dataset, int k){
        int i,j,m;
        double sim,avg=0;
        double [] k_near = new double[k];
        for(i=0;i<process_count;i++){
           sim = similarity_evaluate(test,trained_dataset[i]);      //Double versiyon
           if(Math.abs(1.0 - sim) < 0.01){
              return 1.0;
           }
           for(j=0;j<k;j++){
              if(sim>k_near[j]){
                 for(m=k-1; m>j; m--)
                    k_near[m] = k_near[m-1];
                 k_near[j] = sim;
                 break;
              }
           }
        }
        for(m=20; m<k; m++)
           avg += k_near[m];
        return (avg/k);
}

private static double similarity_evaluate(double []test, double []trained){
        int i,j;
        double sim=0;
        double norm_test,norm_trained;
        norm_test = norm(test);       //Double versiyon
        norm_trained = norm(trained);
        for(i=0;i<syscall_count;i++){
           if((test[i]!=0)&&(trained[i]!=0)){
              sim+=test[i]*trained[i];
           }
        }
        sim = sim/(norm_test*norm_trained);
        return sim;
}

private static double norm(double []data){
        int i;
        double norm = 0;
        for(i=0;i<syscall_count;i++)
           norm += Math.pow(data[i],2);
        return Math.sqrt(norm);
}
```

```
private static void define_syscalls(String []syscalls){
        syscalls[0]="access";
        syscalls[1]="execve";
        syscalls[2]="getmsg";
        syscalls[3]="mkdir";
        syscalls[4]="readlink";
        syscalls[5]="setpgrp";
        syscalls[6]="utime";
        syscalls[7]="audit";
        syscalls[8]="exit";
        syscalls[9]="ioctl";
        .....
}

private static void get_test_data_frequency(String test_file){
        int file_length;
        int count = 0;
        int i,j;
        String line="";
        char character;
        FileInputStream finput;
        try {
           finput = new FileInputStream(test_file); //Dikkat
           file_length = finput.available();
           while((count < file_length)&&(character = (char)finput.read()) != -1) {
              if(character != '\n'){
                 if(character != '\r'){
                    line +=character;

                 }
                 count++;
              }
              else{
                 //line.trim();
                 for(i=0;i<syscall_count;i++){
                    if(line.compareTo(checked_system_calls[i])==0){
                       test_frequency_data[i]++;
                       break;
                    }
                 }
                 line="";
              }
           }
           finput.close();// baglantiyi kapat

        } catch ( IOException e ) {
           System.err.println(e);
        }
}
```

```
private static void calculate_test_data_tfidf(int []test_data){
        int i, j;
        double norm = 0;
        int n_i = 0;
        double a_i_j = 0;
        for (j = 0; j < syscall_count; j++)
        {
           norm+=test_data[j]*test_data[j];
        }
        norm = Math.sqrt(norm);
        for (j = 0; j < syscall_count; j++)
        {
           //n_i = test_data[j];
           n_i = 0;
           a_i_j = 0;
           if (test_data[j] != 0)
           {
              for (int k = 0; k < process_count; k++)
              {
                 n_i += trained_syscall_frequency[k][j];
              }
              if((n_i!=0)&&(norm!=0))
              {
                 a_i_j = (test_data[j] * Math.log((double)(process_count+1) / n_i)) / norm;
                 a_i_j = Math.abs(a_i_j);
              }
           }
           test_data_tfidf[j]=a_i_j;
        }
}

private static void read_trained_data(String filename){

        FileInputStream finput;
        int i=0,j=0,count=0;
        char character;
        String line="";
        int file_length;
        String []calls_freq;

        try {
           finput = new FileInputStream(filename); //Dikkat

           file_length = finput.available();
           while((count < file_length)&&(character = (char)finput.read()) != -1) {
              if(character != '\n'){
                 line +=character;
                 count++;
              }
              else{
```

```
                calls_freq = line.split(",");
                for(j=0;j<50;j++){
                   trained_syscall_frequency[i][j] = Integer.parseInt(calls_freq[j]);
                }
                i++;
                line="";
             }
          }
          finput.close();

       } catch ( IOException e ) {
          System.err.println(e);
       }
}

 private static void read_trained_tfidf(String filename){

       FileInputStream finput;
       int i=0,j=0,count=0;
       char character;
       String line="";
       int file_length;
       String []calls_freq;

       try {
          finput = new FileInputStream(filename);

          file_length = finput.available();
          while((count < file_length)&&(character = (char)finput.read()) != -1) {
             if(character != '\n'){
                line +=character;
                count++;
             }
             else{
                calls_freq = line.split(",");
                for(j=0;j<50;j++){
                   trained_syscall_tfidf[i][j] = parseDouble(calls_freq[j]);
                }
                i++;
                line="";
             }
          }
          finput.close();// baglantiyi kapat

       } catch ( IOException e ) {
          System.err.println(e);
       }
}
```

ÖZGEÇMİŞ

1983 yılında Şanlıurfa' da doğdu. İlk öğrenimini Şehit Nusret İlkokulunda, orta öğrenimini Şanlıurfa Anadolu Lisesi'nde tamamladı. 2000 yılında Karadeniz Teknik Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde lisans programına başladı ve 2005 yılında bu bölümden mezun oldu. Aynı yıl Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans programına ve aynı bölümde araştırma görevlisi olarak çalışmaya başladı. Halen bu görevini sürdürmektedir. Yabancı dil olarak iyi düzeyde İngilizce ve orta düzeyde Almanca bilmektedir.