YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

LINUX İŞLETİM SİSTEMİ ÇEKİRDEĞİ İLE BÜTÜNLEŞİK BİR KRİPTOGRAFİK SİSTEMİN TASARIMI VE GERÇEKLENMESİ

Bilgisayar Mühendisi M. Amaç GÜVENSAN

FBE Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Yrd. Doç. Dr. A. Gökhan YAVUZ (Y.T.Ü.)

İSTANBUL, 2006

İÇİNDEKİLER

KISALTMA LİSTESİ

AES	Advanced Encryption Standart
CBC	Chiper Blocking Chain
ECB	Electronic Code Book
FTP	File Transfer Protocol
HTTP	Hypertext Transport Protocol
HTTPS	Secure Hyper Text Transport Protocol
IBM	International Business Management
ICMP	Internet Control Message Protocol
IDC	International Data Corporation
IETF	International Engineer Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LKM	Loadable Kernel Module
MAC	Medium Access Control
MB	Mega Byte
MTU	Media Transmission Unit
PDA	Personal Data Assistant
PILSC	Protocol Independent Lightweight Secure Communication
OpenSSL	Open Secure Socket Layer
SecureFTP	Secure File Transfer Protocol
SSL	Secure Socket Layer
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
TTL	Time To Live

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Tüm hayatım boyunca yanımda ve her konuda destek olan aileme, özellikle beni yüksek lisans yapmam için teşvik eden anneme, yüksek lisans eğitimi ve tez geliştirme süresince yardımlarını, bilgisini ve zamanını esirgemeyen danışman hocam Yrd. Doç. Dr. A. Gökhan YAVUZ'a ve bölüm başkanımız Prof. Oya KALIPSIZ'a teşekkürü borç bilirim.

ÖZET

Günümüzde Internet sayesinde bilgi paylaşımı hızla artmaktadır. Teknolojide yaşanan hızlı gelişme bilginin önemini daha da arttırmıştır. Bilginin değerinin artması ve bilgi paylaşımının kolay hale gelmesi sonucu güvenli veri iletişimi ile ilgili yapılan çalışmalara ağırlık verilmiştir.

Güvenli veri iletişimi için uygulama odaklı çözümler üretilerek uygulamalara özel bilgilerin korunması sağlanmıştır. Genel anlamda yetersiz kalan bu çözümlere alternatif olarak uygulama bağımsız, Transmission Control Protocol (TCP) paketlerinin güvenliğini sağlayan başka protokoller geliştirilmiştir. Fakat bu protokoller sadece TCP protokolünü destekleyebilmektedir. User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), vb. Internet Protocol (IP) protokollerine destek verebilmek için çalışmalar protokol bağımsız çalışabilecek bir mimari üzerine yoğunlaşmıştır. Sonuç olarak sık kullanılan Internet Protocol Security (IPSec) protokolü tasarlanmıştır. Fakat bu protokolün de bazı dezavantajları vardır. Zor düzenlenebilir yapısı ve her paket için eklediği ekstra verinin, iletişimi yavaşlatması bunlardan bazılarıdır (Alshamsi ve Saito, 2005).

Geliştirdiğimiz sistem, Protocol Independent Lightweight Secure Communication (PILSC), güvenli veri iletişimine standart bir çözüm getirebilmek için IPv4 tarafından tanımlı fakat kullanılmayan güvenlik alanına işlerlik kazandırmaktadır. Tez çalışmasında, veri iletişiminde kullanılan güvenlik mekanizmalarının eksik yönleri incelenip yeni bir sistem tasarlanmıştır. IPSec'in protokol bağımsız çalışabilmesine rağmen yavaş veri transferi ve zor düzenlenebilir yapısı, daha hızlı çalışan ve kolay düzenlenebilir bir yapı olan PILSC'in tasarlanması için bir neden olmuştur. Tez çalışmasının en önemli hedefi, sistemin çekirdek seviyesinde gerçeklenmesi ile kriptografik işlemlerin zaman tüketimini en aza indirip, güvenli veri iletişiminin hızını arttırmaktır. PILSC verinin gizliliğine önem verirken bilginin kimliğinin doğrulanması (authentication) ve bütünlüğü (integrity) konularında yapılacak çalışmalara açık bir yapıya sahiptir.

Tasarlanan mimari Linux işletim sistemi üzerinde gerçeklenmiştir. Geliştirme metodu olarak birçok avantaj sağlayan Loadable Kernel Module (LKM) seçilmiştir. Sistem gerçeklendiğinde çalışan güvenlik modülü, modülün yüklü olduğu bilgisayardaki veri iletişiminin seçilen kurallara uygun olarak yönetilmesini sağlar.Yapılan testler sonucunda PILSC sisteminin çekirdek seviyesinde çalışmasının şifreleme işlemlerinin kullanıcı seviyesinde yapılmasına göre %75-%90 performans artışı sağladığı görülmüştür. Ayrıca PILSC sistemi, IPSec ve SSL'e göre güvenli veri transferini %20-%25 daha hızlı gerçekleştirebilmektedir.

Anahtar Kelimeler: Linux, protokol bağımsız, kolay düzenlenebilir, hızlı güvenli veri iletişimi

ABSTRACT

Nowadays, thanks to Internet, information sharing is growing rapidly. The rapid growth of technology causes the increase of the importance of information. Due to the ease of information sharing and rise of the information value many researches on secure communication has been made recently.

Many application-dependent solutions, which ensure only security of data that belong to its application, were developed. In order to create alternative solutions to overcome disadvantages of these applications, new application-independent protocols, which ensures the confidentiality of TCP packets, have been developed. However, these protocols support only TCP protocol. In order to include support for other IP protocols, like UDP, ICMP, ..., researchers focused on protocol-independent architecture. Consequently, IPSec protocol was designed, which is frequently used. However, this protocol has some disadvantages. Its hardly configurable structure and its redundant overhead problem are some of them (Alshamsi ve Saito, 2005).

PILSC, implemented system, is using the security feature of IPv4, which is defined but not used, in order to have an standardization at secure communication. In this work the disadvantages of well-known security mechanisms are examined and a new system, called PILSC, is designed. One of the important reasons to design is the hardly configurable structure and slow data transfer rate of IPSec. The main intention of this work is to increase the data transfer rate by performing the cryptographic processes in the kernel. PILSC mainly focuses on confidentiality. It does not directly address the authenticity and integrity, although it can be extended this way very easily.

PILSC architecture is implemented on Linux operating system. As a development method Loadable Kernel Module is preferred due to its many advantages. While PILSC is running as a security module on Linux operating system, it manages the data transfer according to the declared rule database. The implementation of PILSC on the kernel-level brings %75-%90 performance enhancement on cryptographic process time according to the implementation of cryptographic processes on user-space. Moreover, secure data transfer rate of PILSC is %20-%25 better than IPSec and SSL.

Keywords: Linux, protocol-independent, easily configurable structure, fast secure communication

1. GİRİŞ

Günümüzde bilgi paylaşımı Internet sayesinde hem kolay hem de hızlı yapılabilmektedir. Her gün birçok önemli ve gizli bilgi yerel ve geniş ağ yapıları arasında bir bilgisayardan diğer bir bilgisayara aktarılmaktadır. Bilgi paylaşımındaki bu hızlı artış, güvenli veri iletişiminin önemini daha da yükseltmiştir.

Bilginin önemi arttıkça transfer edilen verinin gizliliğinin de önemi artar. Birçok ağ uygulamasının temel prensibi transfer edilen verinin bütünlüğünü ve gizliliğini sağlamaktır. Bu temel prensibi yerine getirmek için genelde her uygulama kendine özgü çözümler üretmektedir. Çözümler uygulama odaklı oldukları için ortaya çıkan farklılıklar araştırmacıları yeni yaklaşımlara yöneltmiştir (Oppliger, 1998).

Güvenli veri iletişimini sağlayabilmek için uygulamadan bağımsız çözümler üzerinde çalışılmış ve uygulama katmanı yerine taşıma katmanında servis veren protokoller tasarlanmıştır. Bu protokollerin gerçeklenmesi, ağlarda dolaşan verilerin güvenliğini sağlamak için atılmış akılcı bir adım olsa da güvenli veri iletişimini gerçekleştirmek için bir standart sunmamaktadır. Ayrıca bu protokoller sadece bir IP alt protokolü olan TCP protokolü için çalışmaktadır (Apostolopoulos vd., 1999).

Güvenli ağ trafiğini uygulama ve protokol bağımsız kılabilmek için araştırmacılar tarafından ağ katmanında hizmet veren yeni bir yapı tasarlanmıştır. Tasarlanan yeni yapı (IPSec) birçok konuda çözümler üretse de eksiklikleri ve en önemlisi veri güvenliğinde beklenen standardı sağlayamaması yeni alternatiflere kapıyı açık bırakmıştır.

Güvenli veri iletişimini sağlamak için bugüne kadar gerçekleştirilen tasarımlar hakkında ve PILSC ile ilgili daha ayrıntılı bilgi tezin 2. bölümünde verilmektedir. Geliştirilen platformun daha iyi anlaşılabilmesi için 3. bölümde Linux işletim sistemi ile ilgili ayrıntılı bilgi verilmiştir. 4. bölümde PILSC sisteminin tasarım aşaması, 5. bölümde uygulamanın gerçeklenmesi, 6. bölümde kullanılan fonksiyonlar, 7. bölümde kullanılan veri yapıları, 8. bölümde kullanılan dosya yapıları anlatılmaktadır. 9. ve 10. bölümlerde test sonuçları ve PILSC sisteminin özellikleri yorumlanmıştır.

Bu tez çalışmasında, güvenli veri iletişimini sağlamak için uygulama ve protokol bağımsız bir mimari tasarlanmış, IPv4'ün güvenlik özelliği kullanılarak güvenli veri iletişimi için standart

bir çözüm belirtilmiş ve Linux işletim sistemi üzerinde gerçeklenerek sistemin test edilmesi ile elde edilen sonuçlar değerlendirilmiştir.

2. BİLGİSAYARDA GÜVENLİK

Dijital olarak saklanan bilgilerin güvenliğini sağlayabilmek için çok çeşitli yöntemler uygulanmaktadır. Bu yöntemler incelediğinde dijital ortamdaki güvenlik konusu üç başlık altında değerlendirilmektedir [1].

- *İşletim Sistemi Güvenliği*
- *Ağ Güvenliği*
- *Uygulama Güvenliği*

İşletim sistemi güvenliği bilgisayar üzerinde paylaşılan verinin, belirlenen kullanıcılar tarafından onlara verilen haklar doğrultusunda kullanılmasını amaçlar. İşletim sistemi kullanıcı ile makine arasındaki interaktif işlemlerin yönetiminden sorumludur. Bu nedenle çalıştığı ortamdaki tüm bilgilerin güvenliğini sağlayabilmelidir.

Uygulama güvenliği veri ve bilgi yönetiminin gizliliğinden sorumludur. Bunun gerçeklenmesi ise geliştirilen uygulamaların yaratacağı açıkların tespit edilmesi ve giderilmesine yönelik çalışmalar ile sağlanır.

Ağ güvenliği ise ağlar arasında aktarılan verinin güvenliği ile ilgilenir. Bu tez çalışmasında ağlar içinde ve arasında paylaşılan verinin güvenliğini sağlamak için bir mimari tasarlanıp gerçeklenmiştir.

2.1 Bilgi Güvenliği ve Saldırı Çeşitleri

Bilgisayarda güvenlik başlığı altında bahsedilen temel başlıkların hepsinin amacı bilgi güvenliğini sağlamaktır. Bilgi güvenliğini sağlamak için dört ana ölçüt vardır (Schneier, 1996).

- *Gizlilik* (Confidentiality)
- *Kimlik Doğrulama* (Authentication)

- *İnkar edilememezlik* (Non-repudiation)
- *Bütünlük* (Integrity)

Gizlilik : Bilginin gizliliğini ifade eder. Bilginin güvenliğini sağlayabilmek için en önemli kriterdir. Bu özelliği sağlamak kriptografik sistemlerin ana hedeflerinden biridir.

Orijinallik : Bilginin sahibinin, iddia ettiği kişi olmasını ifade eder. Bilginin bir başkası adına gönderilip gönderilmediğini sorgulayan bir mekanizmadır.

İnkar edilememezlik : Bilgiyi gönderen kişinin gönderdiği bilgiyi ve bilgiyi alan kişinin de aldığı bilgiyi inkar edememesini ifade eder. Gönderici ve alıcı gönderilen ve alınan bilginin varlığını üstlenmelidir.

Bütünlük : Bilginin bütünlüğünü ifade eder. Verinin transferi sırasında bilginin değiştirilmediği mutlaka kontrol edilmelidir.

Yukarıda belirtilen kriterler güvenli veri iletişiminin temel taşlarıdır. Oluşturulan veri güvenliği protokolleri bu özellikler göz önünde bulundurularak tasarlanmaktadır

Bilginin ağlar içinde ve arasında transferi sırasında, kötü niyetli kişiler tarafından çeşitli saldırılar gerçekleştirilmektedir. Her geçen gün artan saldırı çeşitleri üç başlık altında incelenebilir (Schneier, 1996).

- *İzleme* (Monitoring)
- *Araya Girme* (Interception)
- *Yeniden Oluşturma* (Construction)

İzleme : Bilginin pasif olarak kopyalanmasını ifade eder. Verinin aktarılırken istenmeyen kişiler tarafından kopyalanıp okunması ile gerçekleştirilen saldırı çeşididir.

Araya Girme : Bilginin değiştirilmesini ifade eder. Verinin transferi sırasında istenmeyen kişiler tarafından değiştirilmesi ile gerçekleşen saldırı çeşididir.

Yeniden Oluşturma : Olmayan bilginin yaratılmasını ifade eder. Verinin farklı bir kullanıcı tarafından saldırı amaçlı oluşturulup istenen hedefe gönderilmesi ile gerçekleşen saldırı çeşididir.

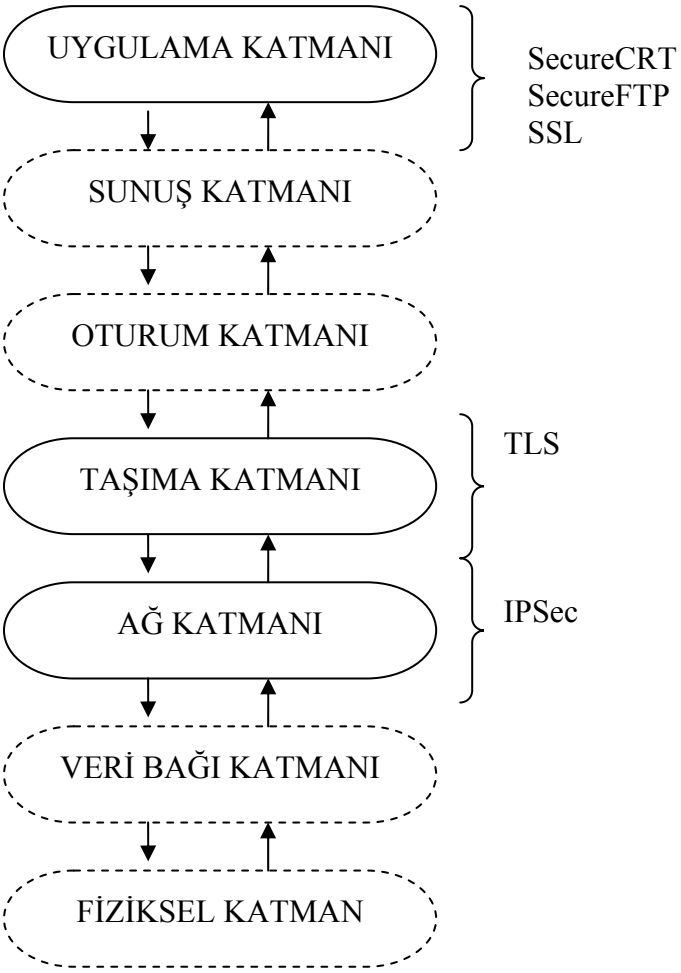
2.2 Ağ Güvenliği

Bir önceki bölümde bahsedilen saldırı tiplerini gerçekleştirmek için çeşitli mekanizmalar düzenlenmekte ve farklı yöntemler uygulanmaktadır. Ağ güvenliğinin amacı bu saldırı tiplerine karşı önlem alarak bilginin gizliliğini ve bütünlüğünü korumaktır. Aynı zamanda gelen bilginin gönderen kişiye ait olup olmadığının kontrolünden de sorumludur.

Son yıllarda güvenli veri iletişimi için çeşitli güvenlik mekanizmaları geliştirilmiştir. Ağırlıklı olarak uygulama bağımlı mimariler tasarlanmıştır. Fakat bu mimariler bir standarda sahip olmadıkları için yeni arayışlar başlamıştır. Bunun neticesinde alt katmanlarda çalışan mekanizmalara yönelinmiş ve Secure Socket Layer (SSL), Transport Layer Security (TLS) protokolleri tasarlanmıştır. Bu protokoller sadece TCP protokolünü destekleyebilmektedir, bunun yanında TCP başlık bilgilerini şifreleme özelliğine de sahip değildir. Bu eksikliklerden dolayı yeni yaklaşımlar ortaya atılmıştır. Sonuç olarak günümüzde en çok bilinen, güvenli ve sık kullanılan, ağ katmanında çalışan IPSec mimarisi tasarlanmıştır. (Alshamsi ve Saito, 2005).

Güvenli veri iletişimi için geliştirilen aynı zamanda Şekil 2.1'de gösterilen protokoller üç değişik katmanda çalışmaktadırlar:

- *Uygulama Katmanı*
- *Taşıma Katmanı*
- *Ağ Katmanı*



Şekil 2.1 Farklı OSI katmanlarında çalışan güvenlik protokolleri [Tannenbaum, 1999]

2.2.1 Uygulama Katmanı Protokolleri

Birçok farklı protokol, Secure Shell (SSH), Secure File Transfer Protocol (SecureFTP), vb., güvenli veri iletişimini sağlamak için tasarlanmıştır. Örneğin SSH uzaktan erişim için sıkça kullanılan bir protokoldür. SSH bir bilgisayara güvensiz ağlar üzerinden güvenli olarak bağlanılmasını, komutların çalıştırılmasını ve uzak bilgisayarla yerel bilgisayar arasında dosya transferi yapılmasını sağlamaktadır. SSH protokolünü kullanan SecureCRT [2] programı güvenli iletişim için bir alternatiftir. Fakat sadece birkaç protokolün şifreli iletişimine destek verebilmesi bu konudaki kısıtlı yaklaşımını göstermektedir.

SSH sadece uzaktan erişim komutlarının güvenli iletişimini gerçekleştirirken, SecureFTP ise sadece dosya transferlerini şifreli olarak gerçekleştirmektedir (Alshamsi ve Saito, 2002). Başka uygulamalar da sadece kendileri tarafından üretilen verinin güvenliğini

sağlayabilmektedir. Bu nedenle uygulama katmanı protokollerinde bir standarttan ve merkezi yönetimden söz etmek mümkün değildir.

SSL (Secure Socket Layer) TCP üzerinden çalışan uygulamaların güvenli veri iletişimini sağlayan Netscape tarafından geliştirilmiş bir protokoldür. SSL bilginin onay almasını sağlayıp transfer edilen verinin şifrelenmesini ve deşifrelenmesini gerçekleştirmektedir. Servis tarayıcılarının çoğu tarafından başta Netscape ve Internet Explorer olmak üzere desteklenen SSL verinin güvenliğini sağlamak için birden fazla kriptografik algoritma kullanmaktadır. Secure Hyper Text Transport Protocol (HTTPS) elektronik ticaret sırasında gerçekleşen web işlemlerinin güvenliğini sağlamak için SSL protokolünü kullanır. SSL özellikle Internet üzerinden gerçekleşen kredi kartı bilgi transferinin gizliliği için tercih edilmektedir (Kant vd., 2000).

SSL istemci/sunucu (client/server) mimarisine dayanır. Her ne kadar sunucular SSL için yüksek performansla çalışsalar da istemci tarafında oluşan uzun cevap süreleri %10-25 civarında web işlemlerinin iptal edilmesine yol açmaktadır (Kant vd., 2000). Uygulamalar SSL'i doğrudan kullanamazlar. Bu protokolü kullanan uygulamalar üzerinde bazı değişiklikler yapılması gereklidir. Ayrıca SSL; UDP, ICMP, vb. IP protokollerini desteklememektedir. Bağlantı aşamasında bağlantının kurulabilmesi için sunucu ve istemci arasında bilgi değişimi gerçekleştirilir. Gönderilen ekstra veri, bilginin bütünlüğünü ve kimlik doğrulamasını sağlarken veri transferinin gecikmeli olarak gerçekleşmesine neden olur. Pentium 4 2.4 Ghz işlemcili, 512 MB hafızalı, üzerinde RedHat Fedora Core 1 Linux işletim sistemi yüklü olan bilgisayarlar arasında yapılan testte ölçülen SSL tokalaşma (handshaking) süreleri Çizelge 2.1'de gösterilmektedir (Alshamsi ve Saito, 2005).

Çizelge 2.1 SSL tokalaşma süresi

Modu	**Kurulma Süresi**
Sunucu Kimlik Doğrulama	41.7 ms
İstemci Kimlik Doğrulama	74.8 ms

2.2.2 Taşıma Katmanı Protokolleri

Internet Engineering Task Force (IETF) SSL protokolünü TLS adı altında bir standart haline getirmiştir. TLS, TCP üzerinden çalışan tüm uygulamalar tarafından desteklenmektedir. Çoğunlukla güvenli web erişimleri için Hyper Text Transfer Protocol (HTTP) protokolü TLS

protokolü ile beraber çalışmaktadır. TLS protokolü iki uygulama arasındaki veri transferinde verinin bütünlüğünü ve gizliliğini (confidentiality) sağlar. TLS protokolü iki katmandan oluşur (Yasinsac ve Childs 2001).

- TLS Record Protocol
- TLS Handshake Protocol

TLS Record protokolü, veri gizliliği ve bütünlüğü için bağlantı güvenliğini sağlar. TLS Handshake protokolü ise veri transferi yapılmadan önce konuşacak iki ucun senkronize olması için gerekli bilgi transferinin yapılmasını kontrol eder.

TLS hem ucuz hem de kolay gerçeklenebilir yapısı ile güvenli veri iletişimi iyi bir alternatiftir. TLS protokolünün artılarından biri de *session caching* yöntemi ile bağlantı sayısı minimumda tutularak performansın arttırılmasıdır. Ama hala TLS, SSL'in yetersizliklerini tam anlamıyla gidememiştir. Örneğin TLS belirli tipteki verileri şifrelemek için kullanılmaktadır. Tam güvenlik gereken durumlarda TLS sadece verinin belirli kısmını şifreleyebilmekte ve bu nedenle yetersiz kalmaktadır. TLS'in bir diğer dezavantajı da çalışmasını hızlandırmak için gerekli donanımın pahalı olması ve zor bulunmasıdır. Uç kullanıcıların bilgisayarlarındaki yavaşlık performans kaybına neden olmaktadır (Apostolopoulos vd., 1999).

TLS protokolünde gerçekleşen tokalaşma sırasında yapılan veri değişimi HTTP işlemleri (transaction) sürelerini arttırmaktadır.

2.2.3 Ağ Katmanı Protokolleri

IPSec bir takım protokollerin bir araya getirilmesi ile ağ katmanı seviyesinde oluşturulan bir mimaridir. IPSec protokolden bağımsız iki uygulama arasında paket işleme seviyesinde güvenlik sağlar. Güvenli iletişim için geliştirilen uygulamalar arasında protokol bağımsız tek çözümdür (Alshamsi ve Saito 2005).

IPSec güvenlik servisi olarak iki seçenek sunmaktadır:

- Authentication Header
- Encapsulating Security Payload

"Authentication Header" servisi gönderilen verinin doğrulanmasını sağlar. Bu servisin quick *mode* olarak adlandırılan tek bir modu vardır. "Encapsulating Security Payload" servisi ise hem gönderilen verinin doğrulanmasını hem de şifreli olarak aktarılmasını sağlar. Bu servis iki ayrı modda çalışabilmektedir. *main mode* ve *aggressive mode* olarak adlandırılan bu modların birbirinden farkı, değiştirilen mesaj sayısı ve ID koruma yapılarıdır.(Alshamsi ve Saito, 2005).

IPSec protokolü de diğer protokollerin (SSL, TLS) kullandığı gibi ekstra başlık bilgisi kullanmaktadır. Ekstra bilgi ekstra zaman gerektirmektedir. IPSec ayrıca güvenilirlik ve parçalara ayırma (fragmentation) işlemlerini sağlamakla yükümlüdür. Oysaki TLS ve SSL protokolleri, TCP katmanında çalıştıkları için güvenilirlik ve parçalara ayırma işlemlerinden sorumlu değildir (Oppliger, 1998).

IPSec protokolünün en büyük dezavantajı zor düzenlenebilir (configure) yapısıdır. IPSec tüm IP protokolleri için şifrelemeyi desteklemesine rağmen tokalaşma mekanizması SSL'inkine göre daha yavaş çalışır. Pentium 4 2.4 Ghz işlemcili, 512 MB hafızalı, üzerinde RedHat Fedora Core 1 Linux işletim sistemi yüklü olan bilgisayarlar arasında yapılan testte ölçülen IPSec süreleri Çizelge 2.2'de, SSL süreleri Çizelge 2.1'de gösterilmiştir. Ayrıca IPSec bağlantılarında iki tarafta da aynı üreticinin ürünleri kullanılmalıdır (Alshamsi ve Saito, 2005).

Çizelge 2.2 IPSec tokalaşma süresi

Mod	**Kurulma Süresi**
Main Mode (PSK)	97 ms
Aggresive Mode (PSK)	56 ms
Main Mode (RSA)	170 ms

2.2.4 Protokollerin Karşılaştırmalı İncelenmesi

Yukarıdaki bölümlerde sık kullanılan güvenlik mekanizmaları incelenmiştir. Çizelge 2.3'te ise bahsedilen mekanizmaların karşılaştırmalı olarak özellikleri belirtilmiştir. IPSec birçok özelliği ile ön plana çıkmaktadır. Örneğin IPSec mimarisinin protokol bağımsız yapısı TCP paketlerinin yanında UDP paketlerinin güvenliğini de sağlamaktadır. IPSec mimarisinin diğer

protokollere göre dezavantajı tokalaşma sırasında diğerlerine göre yavaş kalması ve düzenleme işlemlerinde yaşanan zorluklar olarak gösterilebilir.

Çizelge 2.3 Güvenlik protokollerin özellikleri

Özellik	**SSH/SecureFTP**	**SSL/TLS**	**IPSec**
Uygulama Bağımlı	Evet	Hayır	Hayır
Protokol Bağımlı	Evet	Evet	Hayır
Tokalaşma Süresi	Hızlı	Hızlı	Yavaş
TCP Desteği	Hayır	Bazı	Hepsi
UDP Desteği	Hayır	Hayır	Evet
Konfigürasyon	Kolay	Kolay	Zor

2.3 PILSC (Protocol Independent Lightweight Secure Communication)

Bu tez çalışmasında daha önceki bölümlerde incelenen protokollerin eksiklikleri değerlendirilip PILSC adı verilen güvenli veri iletişimi için alternatif oluşturacak yeni bir mimari tasarlanmıştır.

Kriptografik işlemler zaman alıcı işlemlerdir. Bu nedenle, verinin önemine göre güvenli veri iletişimi ile hızlı veri iletişimi arasındaki dengenin iyi ayarlanması gerekir. Bazı durumlarda saklayacak bilgi yoksa güvenlik ihmal edilebilir.

Veri iletişiminde güvenlik tek önemli kriter değildir. File Transfer Protocol (FTP), HTTP vb. protokolleri kullanan uygulamalar hızlı veri iletimine ihtiyaç duyarlar. IPSec bu ihtiyacı karşılayamamaktır. IPSec'in kullandığı hash algoritmalarından ve tokalaşma mekanizmasından dolayı veri iletişimi yavaş gerçekleşmektedir. PILSC sisteminde kriptografik işlemler çekirdek seviyesinde gerçeklenerek oluşan kayıp azaltılarak veri iletimi hızlandırılmıştır.

Günümüzde sayısı hızla artan birçok uygulama özellikle gerçek zamanlı uygulamalar UDP protokolünü kullanmaktadır. Bu nedenle PILSC; TCP paketleri dışında UDP, ICMP vb. paketlerin de güvenli transferini desteklemektedir.

Internet katmanı seviyesinde sağlanan güvenliğin temel dezavantajı IP yığınlarının yetersiz kalmasıdır (Oppliger, 1998). PILSC sistemi, IPSec'e göre daha hızlı bir mekanizma sunarken diğer uygulamalarda görülen eksiklikleri gidermek için tasarlanmıştır.

PILSC, IPv4 paketleri için oluşturulmuş güvenlik alanını kullanarak şifreli iletişimde bir standart oluşturur. Buna göre şifreli IPv4 paketlerinde güvelik alanının değeri "130" olarak belirlenir [3][4]. IPv6 için zaten geliştirilmiş bir şifre mekanizması bulunmaktadır. PILSC ise IPv4 için tasarlanmış fakat kullanılmayan şifre mekanizmasını işler hale getirmektedir.

Geliştirilen mimarinin en önemli avantajı kolay düzenlenebilir olmasıdır. Hem üst düzey kullanıcı hem de alt düzey kullanıcı rahatlıkla şifreli konuşacak uygulamalara ait IP adreslerini, portları, şifreleme algoritmalarını belirleyerek verimli bir haberleşme sistemi oluşturur.

PILSC yeni bir yaklaşım daha ortaya atmaktadır. Güvenlik seviyesi bilginin hassasiyetine göre kullanıcı tarafından değiştirilebilir. Belirlenen güvenlik seviyesine göre sistem tarafından kullanılacak algoritmaya karar verilir. PILSC sistemi IP adresi, protokol ve port bazında farklı anahtar seçimlerini destekleyebilmektedir.

PILSC sisteminin ana özellikleri Çizelge 2.4'te belirtilmiştir.

Çizelge 2.4 PILSC sisteminin temel özellikleri

1	Uygulama bağımsız
2	Protokol bağımsız
3	Kolay düzenleniş (configuration)
4	Kullanıcıya saydam bir yapı
5	Karar veren bir yapı
6	Mevcut/kabul görmüş algoritmaların desteği
7	Yeni şifreleme algoritmalarına açık olması
8	IPv4 ün güvenlik özelliğinin kullanılması

3. LINUX İŞLETİM SİSTEMİ

Linux, 1991 yılında Finlandiya Helsinki Üniversitesi'nde Linus Torvalds tarafından geliştirilen, UNIX işletim sistemine benzeyen fakat UNIX ile birebir aynı olmayan bir işletim sistemidir. Linux, Intel tabanlı platformlardan Digital Alpha, PowerPC veya IBM'in mainframe bilgisayarlarına kadar birbirinden farklı platformlarda çalışabilmektedir. Linux'un diğer işletim sistemlerinden en önemli farkı serbest dağıtım özelliğidir. Linux'un kaynak kodları ve Linux dağıtımları ücretsiz olarak temin edilebilmektedir.

International Data Corporation'ının yaptığı araştırmaya göre Linux işletim sistemi 2003 yılında marketin %6'sına sahipken, 2008 yılında %17'sine sahip olacağı tahmin edilmektedir. Araştırma ayrıca Linux'a olan ilginin artışına dikkat çekiyor. Bunun altında yatan neden olarak ise sağlam bir işletim sistemi olan UNIX'e alternatif olacak özelliklere sahip olması gösteriliyor.

Linux temelde Intel tabanlı bilgisayarlarda çalışmak için tasarlanmış olsa da, modüler ve taşınabilir kodlama sayesinde, IBM'in Watchpad'i ve Samsung'un Yopy'si gibi Personal Data Assistant'lardan (PDA), Los Alamos National Labs.'da kullanılan ve dünyanın en hızlı 500 bilgisayarından biri olarak gösterilen Avalon'a kadar birbirinden farklı platformlarda çalışabilmektedir.

3.1 Linux İşletim Sisteminin Diğer İşletim Sistemleri ile Karşılaştırması

Linux işletim sisteminin diğer işletim sistemleri ile yapılan karşılaştırmalarında alınan araştırma sonuçları başlıklar halinde incelenmiştir.

3.1.1 Güvenilirlik ve Kararlılık

IBM Linux Teknoloji Merkezi tarafından gerçekleştirilen testte Linux işletim sisteminin güvenilirliği ve kararlılığı ölçülmüştür. Test ortamında donanım olarak IBM pSeries, sunucular ve işletim sistemi olarak SUSE Linux Enterprise v8 kullanılmıştır. Yapılan 30 günlük stres testinde başarı %97,88 olarak ölçülmüştür. 60 günlük test ise %95,12 başarı oranı ile tamamlanmıştır (Ge vd., 2004).

Wisconsin Üniversitesi'nde yapılan bir işletim sistemi testinde farklı işletim sistemlerinde çalışan uygulamalara rasgele giriş yapılarak, sistemlerin devre dışı kalma oranları gözlemlenmiştir. Teste tabi tutulan 7 ticari işletim sisteminin ortalama hatalı çalışma olasılığı gözlemler sonucu %23 olarak hesaplanırken Linux'un hatalı çalışma veya devre dışı kalma olasılığı %9 olarak hesaplanmıştır.

Yapılan araştırmalar Linux'un uzun süreler (aylarca) çalıştığında bile hafıza yönetimi nedeniyle yaşanacak kilitlenme sıkıntısı yaşamadığını ve yavaşlama belirtisi göstermediğini ortaya koymuştur.

3.1.2 Performans

InformationWeeks Research tarafından gerçekleştirilen testte Windows Server 2003 DataCenter Edition üzerinde çalışan SQL Server Enterprise Edition ile SUSE Linux Enterprise Server 9 üzerinde çalışan Oracle 10g. karşılaştırılmıştır. Sonuçta %22 daha ucuz olan Linux-Oracle ikilisi %18 daha hızlı performans ortaya koymuştur (Legard, 2004).

Linux'un performansının ölçüldüğü ve IBM tarafından yapılan bir karşılaştırmada RedHat 7.1 (Kernel 2.4.2) işletim sisteminin, pipe kullanımı, görev (process) ve thread yaratılmasında Windows 2000 ve Windows XP'den daha iyi performans gösterdiği açıklanmıştır. Pipe kullanımında Linux'un G/Ç oranı 700MB/saniye olarak belirlenirken, bu değer büyük blok boyutlarında yaklaşık 100MB/saniyede sabitlenmektedir. Bu değerler Windows 2000'de en çok 500MB/saniye ve ortalama 80MB/saniye, Windows XP'de ise 120MB/saniye ve 80MB/saniyedir. Şubat 2002'de yayınlanan diğer bir yazıya göre RedHat Linux 7.2 saniyede 10.000 görev yaratabilirken Windows 2000, 5000 değerine yaklaşmış, Windows XP ise 6000 görev yaratabilmiştir. Thread yaratma karşılaştırmasında ise RedHat Linux saniyede 330 thread yaratmış; buna karşın Windows 2000, 200 ve Windows XP, 160'ın altında thread yaratabilmiştir (Bradford, 2001).

Linux iş istasyonlarında ve ağ yapılarında sürekli yüksek performans sağlarken, aynı anda alışılmadık oranda fazla kullanıcıyı sorunsuz olarak yönetebilmektedir.

3.1.3 Ölçeklenebilirlik

Linux'un ölçeklenebilirliği ile ilgili en önemli kanıt desteklediği platformların sayısıdır. Linux bir PDA'de çalışabileceği gibi, Intel tabanlı bir kişisel bilgisayarda ve bir ana çatıda (mainframe) çalışabilir. Ayrıca Linux, üniversite ve araştırma merkezlerinde süper bilgisayarlar kurmak ve gruplama (clustering) yapmak için de kullanılmaktadır.

3.1.4 Güvenlik

Evans Data Summer 2004 Linux Development tarafından yapılan araştırmada görüşülen kullanıcıların %92'sinin Linux işletim sistemlerinde hiç virüs sorunu ile karşılaşmadıkları tespit edilmiştir. %7 civarında kullanıcının da üç veya daha fazla korsan (hacker) saldırısına uğradığı gözlemlenmiştir. Aynı araştırmada diğer işletim sistemi kullanıcılarının %60'nın güvenlik açıklarından dolayı zarar gördükleri belirtilmiş %32 oranındaki kullanıcının da üç veya daha fazla korsan saldırısına uğradığı ortaya konmuştur [5].

Yapılan bir diğer araştırma ise Windows 2000 kullanıcılarının %12'sinin, Windows 2003 kullanıcılarının bile sadece %18'nin düşüncesi güvenlik konusunda Windows'un Linux'a rakip olacak düzeye geldiğini belirtiyor. Aynı konudaki başka bir araştırmada ise katılımcıların %84'ü de Linux'un AIX ve Solaris gibi yüksek güvenliğe sahip işletim sistemlerine yakın olduğunu ve tüm Windows ailesi işletim sistemlerinden daha güvenli olduğunu düşündüklerini belirtmişlerdir [5].

3.1.5 Maliyet

Linux işletim sistemi ve üzerinde çalışan birçok uygulama GNU General Public License ile birlikte bedava temin edilebilmektedir. İstenirse uygun bir ücret karşılığında farklı üreticiler tarafından geliştirilen standart ve profesyonel sürümleri satın alınabilmektedir. Diğer işletim sistemlerinin yüksek maliyeti incelendiğinde bu maliyet avantajı Linux'un tercih edilmesine neden olmaktadır.

Örneğin; web, mail, veritabanı sunucusu ve program geliştirme platformu olarak kullanılacak bir sunucu kurulumunda Windows 2003 Server Enterprise Edition ve Redhat Enterprise Linux'un satın alma maliyetleri Çizelge 3.1'de verilmiştir.

Çizelge 3.1 Linux ve Windows toplam sahip olma maliyetleri karşılaştırması

	Windows 2003 Server EE	**RedHat Enterprise Linux**
İşletim Sistemi	1325 $ - 20 istemci	349 $ Basic, 799 $ Standart Sınırsız İstemci
Mail Sunucusu	1000 $	İşletim Sistemi ile Birlikte - Sınırsız istemci
Web Sunucusu	İşletim Sistemi ile Birlikte	İşletim Sistemi ile Birlikte
Veritabanı Sunucusu	3510 $ - 20 istemci	İşletim Sistemi ile Birlikte - Sınırsız istemci
Yazılım Geliştirme Paketi	975 $	İşletim Sistemi ile Birlikte

Tüm bileşenleri içeren bir sunucunun Windows 2003 Server EE ile kurulumu 5810 dolara mal olurken RedHat Enterprise Linux ile kurulumu Standart Edition ile 799 dolara mal olmaktadır.

İşletim sisteminin versiyonunun yükseltilmesi işleminde de Linux işletim sistemi diğer işletim sistemlerine göre daha ucuzdur. Örneğin Windows işletim sisteminin yeni sürümünü almak için her lisans, işletim sisteminin orijinalinin fiyatının yaklaşık yarısına gelirken, Linux'un yükseltilmesi işlemi ise sınırsız lisans için sadece işletim sisteminin yeni sürümünün yüklenmesi ile çözümlenmektedir.

Linux işletim sistemi oldukça düşük yapılandırmalar üzerinde çalışabilir. Windows 2003 Server kurulum için, Pentium uyumlu en az 133 MHz bir işlemci, tercihen 256 MB olmakla beraber minimum 128 MB hafıza birimi ve en az 1GB'ı boş olan 2GB'lık disk birimine ihtiyaç duymaktadır. Buna karşın RedHat Enterprise Linux işletim sistemi kurulum için en az, tercihen Pentium tabanlı olmakla birlikte, minimum i486 işlemci, önerileni 128 MB olan 64 MB hafıza birimi ve 800 MB disk birimine ihtiyaç duymaktadır.

Sonuç olarak birçok kurum Linux işletim sistemini tercih ederek harcamalarında büyük miktarda kısıtlamalara gitmektedir.

3.1.6 Açık Kaynaklı Kod Özelliği

Linux'un araştırma dünyasında çok tercih edilmesinin en önemli nedeni açık kaynak kodlu oluşudur. Birçok araştırmacı, önemli oranda tercih edilen bir işletim sistemi üzerinde ve/veya işletim sistemine eklenecek modüller tasarlayarak bilim dünyasına katkılar sağlayıp yeni ufuklar açabilmektedir.

Linux'un açık kaynak kodlu oluşu araştırmacıların özellikle Internet üzerinde sorunlarını paylaşıp çözerek grup çalışmaları yapmalarına olanak vermektedir.

3.1.7 Bilişim Dünyasında Yaygınlık

IBM'in verdiği rakamlara göre yaklaşık 200 devlet; örneğin Fransa, İngiltere, Amerika vb. maliyetleri düşürebilmek, işyükünü dengeleyip verimliliği arttırabilmek için platform olarak Linux kullanmaya başlamıştır. Sadece 2003'te Rusya, İngiltere, Brezilya, Bahreyn, Romanya hükümetleri Linux ile ilgili büyük projelerini açıklamıştır [6].

International Data Corporation (IDC)'nin yaptığı tahminlere göre Linux'un sunucu gelirlerinde %15 market payına sahip olacağı ve %38 paya sahip olacak lider Windows ailesinin en yakın takipçisi olacağı beklenmektedir. Diğer işletim sistemlerini kullanan eski sunucuların da önemli bir bölümünün Linux işletim sistemi kullanacağı hesaplandığında %15'lik payın %28'lere kadar çıkması bekleniyor.[Abramson, 2004]

IBM'in yaptığı bir başka tahmine göre 2002-2007 yılları arasında Windows %5,4 ile büyürken Linux %9 oranında büyüme sağlayacaktır [6].

3.2 Linux İşletim Sisteminde IPv4 Desteği

Linux çekirdeği, IPv4 protokolünü, birbirine bağlı fonksiyonlar kullanarak gerçekler. IP katmanı, IP işlevlerini yerine getiren koddan oluşur. Bu kod, gelen bilgiye IP bilgilerini ekler ve gelen paketin TCP veya UDP protokollerinden hangisine gönderileceğine karar verir. Kod, giden paket içinse gerekli IP başlık bilgilerini paketin başına ekleyip paketin alt katmana iletilmesini sağlar. IP işlevlerinin gerçeklenmesi, alt katmanda kullanılan ağ cihazları sayesinde olur. Ağ cihazları her zaman fiziksel olmayabilir. Linux'ta kullanılan 'loopback' cihazı tamamen yazılım yardımıyla gerçeklenmektedir.

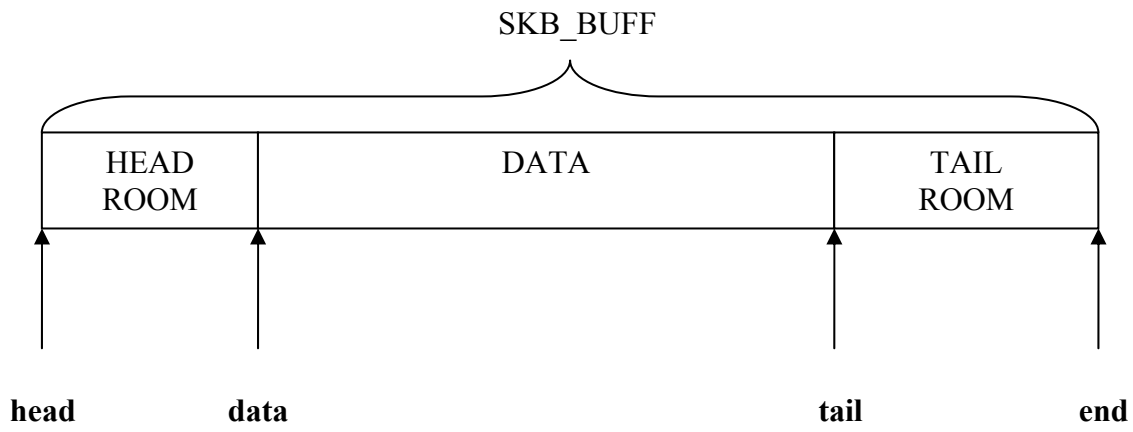
3.2.1 Linux'ta Ağ Yönetiminde Kullanılan Veri Yapısı

Linux çekirdeği, katmanlar arasında bilgi taşımasını kolaylaştırmak için soket bilgisini tek bir veri yapısında tutmaktadır. Tüm protokollere ait bilgiyi içeren bu yapı *sk_buff* veri yapısıdır. Gönderilen ve alınan her paket bu veri yapısı kullanılarak yönetilir. *sk_buff* dört temel işaretçi tutmaktadır.

- **head:** *skb_buff*'ın tuttuğu verinin başlangıç adresidir.
- **data:** Protokol bilgisinin başladığı adrestir. *sk_buff*'ı kullanan protokole göre değişir.
- **tail:** Protokol bilgisinin bittiği adrestir. *sk_buff*'ı kullanan protokole göre değişir.
- **end:** *sk_buff*'ın tuttuğu verinin bitiş adresini gösterir.

Yukarıda belirtilen işaretçilerin dışında, *sk_buff* yapısında, taşıma, ağ ve veri bağı katmanlarına ait bilgiler de tutulmaktadır [7]. Bunlardan bazıları aşağıda belirtilmiştir. sk_buff'ın genel yapısı Şekil 3.1'de gösterilmiştir.

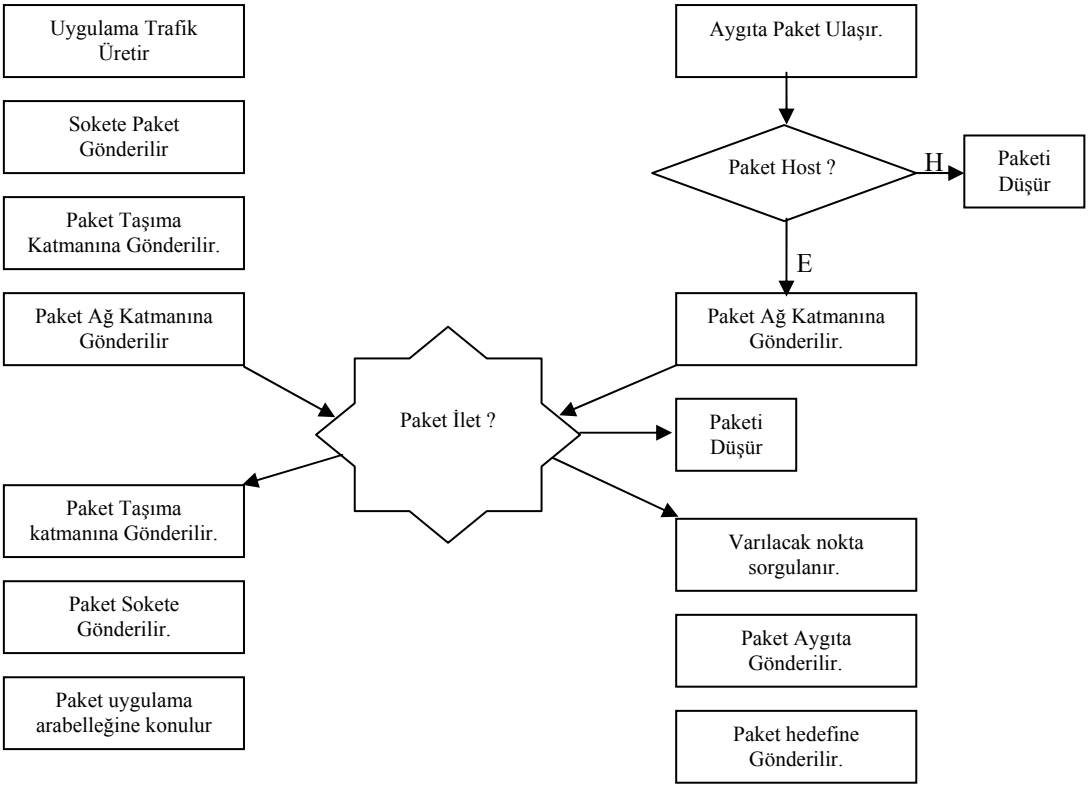
- **stamp:** paketin ulaştığı zamandır.
- **dev:** Alan ya da gönderen aygıtı işaret eder.
- **h:** Taşıma katmanı başlık işaret eder.
- **nh:** Ağ katmanı başlık bilgisini işaret eder.
- **mac:** Veri bağı katmanını gösterir.
- **len:** Gerçek veri uzunluğunu gösterir.
- **csum:** Veriye ait kontrol bilgisidir.
- **protocol:** Verinin hangi protokole ait olduğunu tutar.



Şekil 3.1 sk_buff veri yapısı [7]

3.3 Linux Çekirdeğinde Ağ Paketinin İzlediği Yol

Bir ağ paketinin Linux 2.4 çekirdeğindeki adımları Şekil 3.2'de gösterilmiştir (Kossak, 1999).



Şekil 3.2 Gelen ve giden paketlerin izlediği yol

Linux çekirdeğinde gelen ve giden paketler için gerçekleşen adımlar aşağıdaki bölümlerde anlatılmaktadır.

3.3.1 receive Kesmesi

Sistemdeki ağ kartı, sistemin Medium Access Control (MAC) adresine gelen veya yayınlanan (broadcast) bir paket alındığında bir kesme çağırır. Ağ sürücüsü paketi hafızasına alır. Daha sonra *sk_buff* tipinde bir yapı oluşturulur ve protokolden bağımsız olan cihaz destek rutinleri çağrılır. *sk_buff* yapısına zaman bilgisi konulur ve *sk_buff*, paket tipini işleyecek olan işlemci için sıraya konulur. *sk_buff* sıraya yerleştirildikten sonra __cpu_raise_softirq kesmesi çalıştırılmak üzere işaretlenir. Bundan sonra receive kesmesi işini bitirir.

3.3.2 Ağ Paket Alma Softirq'su

Ağ paket alma kesmesi, net_init, /net/core/dev.c dosyasında tanımlanmıştır. Paket üzerindeki diğer işlemler, do_softirq tarafından çağrılan ağ paket alma kesmesinde gerçekleşmektedir. do_softirq, çekirdek içinde üç yerde çağrılmaktadır.

- arch/i386/kernel/irq.c dosyasında tanımlanan *do_IRQ* fonksiyonunda,
- arch/i386/kernel/entry.S dosyasında,
- kernel/sched.c dosyasındaki çizelgeleme (schedule) fonksiyonunda.

Eğer program bu üç noktadan birinden geçerse, hem *do_softirq* hem de ağ paket alma kesmesi olan *net_rx_action* fonksiyonu çağrılır. Burada *sk_buff*, işlemcinin bekleme kuyruğundan alınır ve uygun paket işleyici fonksiyonuna gönderilir. Uygun paket işleyici fonksiyonu, çekirdekte yer alan *ptype_base* hash tablosu kullanılarak belirlenir. Bu tablo, paket tipleri temel alınarak oluşturulmuştur. Gelen paketin tipi bu tablodaki değerlerle eşlenir ve uyan paket tipine ait işleyici fonksiyon çağrılır. Paket birden fazla paket tipine uyuyorsa pakete ait *sk_buff* yapısının kopyası çıkarılır ve her işleyici fonksiyona ayrı ayrı gönderilir.

3.3.3 IP Paket İşleyicisi

IP paket işleyici fonksiyonu net/ipv4/ip_input.c dosyasında tanımlanan *ip_rcv* fonksiyonudur. Temel kontroller yapılır ve pakete ait checksum hesaplanır. Kontroller sonucu hatalı olduğuna karar verilen paket bu fonksiyonda düşürülür. Paket, kontrolleri başarı ile geçtiği takdirde, IP paketinin boyutu hesaplanır ve donanımın eklemiş olabileceği bilgiler skb'den çıkarılır.

Tüm işlemlerden sonra, *ip_rcv_finish* fonksiyonuna ait işaretçi parametre olarak verilerek NetFilter hook'u çağrılır. Bu hook *ip_rcv_finish* fonksiyonunu çağırır. Bu fonksiyon içinde, *ip_route_input* çağrılarak paketin hedefi belirlenir. Sonuca göre paket aşağıdaki dört fonksiyondan birine gönderilir.

- *ip_local_deliver*: Paket sistemin kendisine gelmiştir.
- *ip_forward* : Paket yönlendirilecektir.

- *ip_error* : Paketin yönlendirileceği bir adres bulunamamış ve bir hata oluşmuştur.
- *ip_mr_input* : Paket birçoğa gönderim (multicast) paketidir ve çoğa gönderim yönlendirme yapılır.

3.3.4 Ipv4 Kesmeleri

Netfilter tarafından tanımlanan 5 adet kesme bulunmaktadır.

- NF_IP_PRE_ROUTING : Paket ip_rcv fonksiyonundan alındıktan sonra yönlendirilmeden önce çağrılır.
- NF_IP_LOCAL_IN : Yönlendirme kararı verildikten sonra paket konakçıya (host) ait ise çağrılır.
- NF_IP_FORWARD : Paket için başka bir ara yüz belirtilmişse çağrılır.
- NF_IP_LOCAL_OUT : Paket yerel görev tarafından üretilmişse ve dışarı gönderilecekse çağrılır.
- NF_IP_POST_ROUTING : Gönderilecek paket hattan çıkmadan önce çağrılır.

Çağrılan kesmeler işlerini bitirip belli değerler döndürürler. Dönüş değerleri aşağıdaki gibidir.

- NF_DROP : Paketi düşürme anlamına gelir. Paket için ayrılan tüm kaynaklar bırakılır.
- NF_ACCEPT : Paketin kabul edilmesi anlamına gelir. Paket bir sonraki ağ adımına gönderilir.
- NF_STOLEN : Paket artık Netfilter tarafından değerlendirilmez. Fakat paket için ayrılan kaynaklara dokunulmaz.
- NF_QUEUE : Paket kullanıcı seviyesi için sıraya koyulur.
- NF_REPEAT : Kesme tekrar çağrılır.

3.3.5 Paketin Başka Bir Cihaza Yönlendirilmesi

Yönlendirme fonksiyonu paketin başka bir cihaza yönlendirilmesine karar verdiyse, net/ipv4/core/ip_forward.c dosyasındaki *ip_forward* fonksiyonu çağrılır. Bu fonksiyon ilk

olarak paketin Time To Live (TTL) değerini kontrol eder. TTL değeri "1" değerinden küçük veya "1" değerine eşitse paket düşürülür ve gönderene ICMP kullanılarak zaman aşımı mesajı gönderilir.

Paket düşürülmediyse, *sk_buff* 'da hedef cihazın veri bağı katmanı bilgisi için yer olup olmadığı kontrol edilir. Eğer yeterince yer yoksa *sk_buff* yapısının boyutu artırılır ve TTL değeri bir azaltılır. Yeni oluşan paketimizin boyutu hedef cihazın Maximum Transmission Unit (MTU) değerinden büyük ve IP bilgisindeki 'parçalara ayırma' biti 1 ise, paket düşürülür ve gönderene parçalara ayırdığına dair ICMP mesajı gönderilir.

Bundan sonra yeni NetFilter hook'u çağrılır. Bu hook, *ip_forward_finish* fonksiyonunu çağırır. *ip_forward_finish*, IP bilgisi içindeki IP seçeneklerini atar ve *ip_send*'i çağırır. Paketin parçalara ayrılması gerekirse bu işlem gerçekleştirilir ve *ip_finish_output* çağrılır. Bu fonksiyon da *ip_finish_output2*'yi çağırır. *ip_finish_output2*, donanım bilgisini *sk_buff*'a ekler ve *ip_output*'u çağırarak paketin gönderilmesini sağlar.

3.3.6 send Kesmesi

ip_build_xmit() fonksiyonu IP başlık bilgilerini oluşturur. *ip_finish_output* ve *ip_finish_output2* donanım bilgisini *sk_buff*'a ekler ve *ip_output* çağrılarak paketin gönderilmesi sağlanır. *dev_queue_xmit* gönderilecek paketleri işlemcinin bekleme kuyruğuna yerleştirir. send kesmesi sonucunda da paket ağ kartı tarafından gönderilir.

3.4 Linux İşletim Sisteminde Ağ Güvenliği

Ağ trafiği arttıkça bilgi transferi sırasında meydana gelecek saldırıların ihtimali de bir o kadar artmaktadır. Bu nedenle, günümüzde ağ trafiğinin hızlı artışı beraberinde işletim sistemlerinin güvenlik önlemlerini yenilemesine ve arttırmasına yol açmıştır. Linux işletim sistemi ağ güvenliğini sağlayabilmek için çeşitli güvenlik önlemleri geliştirmiştir. Bunlardan bazıları bu bölümde incelenmektedir.

Uygulamalar kendileri için belirledikleri kapıları (port) kullanarak iletişim yaparlar. Tehlikeler genelde önlem alınmayan bu açık kapılardan gelir. Bu nedenle Linux işletim sistemi kapıların detaylı bir şekilde yönetilmesine olanak verir. Örneğin /etc/hosts.allow ve

/etc/hosts.deny dosyalarındaki kayıtlar hangi kapıların hangi IP adresleri için kullanımda olduğunu belirler (Reilly, 2003).

Bir diğer önlem ise verinin hat üzerinde şifreli olarak iletimine yöneliktir. Uygulamaların gönderdikleri veriler de hat üzerine çıktıklarında artık güvende değildir. Bu veriler başkası tarafından okunabilir veya değiştirilebilir. Veriyi güvenlik altına alabilmek için SSH protokolü geliştirilmiştir. Transfer edilen veri kriptografik algoritmalarla şifreleme/deşifreleme yapılarak önemli bilgilerin başkaları tarafından erişilmesi engellenmektedir. OpenSSL kütüphanesi kullanılarak özellikle web işlemleri gizlilik içinde yapılmasına rağmen birçok kapıdan yapılan veri transferi şifreli olarak gerçekleşmemektedir [8].

Linux'ta uygulamalar için kullanılan bir başka güvenlik açığı tespit yöntemi de her uygulama için tutulan log dosyalarıdır. Bu dosyalar yardımıyla sisteme yapılan atakların tespit edilme olasılığı yüksektir. Ethereal [9], Snort gibi programlarla paketler incelenebilir, kapılar taranarak ağdan yapılacak saldırılar takip ve/veya tespit edilebilir.

Ağ üzerinden gelebilecek tehlikeleri önlemek için Linux güvenlik duvarı (firewall) kullanmaktadır. Dışarıdan ve içeriden yapılan tüm veri transferleri kontrol altında gerçekleşir. Sadece belirlenen kapılar belirlenen IP adresleri veri alışverişi gerçekleştirebilir (Reilly, 2003).

4. TASARIM

Güvenli veri iletişimi ile ilgili yapılan çalışmalar ve geliştirilen uygulamalar 2. bölümde incelenmiştir. Bu tez çalışmasında, daha önceki uygulamaların eksik ve zayıf yönleri ele alınarak, aşağıdaki önemli noktaların ışığında yeni bir mimari tasarlanmıştır.

1. Sistem gelen ağ bağlantılarını kontrol edebilmelidir.
2. Sistemin amacına uygun çalışabilmesi için kurallar tanımlanabilmelidir.
3. Tanımlanan kurallar kolayca değiştirilebilmelidir.
4. Hem üst düzey hem de alt düzey kullanıcı kuralları zorlanmadan yönetebilmelidir.
5. Gelen ve giden ağ paketleri önceden tanımlanan kurallara uygun şekilde kriptografik işlemlerden geçirilebilmelidir. Kriptografik işlemlerden dolayı meydana gelebilecek gecikme minimum seviyede tutulmalıdır.
6. Sistem sadece tanımlı kurallara uygun paketlere müdahale etmelidir.
7. Sistemdeki kural değişiklikleri servisin durdurulması ve tekrar çalıştırılması ile kolayca devreye alınabilmelidir.

Yukarıdaki noktaları sağlayacak şekilde bir tasarım temel olarak dört ayrı modül oluşturulmuştur.

1. Sistemin Başlatılması
2. Ayrıştırma
3. Kriptografik İşlemler
4. Sistemin Durdurulması

4.1 Sistemin Başlatılması

Sistemin başlatılması aşamasında, ayrıştırma ve kriptografik işlem fonksiyonlarının yerine getirilebilmesi için gereken başlangıç işlemleri gerçekleştirilmelidir. Bu aşamada yapılması gereken işlemler aşağıda belirtilmiştir.

- Genel kullanıma açık veri yapılarının oluşturulması
- Yapılandırma dosyalarının okunması. Hata varsa hatanın kullanıcıya aktarılması. Hata yoksa okunan verilerin veri yapılarında saklanması
- Kriptografik algoritmalar için gerekli anahtar değerlerinin yüklenmesi

- Gelen ve giden paketleri alıp değerlendirecek yeni paket tipinin kaydedilmesi
- Mevcut paket tipinin yedeklenip yerine tanımlanan yeni paket tipinin kaydedilmesi

Bu işlemler ayrıntılı olarak sonraki bölümlerde incelenmiştir.

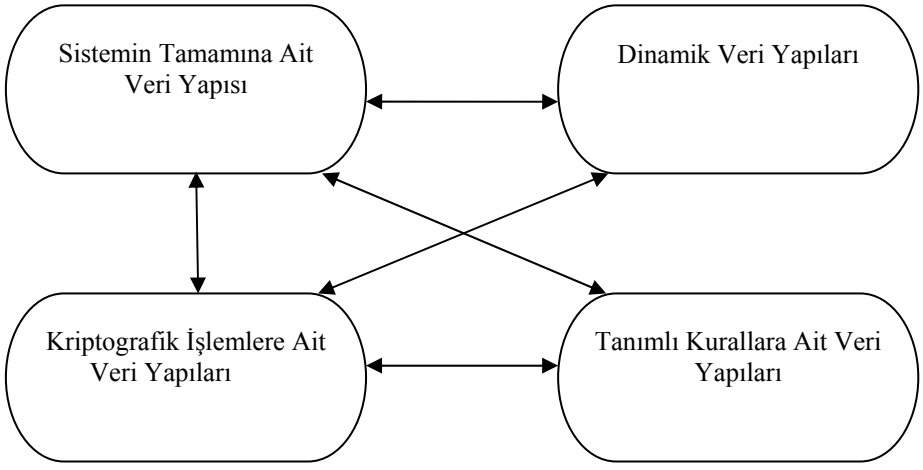
4.1.1 Genel Kullanıma Açık Olan Veri Yapılarının Oluşturulması

Sistemin, gelen ve giden paketler ve tanımlı kurallar üzerinde işlem yapacağı, bu yapıların değerleri üzerinde değişiklik yapacağı göz önüne alınırsa, kullanılacak değerlerin genel veri yapılarında tutulması zorunluluğu ortaya çıkar. Sistemin başlaması sırasında bu veri yapıları oluşturmuştur ve yeni paket tipi devreye alınmadan veri yapılarının doğruluğu sağlanmıştır.

Sistemde genel kullanıma açık olan dört temel veri yapısı oluşturulmuştur.

- Sistemin Geneline Ait Veri Yapısı
- Tanımlı Kurallara Ait Veri Yapıları
- Dinamik Veri Yapıları
- Kriptografik İşlemlere Ait Veri Yapıları

Dört temel yapı tanımlarının doğru yapılması sistemin doğru çalışması için mutlak şarttır. Temel yapı tanımlarının ilişkilendirme şeması
Şekil 4.1'de belirtilmiştir.



Şekil 4.1 Temel veri yapıları ve birbirleri ile ilişkileri

4.1.1.1 Sistemin Geneline Ait Olan Veri Yapısı

Sistem genelinde erişilebilecek ve çalışma esnasında her aşamada ihtiyaç duyulabilecek olan bilgileri içerecek şekilde tasarlanmıştır. Bu yapıdaki bilgilerin bir kısmı çalışmanın her esnasında değiştirilebileceği gibi, bir kısmı da sistemin başlatılması esnasında aldıkları değerleri sistem devreden çıkarılana kadar koruyabilmelidir.

Çalışma sırasında erişilmesi gereken bilgilerden bazıları şunlardır.

- Gelen toplam paket sayısı
- Giden toplam paket sayısı
- Şifrelenen toplam paket sayısı
- Deşifre edilen toplam paket sayısı
- Yapılandırma dosyalarının disk üzerindeki yerleri

4.1.1.2 Tanımlı Kurallara Ait Olan Veri Yapıları

Sistemde güvenli veri iletişimi için temel alınacak kuralların tanımı, sistemin başlaması esnasında ilgili veri yapılarına alınmıştır. Bu kurallar, gelen ve giden paketlerin kriptografik işlemlerden geçmesi aşamasında kullanılan kuralların oluşturulmasında kullanılmıştır.

Kuralların tanımında, bir bağlantıyı ayırt edebilecek tüm bilgiler kullanılmıştır. Kural tanımında bağlantıyı tanımlayacak bilgilere ek olarak, bağlantı için kullanılacak kriptografik yöntem, çalışan uygulamanın istemci/sunucu modunun belirtildiği bilgi, kuralın geçerli olup olmadığını belirten değer bulunacak şekilde tasarım yapılmıştır.

Kuralların sayısı önceden bilinmediği için dosyadan okunan kural bilgilerinin linkli liste yapısında tutulması daha sonra da eş olan kurallar silinerek yeni bir kural listesine aktarılması sağlanmıştır.

4.1.1.3 Dinamik Veri Yapıları

Sistemin çalışması sırasında gelen ve giden paketler değerlendirilerek yeni kuralların eklenebileceği veri yapıları tasarlanmıştır. Bu veri yapıları sistemin dinamik olarak hizmet

verebilmesini sağlamıştır. Örneğin sistem bu veri yapıları sayesinde yapılandırma dosyasında bulunmayan kapıların güvenli veri iletişimi için kullanılmasını sağlar.

4.1.1.4 Kriptografik İşlemlere Ait Veri Yapıları

Güvenli veri iletişimi için kullanılacak kriptografik algoritmaların ihtiyaç duyduğu veri yapıları tasarlanmıştır. Algoritmalara ait anahtar değerlerinin tutulduğu ve/veya algoritmaya ait özel bilgilerin saklandığı bu veri yapıları kriptografik işlemlerin sorunsuz çalışmasını sağlar.

4.2 Ayrıştırma

Ağ katmanında çalışan güvenli bir iletişim protokolünün ana görevi gelen ve giden paketleri tanımlı kurallardan birisi ile eşleştirmektir. Eşleştirme işlemi sonucunda paket ya uygun kriptografik işleme gönderilir ya da işlem görmeden yoluna devam eder.

Ayrıştırma işleminde önem verilecek noktalar aşağıdaki gibidir.

- Gelen ve giden paketlerin tanımlı kurallarla karşılaştırma işlemi en kısa sürede gerçekleşmelidir.
- En kısa sürede ayrıştırmayı sağlamak amacıyla kurallar sırası ile IP adresi, protokol ve port bilgisine göre değerlendirilmelidir.
- Ayrıştırma sonucu kriptografik işleme girmeyecek paketler çekirdek içerisindeki yollarına devam etmelidir.

Ayrıştırma işlemi sırasında bağlantılara ait bilgilerin mevcut kurallarla karşılaştırma süresinin mümkün olduğunca düşük olmasına dikkat edilmiştir. Ayrıştırma işleminin yavaş olması gelen ve giden paketlerin kuyrukta birikmesine ve veri iletişiminin yavaşlamasına yol açar. Bir pakete ait kural mevcut ise paket mevcut kurala ait kriptografik işleme yönlendirilecek şekilde tasarlanmıştır.

4.3 Kriptografik İşlemler

Ayrıştırma işleminde gerçekleşen eşleşmeye göre gelen paketler tanımlı algoritmaların şifreleme fonksiyonlarına, giden paketler de deşifreleme fonksiyonlarına yönlendirilirler.

Sistemde kullanılacak kriptografik algoritmaların ve kütüphanelerinin önceden tanımlanmış olması gerekir.

Şifreleme/deşifreleme işlemlerinde algoritmalar tarafından üretilen anahtar değerleri kullanılmıştır. Giden ve gelen paketler için uygulanan adımlar Çizelge 4.1'de belirtilmiştir.

Çizelge 4.1 Giden ve gelen paketler için uygulanan adımlar

Adım	**Giden Paketler**	**Gelen Paketler**
1	Anahtar değerinin üretilmesi	Anahtar değerinin üretilmesi
2	Paketteki bilginin bloklar halinde şifreleme fonksiyonundan geçmesi	Paketteki bilginin bloklar halinde deşifreleme fonksiyonundan geçmesi
3	IP başlık bilgisindeki güvenlik alanına gerekli değerin verilmesi	Paketin serbest bırakılması
4	Paketin serbest bırakılması	

5. UYGULAMA

Sisteme ait uygulama, tasarımda ortaya çıkan dört aşamaya ait fonksiyonların tanımlanıp gerçekleştirilmesi sonucu ortaya çıkmıştır. Bu aşamalara ait uygulama ayrıntıları aşağıda anlatılmıştır. Uygulamada kullanılan fonksiyonlar, veri yapıları ve dosya formatları 6., 7. ve 8. bölümlerde açıklanmıştır. Sistemin uygulama ayrıntılarından önce, seçilen uygulama geliştirme metodu olan "Yüklenebilir Çekirdek Modülleri" (LKM) anlatılmıştır.

5.1 Geliştirme Metodu

Hedeflenen sistemin, sisteme gelen bütün bağlantıları ele geçirebilmesi, işleyebilmesi ve gerektiği takdirde paketlerin iletilip iletilmemesine karar verebilmesi gerekmektedir. Bu işlemlerin C soket kütüphanesi kullanılarak gerçekleştirilmesi, hem uygulamaya getireceği yük açısından, hem de çekirdekte çalışmanın getireceği avantajları kullanamama açısından uygun değildir. Bu yüzden kullanabilecek 2 adet seçenek vardır:

1. Çekirdek kodunun amaca uygun olarak değiştirilmesi,
2. Yüklenebilir çekirdek modüllerinin kullanımı.

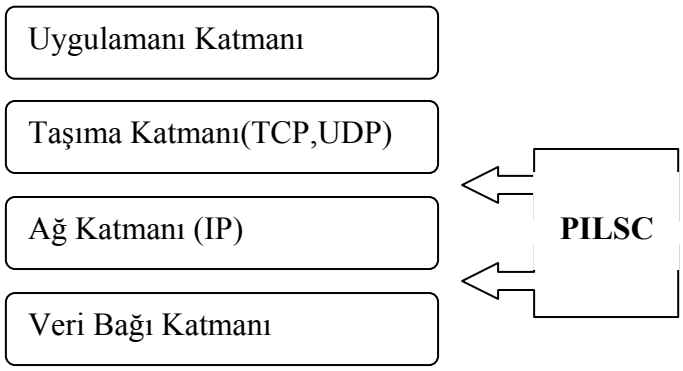
Yüklenebilir çekirdek modülünün getireceği avantajlar şu şekilde sıralanabilir (Henderson, 1999)(Pragmatic, 1999).

1. Derleme ve hata izleme kolaylığı
2. Zaman tasarrufu
3. Kod taşınabilirliği
4. Sistemin tekrar başlatılmasına gerek kalmayıp her kod değişikliğinde sadece modülün derlenmesi ve yüklenmesi.
5. Modülün bir servis gibi istenildiğinde devreye alınması veya devreden çıkarılabilme kolaylığı
6. Modülün gerekli olmayan durumlarda çalıştırılmayarak hafıza tasarrufu sağlanması
7. Yüklenebilir modüllerin çalışma hızı ile temel çekirdek modüllerinin çalışma hızı aynı olması

Yukarıda bahsedilen avantajlarından dolayı sistemin geliştirilmesi için yüklenebilir çekirdek modülleri tercih edilmiştir.

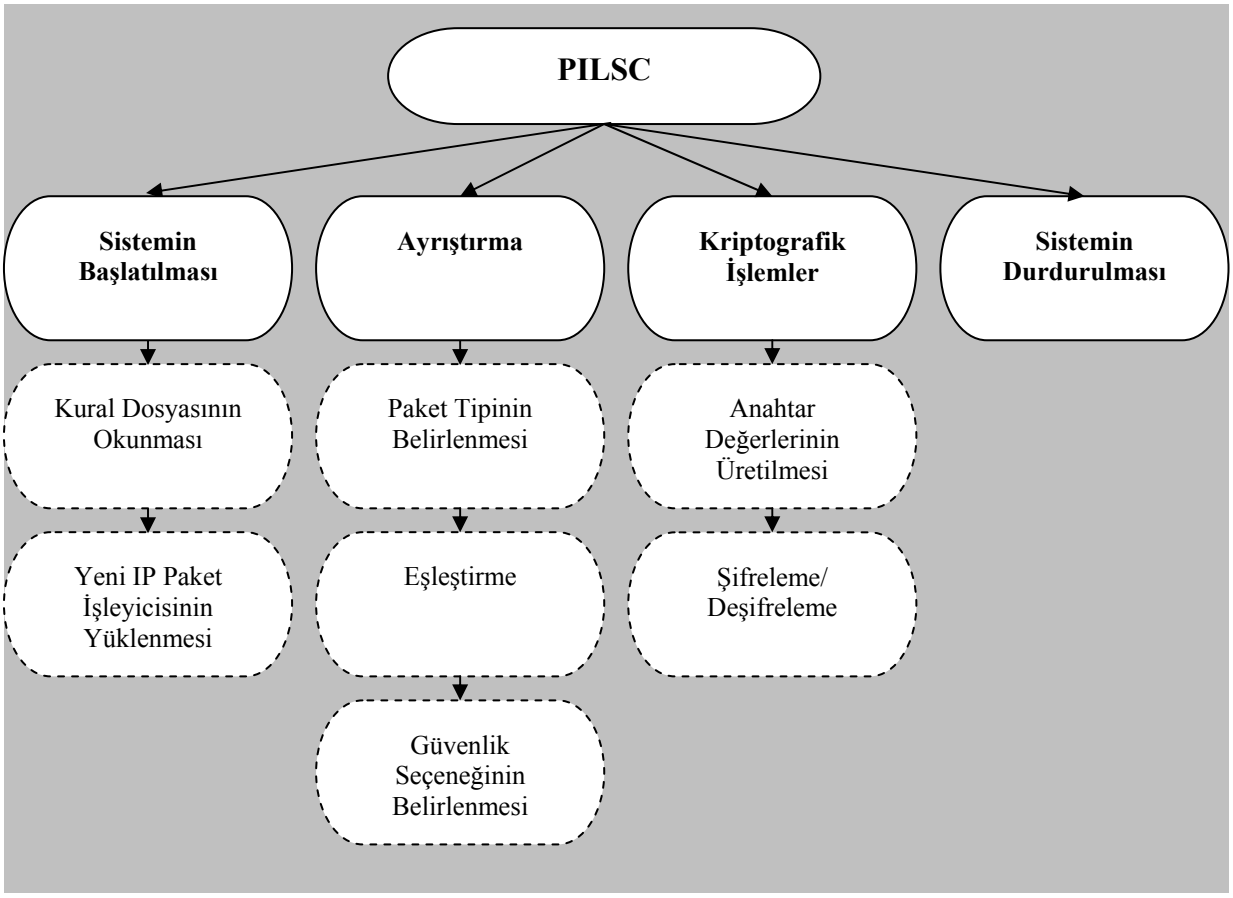
5.2 PILSC Sisteminin Temel Yapısı

PILSC Şekil 5.1'de gösterildiği gibi ağ katmanında çalışacaktır. Tüm IP paketlerini değerlendirebilen PILSC ağ katmanının içinde çalışarak güvenli veri iletişimini yönetmektedir.



Şekil 5.1 PILSC'in OSI katmanındaki yeri

Uygulama sırasında, tasarlanan dört ana modül gerçeklenmiştir. Bu modüller, kendi içlerinde de başka ufak modüllerden oluşmaktadır. Modül şeması Şekil 5.2'de gösterilmiştir.



Şekil 5.2 Uygulamanın temel ve ara modülleri

5.3 Sistemin Başlatılması

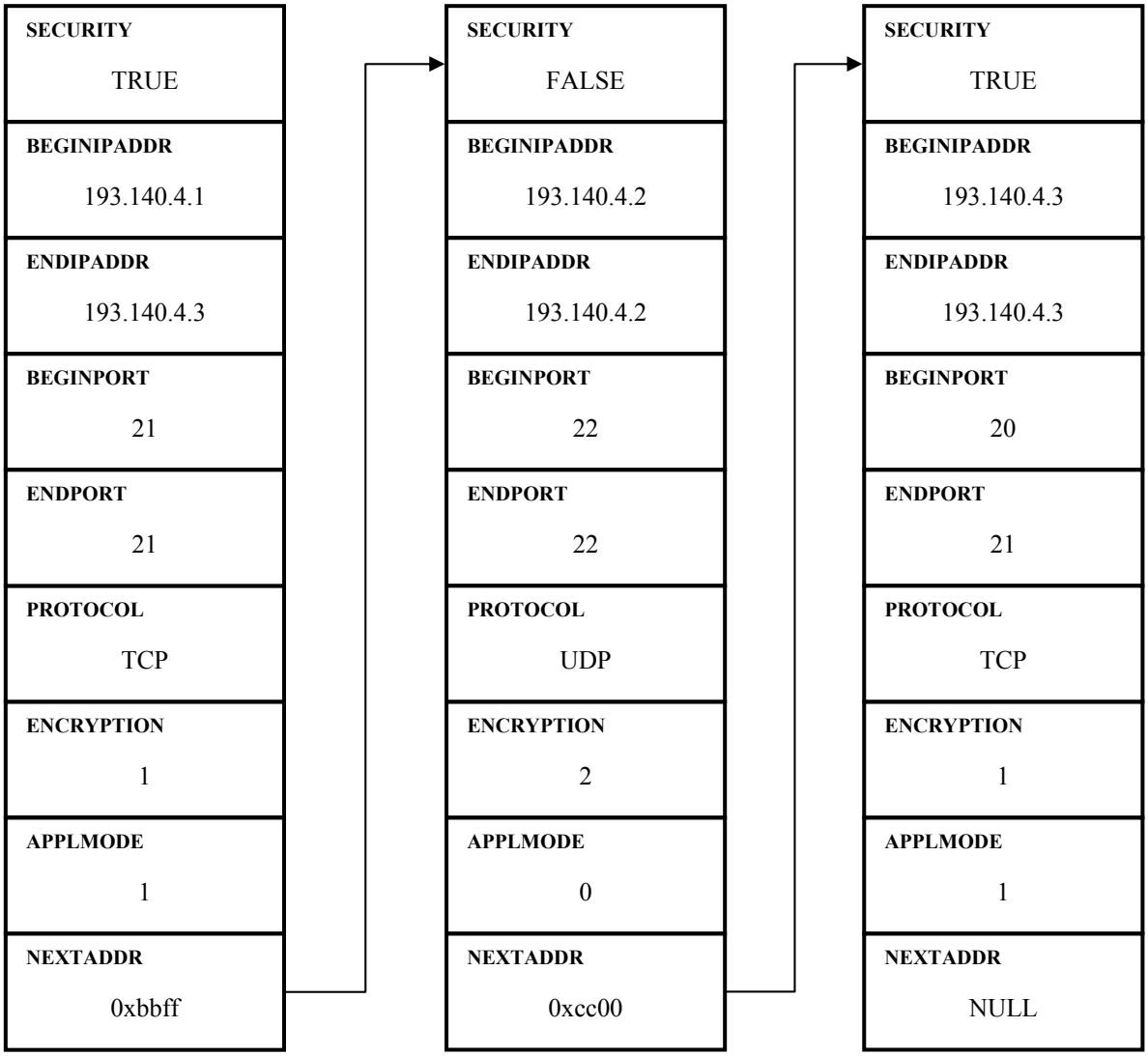
Sistemin başlatılması aşamasında, ilk gerçekleşen işlem kullanılacak veri yapılarının oluşturulmasıdır. Kullanılacak veri yapılarına ait bilgi yapılandırma dosyasında tanımlanmış olup, bu dosyanın okunup parçalara ayrılması sonucu, tasarlanan veri yapılarına yerleştirilir.

5.3.1 Kural Dosyasının Okunması

Gelen ve giden paketlerin değerlendirilmesi için kural dosyasının doğru yorumlanması gerekmektedir. İşlem akışına geçmeden önce kural dosya yapısını oluşturmak için kullanılan pakete ait TCP ve IP protokol bilgilerinden bahsedilmektedir. Bu bilgiler aşağıda listelenmiştir.

- **Kaynak IP Adresi :** Paketin geldiği kaynak adrese göre deşifreleme işlemini gerçekleştirmek için ayrıştırmada kullanılır.
- **Hedef IP Adresi :** Paketin gideceği hedef adrese göre şifreleme işlemini gerçekleştirmek için ayrıştırmada kullanılır.
- **Port :** Belirli bir servise ait veri iletişimin gerçekleştiği kapılardır. Kriptografik işlemleri gerçekleştirmek için paketin tipine ve uygulamanın moduna göre hedef ya da kaynak kapı ayrıştırmada kullanılır.
- **Protokol :** IP paketinin ait olduğu protokole göre şifreleme/deşifreleme yapmak için kullanılır.

Kurallara ait dosyanın okunmasında yapılan işlem her satırın okunup satırdaki bilgilerin hata kontrolünün yapılması ve kuralın *config* yapısına aktarılmasıdır. Yapılan kontroller sonucu, hatasız her kural config veri yapısında saklanır. *config* veri yapısının kural dosyasından bilgilerin aktarılmasından sonraki durumu Şekil 5.3'te gösterilmiştir.



Şekil 5.3 *config* veri yapısında bilgilerin saklanması

Dosya okuma işlemi tamamlandıktan sonra config veri yapısındaki kayıtların toplam sayısı, security alanı doğru (true) olmak şartı ile hesaplanır. *ruledb* veri yapısı için elde edilen sayıya bağlı olarak hafızada yer ayrılır. *config* veri yapısındaki bilgiler genişletme işleminden geçirilerek *ruledb* listesine aktarılır. Genişletme işlemi sırasında başlangıç ve bitiş arasındaki IP adresleri ile port adresleri birer kayıt olmak üzere aktarılır. *ruledb* veri yapısı eşleşme için kullanılacağı için kayıtlar tek tek tutulur. Genişletme ve sıralama işleminden sonra *ruledb* veri yapısının son hali Şekil 5.4'te gösterilmiştir.

IPADDR	IPADDR	IPADDR	IPADDR	IPADDR
193.140.4.1	193.140.4.2	193.140.4.3	193.140.4.3	193.140.4.3
PORT	PORT	PORT	PORT	PORT
21	21	20	21	21
PROTOCOL	PROTOCOL	PROTOCOL	PROTOCOL	PROTOCOL
TCP	TCP	TCP	TCP	TCP
ENCRYPTION	ENCRYPTION	ENCRYPTION	ENCRYPTION	ENCRYPTION
1	1	1	1	1
APPLMODE	APPLMODE	APPLMODE	APPLMODE	APPLMODE
1	1	1	1	1

Şekil 5.4 Genişletme ve sıralama işleminden sonra ruledb veri yapısının son hali

Daha sonraki sıralama ve temizleme işlemleri sırasında ruledb listesinde eş olan kayıtlar varsa silinir ve liste sıralı hale getirilir. Silme işlemi gerçekleştikten sonraki *ruledb* veri yapısının durumu Şekil 5.5'te gösterilmiştir.

IPADDR	IPADDR	IPADDR	IPADDR
193.140.4.1	193.140.4.2	193.140.4.3	193.140.4.3
PORT	PORT	PORT	PORT
21	21	20	21
PROTOCOL	PROTOCOL	PROTOCOL	PROTOCOL
TCP	TCP	TCP	TCP
ENCRYPTION	ENCRYPTION	ENCRYPTION	ENCRYPTION
1	1	1	1
APPLMODE	APPLMODE	APPLMODE	APPLMODE
1	1	1	1

Şekil 5.5 Temizleme işleminden sonra ruledb kural listesi

5.3.2 Yeni IP Paket Tipinin Oluşturulması ve Kaydedilmesi

Sistemin gelen ve giden paketler üzerinde işlem yapabilmesi ve tanımlanmış kuralları paketlere uygulayabilmesi için, paketlerin sistemin bir noktasında ele geçirilmesi gerekmektedir. Bunun için de Linux çekirdeği tarafından çekirdek programlamada

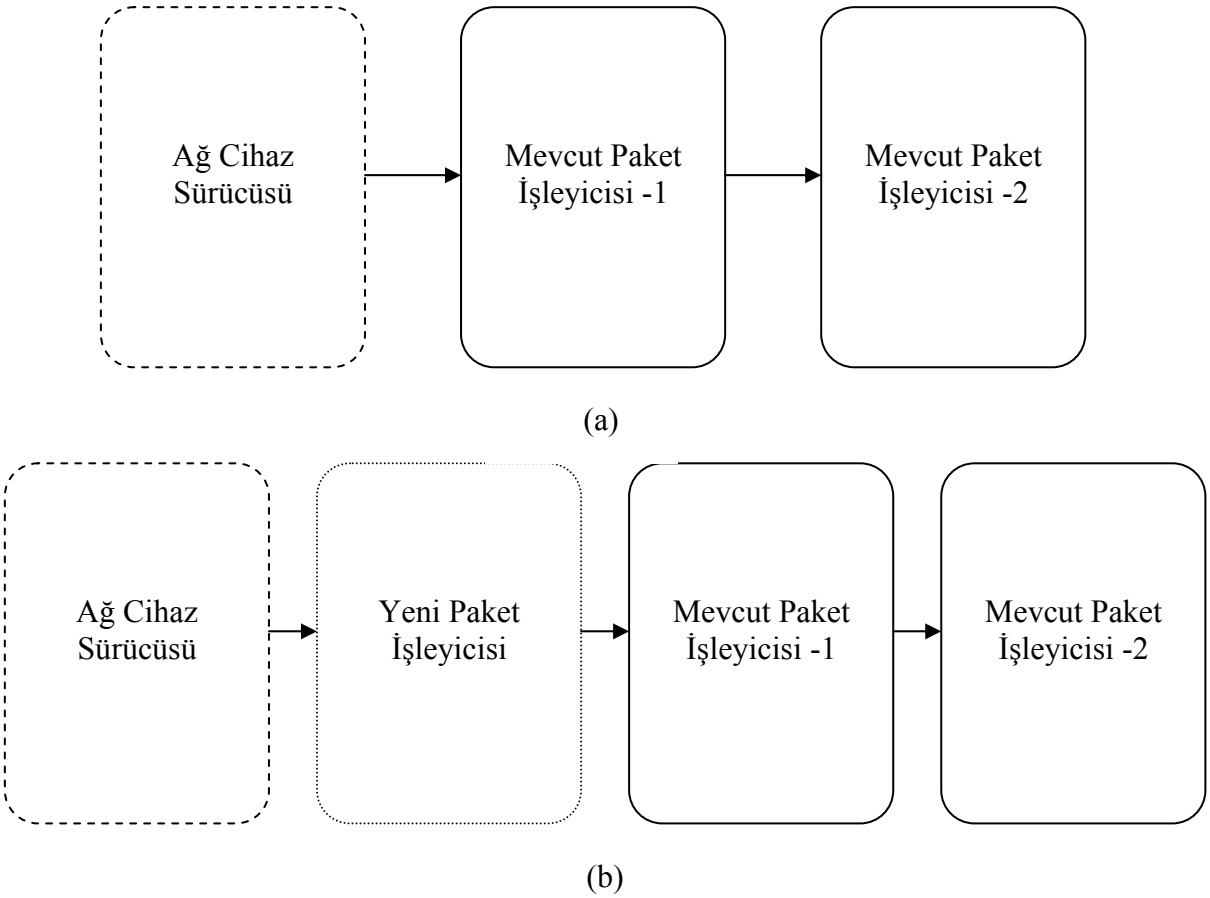
kullanılması için dışarı aktarılan fonksiyon işaretçilerinden ve veri yapılarından faydalanılmıştır.

Linux çekirdeği gelen ve giden ağ paketlerine, tiplerine göre servis verebilmek için, paket tiplerinin kaydedildiği bir dizi tutmaktadır. Kayıtlı olan her paket tipinde aşağıdaki bilgiler bulunmaktadır.

- Paketin tipi
- Paket tipinin ilişkilendirileceği ağ cihazına ait işaretçi
- Paket tipinde ağ paketi geldiğinde kullanılacak işleyici fonksiyona ait işaretçi

Paketlere, işlemci kuyruğunda müdahale edilebilmeli ve mevcut IP işleyici fonksiyonunun paketler üzerinde işlem yapması engellenmelidir. Sistem bu amaçla kendi IP paket tipini oluşturmalı, bu paket tipi için kendi işleyici fonksiyonunu bu paket tipi ile ilişkilendirip, yeni paket tipini çekirdeğe kaydetmelidir. Çekirdeğe yeni paket tipi kaydetme işlemi şu şekilde gerçekleşmektedir:

Öncelikle yeni paket tipine ait işleyici fonksiyon yazılmalı ve yeni paket tipi için packet_type tipinde bir veri yapısı oluşturulmalıdır. Daha sonra çekirdekte tanımlı olan *dev_add_pack* fonksiyonu yardımı ile yeni paket tipi mevcut paket tiplerinin tanımlı olduğu diziye eklenmelidir. Yeni kaydedilen paket tipi, cihaz sürücüsü ile aynı tipte daha önce tanımlanmış olan paket tipinin önüne yerleştirilmeli ve yeni paket işleyici fonksiyonu, mevcut paket işleyicisinden daha önce çalıştırılmalıdır. Bu yapı Şekil 5.6'da gösterilmiştir.



Şekil 5.6 (a) Mevcut paket işleyicilerinin durumu
(b) Yeni paket tipinin eklendikten sonraki paket işleyicilerinin durumu

Yeni oluşturulan IP paketine ait işleyici fonksiyon *newiprcv* fonksiyonudur. Yeni paket çekirdeğe kaydedildikten sonra gelen ve giden IP paketleri bu fonksiyon tarafından işlenecektir.

5.4 Paketlerin Ayrıştırılması

Gelen ve giden paketlerin kriptografik fonksiyonlarda işlem görebilmeleri için gerekli kontrol mekanizmalarından geçmeleri gerekmektedir. Bu kontroller şu şekilde sıralanabilir.

- Paketin tip kontrolü
- Gelen paket için IPv4 güvenlik alanının kontrolü
- Gelen ve giden paket için kural listesinin kontrolü
- Gelen ve giden paket için dinamik kural listesinin kontrolü

5.4.1 Paketin Tip Kontrolü

IP paket işleyicisine paket geldiğinde yapılan ilk işlem paketin gelen paket mi yoksa giden paket mi olduğunun belirlenmesidir. Bunun kontrolü için pakete ait *sk_buff* tipindeki, paketin protokol bilgisini içeren skb değişkeninin, *pkt_type* alanı kontrol edilmelidir. Bu alan ve alabileceği değerler aşağıda sıralanmıştır:

- PACKET_HOST : Paket konakçıya gelmiştir.
- PACKET_BROADCAST : Paket tüm konakçılara iletilecektir.
- PACKET_MULTICAST : Konakçının içinde bulunduğu konakçı grubuna iletilecektir.
- PACKET_OTHERHOST : Başka bir konakçıya iletilecektir.
- PACKET_OUTGOING : Dışarı gönderilecek bir pakettir.
- PACKET_LOOPBACK : MC/BRD frame looped back
- PACKET_FASTROUTE : Fastrouted frame

Paket bilgisi PACKET_HOST veya PACKET_OUTGOING ise paket eşleştirme işlemine yönlendirilir.

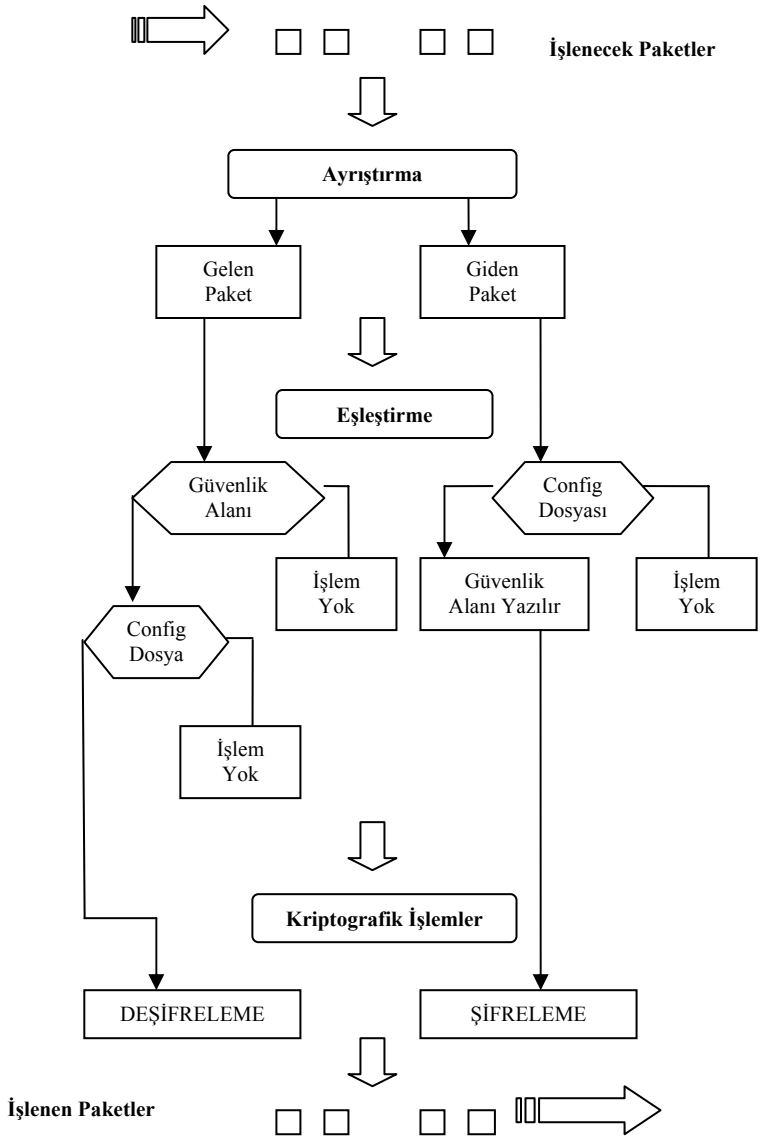
5.4.2 Gelen Paketler

Gelen paketlerin öncelikli olarak IPv4 güvenlik alanı kontrol edilir. Eğer doğru değer bulunursa, paket kural listesindeki kayıtlarla eşleştirilir. Eşleşme sağlanırsa paket deşifreleme fonksiyonlarından uygun olanına gönderilir. Kural listesinde herhangi bir kayıt ile eşleşme sağlanamazsa dinamik veri listesinin kayıt sayısına bakılır. Dinamik veri listesinde kayıt varsa bu kayıtlar arasında arama yapılır. Uygun eşleşme olursa paket deşifreleme fonksiyonlarından uygun olanına gönderilir.

5.4.3 Giden Paketler

Giden paketler öncelikle kural listesindeki kayıtlarla eşleştirilir. Eşleşme sağlanırsa IPv4 güvenlik alanına gerekli değer yazılır. Paket şifreleme fonksiyonuna aktarılır. Eşleşme sağlanamazsa dinamik veri listesinin kayıt sayısı kontrol edilir. Kayıt mevcut ise uygun eşleşme aranır, bulunursa şifreleme fonksiyonu çağrılır.

Gelen ve giden paketler için blok akış diyagramı Şekil 5.7'de verilmiştir.



Şekil 5.7 PILSC'in blok diyagramı

5.5 Tanımlı Kurallar İçinde Yapılan Arama

Kurallar içinde yapılan arama için "binary search" algoritması kullanılır. Paket bilgileri sırası ile kayıt listesindeki IP adresi, protokol bilgisi ve port numarası ile karşılaştırılır. Kural listesi sıralı tutulduğu ve "binary search" algoritması kullanıldığı için **n** adet uzunluktaki listede yapılan arama en kötü durum senaryosunda $\mathbf{log_2 n}$ adımda sonuçlanır. Bu uygulama eşleşme işleminin hızlanmasını sağlar.

5.6 Kriptografik İşlemler

Kriptografik işlemler paketin eşleştiği kayıttaki güvenlik seçeneğine ve paketin tipine göre gerçekleşir. Gelen paketler deşifrelenirken giden paketler şifrelenir. Güvenlik seviyesi şifreleme/deşifreleme işlemi için seçilecek algoritmanın hangisi olacağını belirler.

Bu tez çalışmasında iki adet güvenlik seçeneği belirlenmiştir. Güvenlik seçenekleri Çizelge 5.1'te belirtilmiştir.

Çizelge 5.1 PILSC'te kullanılan güvenlik seçenekleri

Güvenlik Seçenekleri	**Kullanılan Algoritma**
1	RC5
2	AES

5.7 Sistemin Durdurulması

Sistemin durdurulması sırasında kural listeleri temizlenerek hafızada yer açılır. Yeni IP paket işleyicisi görevden alınırken orijinal IP paket işleyicisi devreye alınır. Böylece güvenlik modülü devre dışı bırakılır.

6. KULLANILAN FONKSİYONLAR

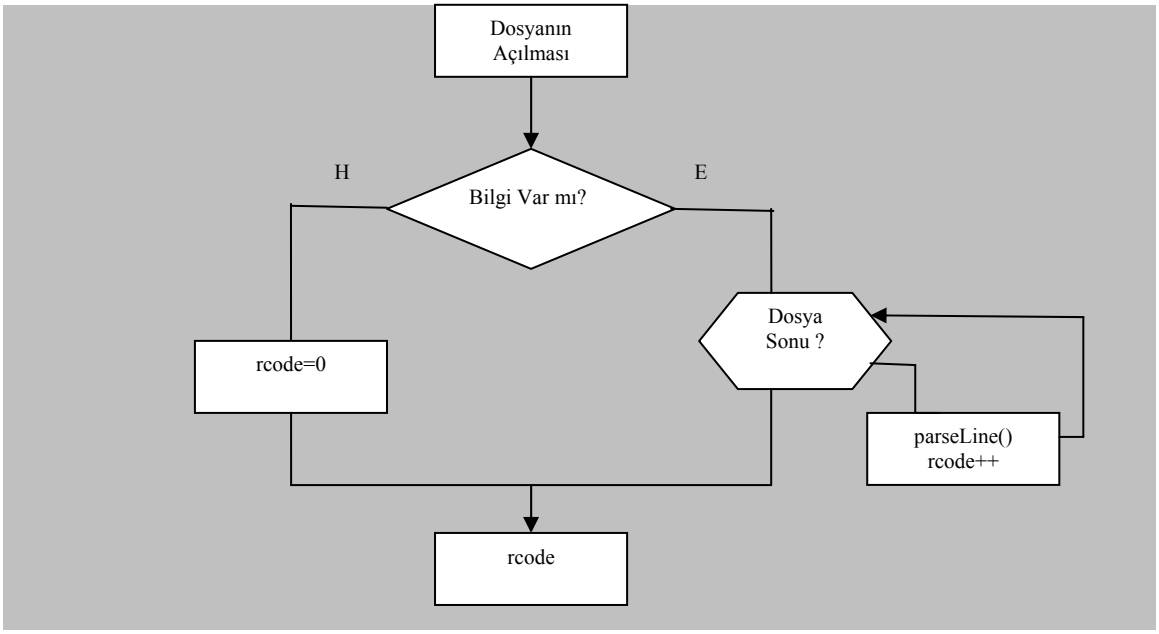
Sistemde gerçeklenen fonksiyonlar, kullanıldıkları aşamalara göre aşağıda belirtilmiştir.

6.1 Sistemin Başlatılması İçin Gerekli Fonksiyonlar

Sistemin başlatılması aşamasında kullanılan fonksiyonlar aşağıda listelenmiştir.

6.1.1 parseRuleFile Fonksiyonu

Ana kural dosyasının okunup parçalara ayrılması için kullanılan fonksiyondur. Yüklenebilir modüle ait *init_module* fonksiyonu tarafından çağrılır. Dosyanın açılıp her satırdaki bilginin okunması ve kontrolünden sorumludur. Fonksiyonun akışı Şekil 6.1'de verilmiştir.



Şekil 6.1 parseRuleFile fonksiyonuna ait akış diyagramı

Linux çekirdeğinde, okunacak dosyayı açmak için kullanılan C kütüphanesi kullanılamaz. Dosya işlemleri için Linux çekirdeğinde kullanılan dosya veri yapısı ve *filp_open* fonksiyonu kullanılmıştır. Dosyadaki her satır tek tek okunarak *parseLine* fonksiyonuna gönderilir. *parseLine* fonksiyonu parçalara ayırma işlemini gerçekleştirir. parseRuleFile fonksiyonu içinde ayrıca kural dizisini oluşturan bir fonksiyon ve bu diziyi sıralayan başka bir fonksiyon çağrılmaktadır. Fonksiyona ait prototip aşağıdadır.

*int parseRuleFile(RULEDB *, int *);*

6.1.2 parseLine Fonksiyonu

parseRuleFile fonksiyonu tarafından çağrılan bu fonksiyonun görevi satırda yer alan bilgileri ayırarak *config* adı verilen linkli liste yapısındaki veri yapısına ilgili bilgilerin yerleştirilmesini yönetmektir. Kural dosyasında yedi adet bilgi saklanır. Çizelge 6.1'de ayrıntılı bilgi verilmiştir.

Çizelge 6.1 Kural dosyasında saklanan bilgiler

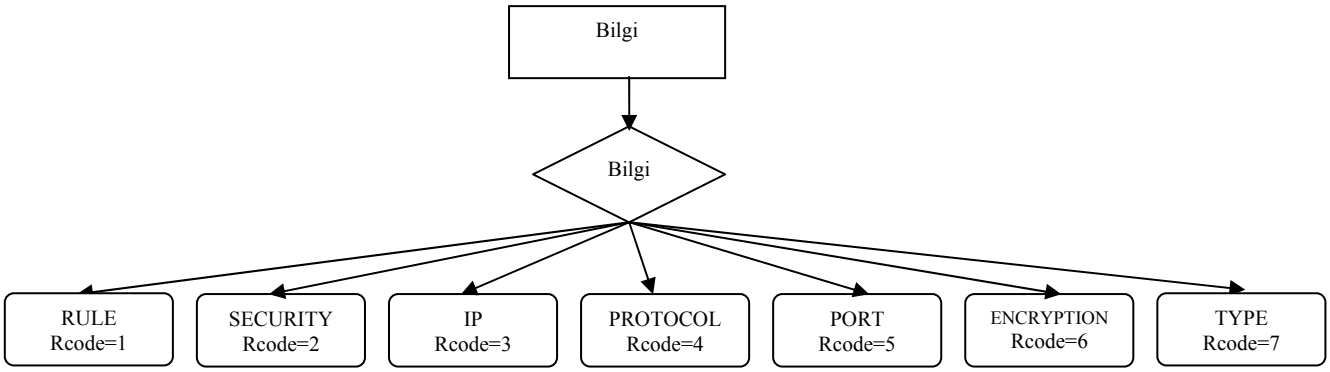
İsim	Açıklama
Rule	Kural numarasını gösterir.
Security	Kuralın o an gerekli olup olmadığını gösterir.
Protocol	Şifreli konuşulacak IP protokolünü gösterir.
Port	Şifreli konuşulacak port aralığını gösterir.
Encryption	Güvenlik seçeneğini gösterir.
Type	Uygulamanın hangi modda (istemci/sunucu) çalıştığını gösterir

compare fonksiyonu çağrılarak hangi bilginin yorumlanan satırda yer aldığı, *parseAttribute* ve *parseFeature* fonksiyonları çağrılarak bilgilerin hangi değere sahip olduğu belirlenir. Elde edilen bilgiler *config* veri yapısındaki uygun alanlara kaydedilir. İşlemler sırasında dosyada hatalı veri olduğu tespit edilirse "-1" değeri döndürülür. Fonksiyonun prototipi aşağıda verilmiştir.

*int parseLine (char *, CONFIG *, CONFIG **);*

6.1.3 compare Fonksiyonu

parseLine fonksiyonu tarafından çağrılır. Görevi satırda yorumlanan satırda hangi bilgi tipinin yer aldığını belirlemektir. Fonksiyona ait prototip aşağıda yer almaktadır. Fonksiyonun akış yapısı Şekil 6.2'de gösterilmiştir.



Şekil 6.2 compare fonksiyonuna ait akış diyagramı

Uygun eşleşme yapılamazsa hata değeri olarak "-1" döndürür. Fonksiyona ait prototip aşağıdadır.

*int compare (char *);*

6.1.4 parseAttribute Fonksiyonu

parseLine tarafından çağrılan bu fonksiyon *security, protocol, encryption* ve *type* alanlarına ait bilgileri değerlendirir. Security alanı *true* veya *false*, protocol alanı *tcp,udp* veya *icmp*, encryption alanı *1,2,3...n,* type alanı *server* ve *client* değerlerini alabilir. Değerler hem küçük hem de büyük harf olarak kontrol edilir. Hatalı değer var ise "-1" değeri döndürülür. *parseAttribute* fonksiyonu tarafından değerlendirilen alanlar ve alabileceği değerler Şekil 6.3'de gösterilmiştir. Fonksiyona ait prototip aşağıdadır.

security	<true|false>
protocol	<tcp|udp|icmp>
encryption	<1..n>
type	<server|client>

Şekil 6.3 Kural dosyasında yer alan bazı alanlar ve değerleri

*int parseAttribute(char *,int);*

6.1.5 parseFeature Fonksiyonu

parseLine tarafından çağrılır. IP adreslerinin ve port numaralarının yorumlandığı fonksiyondur. IP adreslerinin ve port numaralarının başlangıç ve bitiş değerleri, ilgili veri yapısının alanlarına kaydedilir. Eğer tek bir değer mevcut ise başlangıç ve bitiş değerleri aynı kaydedilir. *parseFeature* fonksiyonu tarafından değerlendirilen alanlar ve alabileceği değerler Şekil 6.4'de gösterilmiştir. Fonksiyona ait prototip aşağıdadır.

port	<0..0xffff>
ipaddress	<0.0.0.0. .. ff.ff.ff.ff>

Şekil 6.4 Kural dosyasında yer alan diğer alanlar ve değerleri

*void parseFeature(char *,int, CONFIG *);*

6.1.6 createRuledb Fonksiyonu

parseRuleFile tarafından çağrılan bu fonksiyonun görevi ana fonksiyonda kullanılan *ruledb* veri yapısını oluşturmaktır. Linkli veri yapısı *config* kullanılarak kaç adet kural olduğu belirlenir. *security* değeri false olan kayıtlar değerlendirmeye alınmaz ve yeni listeye aktarılmaz. Fonksiyona ait prototip aşağıdadır.

*RULEDB *createRuledb(CONFIG *, CONFIG *, int **) ;*

6.1.7 sortTrash Fonksiyonu

parseRuleFile tarafından çağrılır. Ana görevi *ruledb* veri yapısını sıralı hale getirip eş olan kayıtları silmektir. Böylece kullanıcı tarafından yapılan hata sistem tarafından giderilmektedir. Fonksiyona ait prototip aşağıdadır.

*void sortTrash(RULEDB **, int **);*

6.1.8 sortDecide Fonksiyonu

sortTrash tarafından çağrılan bu fonksiyonun görevi aktarılan parametreye bağlı olarak sırası ile IP adresi, protokol ve port numarasına göre sıralama işlemini gerçekleştirmektir. Fonksiyona ait prototip aşağıdadır.

*int sortDecide(RULEDB *,RULEDB,int,int);*

6.2 Ayrıştırma İşlemine Ait Fonksiyonlar

Ayrıştırma aşamasında kullanılan fonksiyonlar ve fonksiyonlara ait bilgi aşağıda verilmiştir.

6.2.1 packetTypeMenu Fonksiyonu

newiprcv fonksiyonu tarafından çağrılan bu fonksiyon işlemci kuyruğunda yer alan paketlerin tiplerine göre ayrıştırma işleminin ilk adımını gerçekleştirir. *skb_buff* veri yapısının type alanı kontrol edilir. PACKET_OUTGOING değerine sahip olanlar giden, PACKET_HOST değerine sahip olanlar gelen paket olarak işlem görür. Eşleştirme aşamasında kullanılacak hedef ve kaynak IP adres bilgisi, hedef ve kaynak port bilgisi, protokol tipi, verinin uzunluğu, IP ve TCP başlık bilgilerinin başlangıç adresleri ve uzunlukları gerekli yerel değişkenlere aktarılır. Fonksiyona ait prototip aşağıdadır.

*void packetTypeMenu(struct sk_buff *);*

6.2.2 addressVerification Fonksiyonu

packetTypeMenu tarafından çağrılır. Eşleşme kontrolü yapılacak paketin IP adresi bilgisi *ruledb* listesindeki kayıtlarla karşılaştırılır. Hızlı arama yapabilmek için IP eşleşmesinde "binary search" yöntemi kullanılır. Eşleşme kontrolünün dışında kayıt listesinin sonuna veya başına ulaşılıp ulaşılmadığı da kontrol edilir. Fonksiyona ait prototip aşağıdadır.

int addressVerification(unsigned long, unsigned long,int, unsigned int,unsigned int,char);

6.2.3 protocolVerification Fonksiyonu

addressVerification fonksiyonu tarafından çağrılır. IP adresi eşleşmesi doğru sonuçlanırsa çağrılan bu fonksiyon paketin protokol bilgisini *ruledb* listesindeki kayıtlarla karşılaştırır. Hızlı arama yapabilmek için protokol eşleşmesinde "binary search" yöntemi kullanılır. Eşleşme kontrolünün dışında kayıt listesinin sonuna veya başına ulaşılıp ulaşılmadığı da kontrol edilir. Fonksiyona ait prototip aşağıdadır.

int protocolVerification(unsigned long, unsigned long,int, unsigned int,unsigned int,char);

6.2.4 portVerification Fonksiyonu

protocolVerification fonksiyonu tarafından çağrılır. Protokol bilgisi eşleşmesi doğru sonuçlanırsa çağrılan bu fonksiyon paketin hedef ya da kaynak port bilgisini *ruledb* listesindeki kayıtlarla karşılaştırır. Hızlı arama yapabilmek için port eşleşmesinde "binary search" yöntemi kullanılır. Eşleşme kontrolünün dışında kayıt listesinin sonuna veya başına ulaşılıp ulaşılmadığı da kontrol edilir. Fonksiyona ait prototip aşağıdadır.

int portVerification(unsigned long, unsigned long,int, unsigned int,unsigned int,char);

6.2.5 ipChoice Fonksiyonu

addressVerification fonksiyonu tarafından çağrılan bu fonksiyon paketin tipine göre hedef ve kaynak adresten hangisinin eşleşme için kullanacağını belirler. Fonksiyona ait prototip aşağıdadır.

unsigned long ipChoice(unsigned long,unsigned long,char);

6.2.6 portChoice Fonksiyonu

portVerification fonksiyonu tarafından çağrılan bu fonksiyon paketin tipine ve uygulamanın çalışma moduna (istemci/sunucu) göre hedef ve kaynak porttan hangisinin eşleşme için kullanacağını belirler. Fonksiyona ait prototip aşağıdadır.

unsigned int portChoice(unsigned int,unsigned int,int, char);

6.2.7 ftpFix Fonksiyonu

packetTypeMenu fonksiyonu tarafından çağrılır. Fonksiyonun görevi ftp bağlantısı sırasında veri transferi için açılan yeni portları *ftpdb* adındaki veri yapısına eklemek ve ftp paketleri için bu listede eşleşme kontrolü yapmaktır. Bağlantı sonlandığında listedeki açılan güvenli portlar listeden çıkarılır. Fonksiyona ait prototip aşağıdadır.

*void ftpFix(struct sk_buff *, unsigned long,unsigned long,char);*

6.3 Kriptografik İşlemlere Ait Fonksiyonlar

Kriptografik işlemlere ait fonksiyonların açıklamaları aşağıdadır.

6.3.1 cryptProcess Fonksiyonu

packetTypeMenu tarafından çağrılan bu fonksiyon aktarılan paketin içerisinde yer alan bilginin kriptografik işlemlerini yönetir. Giden paketler ilgili şifreleme algoritmasına yönlendirilirken gelen paketler ilgili deşifreleme algoritmasına aktarılır. Hangi algoritmanın seçileceğine *ruledb* listesindeki eşleşmeye uygun kayıtta yer alan encryption tipine göre karar verilir. Ayrıca fonksiyonun başında gerekli anahtar değerleri ilklendirilir (initialize).

6.3.2 Kriptografik Algoritmalar

PILSC sistemi güvenli veri iletişimi için kriptografik algoritmalar kullanmaktadır. Sistemin en önemli avantajı birçok şifreleme algoritmasına verdiği destektir. Bu tez çalışmasında, yapılan testler sırasında RC5 ve AES algoritmaları kullanılmıştır (Schneier, 1996).

6.3.2.1 RC5 Algoritması

RC5 şifreleme algoritması simetrik şifreleme yöntemidir. RC5 algoritmasında kullanılan blok büyüklüğü (32, 64, 128 bit), anahtar büyüklüğü (0-2040 bit) ve döngü sayısı (0-255) değişkendir. Genel olarak kullanılan blok büyüklüğü 64 bit, anahtar değeri 64 bit, döngü sayısı 12'dir.

RC5 algoritması modüler ve XOR işlemleri kullanarak gerçeklenir. Algoritmanın genel yapısı Feistel ağ yapısından oluşur [10].

6.3.2.2 AES Algoritması

AES şifreleme algoritması Amerikan hükümeti tarafından standart olarak kullanılan simetrik şifreleme yöntemidir. Blok büyüklüğü olarak 128 bit kullanan algoritma, anahtar değeri olarak 128, 192 ya da 256 bit kullanmaktadır. AES şifreleme algoritması 4*4 byte'lık dizi üzerinde yapılan işlemler ile gerçeklenir [10].

6.4 Dosyaların Ayrıştırılmasında Kullanılan Yardımcı String Fonksiyonları

Konfigürasyon dosyalarının parçalara ayrılmasında ihtiyaç duyulan string fonksiyonları çekirdek seviyesinde kullanılmadığı için, bu fonksiyonlar tekrar gerçeklenmiştir. Yeniden gerçeklenen fonksiyonlara ait prototipler, dönüş değerleri ve kullanımları, standart C kütüphanelerinde aynı amaç için kullanılan fonksiyonlar ile aynıdır. Bu fonksiyonların isimleri ve görevleri Çizelge 6.2'de belirtilmiştir.

Çizelge 6.2 Kullanılan yardımcı fonksiyonlar ve görevleri

İSİM	AÇIKLAMA
int isSpace(char)	Karakterin boşluk karakteri olup olmadığını döndürür
int isDigit(char)	Karakterin nümerik olup olmadığını döndürür.
int isAlpha(char)	Karakterin alfabetik olup olmadığını döndürür.
int strLen(char *)	Parametre olarak verilen stringin karakter cinsinden uzunluğunu verir
int strCmp(char*,char *)	Verilen iki stringi karşılaştırır.

6.5 Kullanılan Modifiye Edilmiş Çekirdek Fonksiyonları

Sistemin kullandığı, çekirdekte tanımlı olan modifiye edilmiş fonksiyonlara ait bilgi aşağıda belirtilmiştir.

6.5.1 new_ip_rcv Fonksiyonu

Sistemde tanımlanan orijinal IP işleyici fonksiyonun kod parçalarından faydalanarak yazılan yeni IP işleyici fonksiyonudur. Aldığı parametreler pakete ait soket ve protokol bilgisini içeren *sk_buff* yapısı, cihazın geldiği cihaza ait işaretçi ve paketin tipine ait yapıdır. Gelen ve giden paketlere servis vermek için kullanılmıştır. Fonksiyona ait prototip aşağıdadır.

*int new_ip_rcv(struct sk_buff *, struct net_device *, struct packet_type *);*

6.6 Yüklenebilir Çekirdek Modülüne Ait Genel Fonksiyonlar

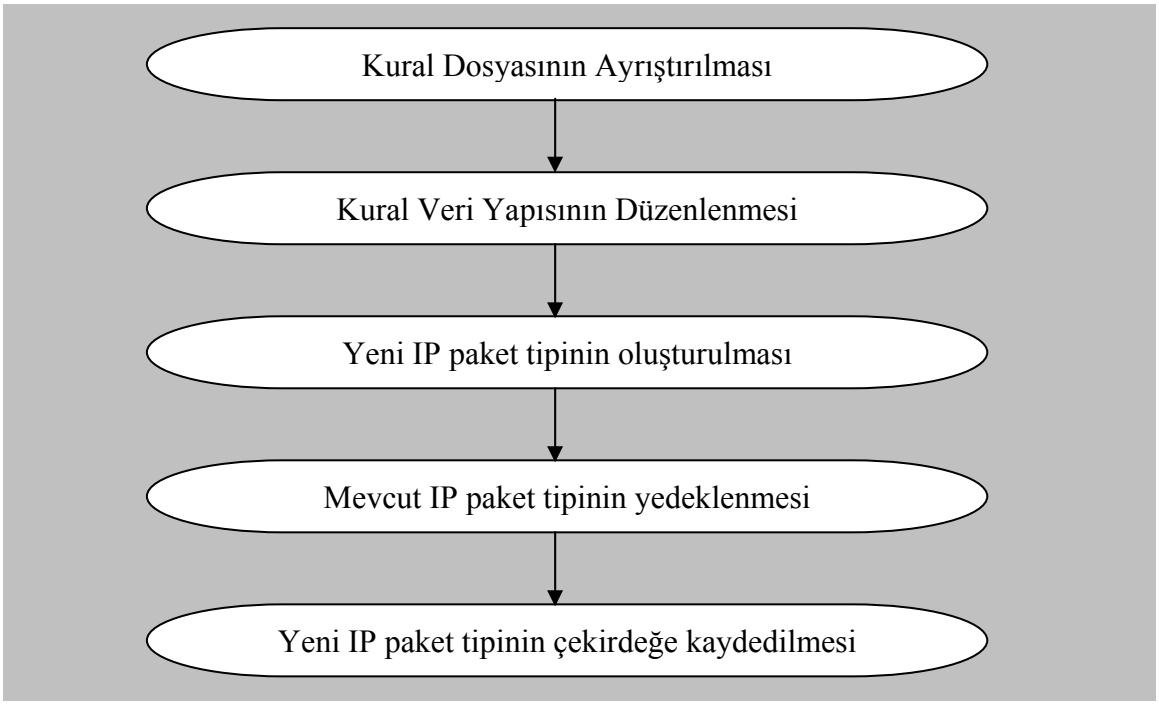
Sistem, çekirdeğe dinamik olarak yüklenme imkanı veren yüklenebilir çekirdek modülü olarak gerçeklendiğinden, bu modüllerde bulunması gereken iki adet fonksiyonun gerçeklenme zorunluluğu vardır. Bu iki fonksiyona ait bilgiler aşağıdadır.

6.6.1 init_module Fonksiyonu

init_module fonksiyonu, sistemin çalışması için kullanılan modülün çekirdeğe yüklenmesi anında ilk çalışan fonksiyondur. Bu fonksiyonda modül devreye girmeden önce yapılması gereken işlemler gerçekleştirilir. Gerçeklenen sistemde *init_module* fonksiyonu sistemin başlaması için gerekli olan fonksiyonların çağrılması için kullanılmıştır. Fonksiyonun akışı Şekil 6.5'te gösterilmiştir. Fonksiyonun prototipi aşağıdaki gibidir.

int init_module(void) ;

init_module fonksiyonu LKM'nin çalışması gereğince, işlemlerin başarılı olması durumunda, modülün yüklenebilmesi için "0" değerini döndürmelidir. *init_module* sistemin başlaması için gerekli olan bütün fonksiyonları çağırır. Tüm fonksiyonları başarılı olması durumunda modülün yüklenmesi işlemini onaylayacak olan 0 değerini döndürür. Çağrılan fonksiyonlardan birinin hatalı değer döndürmesi, *init_module* fonksiyonunun "0"dan farklı bir değer döndürmesini sağlar ve modülün yüklenmesini engeller.



Şekil 6.5 init_module fonksiyonunun akış diyagramı

6.6.2 cleanup_module Fonksiyonu

Sistemin devre dışı bırakılması sırasında çağrılan fonksiyondur. Kullanım amacı modülde oluşturulan veri yapılarının temizlenmesidir.

Gerçeklenen sistemde cleanup_module fonksiyonu, orijinal IP paket yapısını devreye alır ve sistem tarafından tanımlanmış olan yeni IP paketi tipi devreden çıkarılır. Böylece sistemin orijinal IP işlevselliği korunmuş olur. Fonksiyona ait prototip aşağıdadır.

void cleanup_module(void);

7. KULLANILAN VERİ YAPILARI

Sistemde kullanılan veri yapıları, kullanım amaçları, içerdikleri alanlar ve alanların amaçları aşağıda belirtilmiştir.

7.1 config Veri Yapısı

Sistemin başlaması sırasında kural dosyasından alınan kayıtların aktarıldığı linkli liste yapısındaki veri yapısıdır. Ana kural veri yapısını oluşturmak için kullanılır. Boyutu dinamik olarak belirlenir. Veri yapısına ait alanlar ve alanların anlamları Şekil 7.1'de gösterilmiştir.

```
typedef struct
{
  char security          ;    // Kuralın geçerli olup olmadığını gösterir.
  char protoType         ;    // Geçerli olacak protokolü gösterir.
  u32 fromIpaddr         ;    // Geçerli olacak IP aralığının başlangıç adresini saklar.
  u32 toIpaddr           ;    // Geçerli olacak IP aralığının bitiş adresini saklar.
  u16 fromPortNumber     ;    // Geçerli olacak port aralığının başlangıç no'sunu saklar.
  u16 toPortNumber       ;    // Geçerli olacak port aralığının bitiş no'sunu saklar.
  char encrypType        ;    // Güvenlik seviyesini belirtir.
  char type              ;    // Uygulamanın modunu gösterir. (istemci/sunucu)
  struct CONFIG *next    ;    //  Linkli listedeki sonraki elemanın adresini tutar.
} CONFIG;
```

Şekil 7.1 CONFIG veri yapısı

7.2 ruledb Veri Yapısı

Sistemin başlaması sırasında oluşturulan ayrıştırma işlemi için kullanılan veri yapısıdır. *config* adındaki linkli liste kullanılarak oluşturulur. *config* listesinde yer alan aynı kayıtlar *ruledb* listesine aktarılmaz. Ayrıca security alanı false olan kayıtlar da elenir. *ruledb* yapısı Şekil 7.2'de gösterilmiştir.

```
typedef struct
{
  char security          ;     // Kuralın geçerli olup olmadığını gösterir.
  char protoType         ;     // Geçerli olacak protokolü gösterir.
  u32  ipaddr            ;     // Eşleştirilecek IP adresini saklar.
  u16  portNumber        ;     // Eşleştirilecek port numarasını gösterir.
  char encrypType        ;     // Güvenlik seviyesini belirtir.
  char type              ;     // Uygulamanın modunu gösterir. (istemci/sunucu)
} RULEDB;
```

Şekil 7.2 RULEDB veri yapısı

7.3 ftpdb Veri Yapısı

Sistemin çalışması sırasında FTP protokolü bağlantılarında veri alışverişi için açılan yeni portların güvenli veri iletişimi yapabilmesini sağlamak için kullanılan veri yapısıdır. FTP bağlantısı sırasında FTP komutlarına göre listeye yeni portlar eklenir veya listeden daha önce eklenen portlar silinir. *ruledb* listesinde eşleşme sağlanamazsa *ftpdb* listesi kontrol edilir. *ftpdb* Şekil 7.3'te gösterilmiştir.

```
typedef struct
{
  u32 ipaddr             ;     // Eklenen ftp  bağlantı numarasını saklar.
  u16 portNumber         ;     // Eklenen ftp veri port numarasını saklar.
  char encrypType        ;     // Eklenen ftp veri portunun güvenlik seviyesini belirler.
}FTPDB;
```

Şekil 7.3 FTPDB veri yapısı

8. KULLANILAN DOSYA YAPILARI

Sistemde kullanılan yapılandırma dosyasına ve yapısına ait bilgi aşağıda verilmiştir. Dosyaların kullanımında yapıların değiştirilmesi söz konusu değildir.

8.1 Kural Dosyası

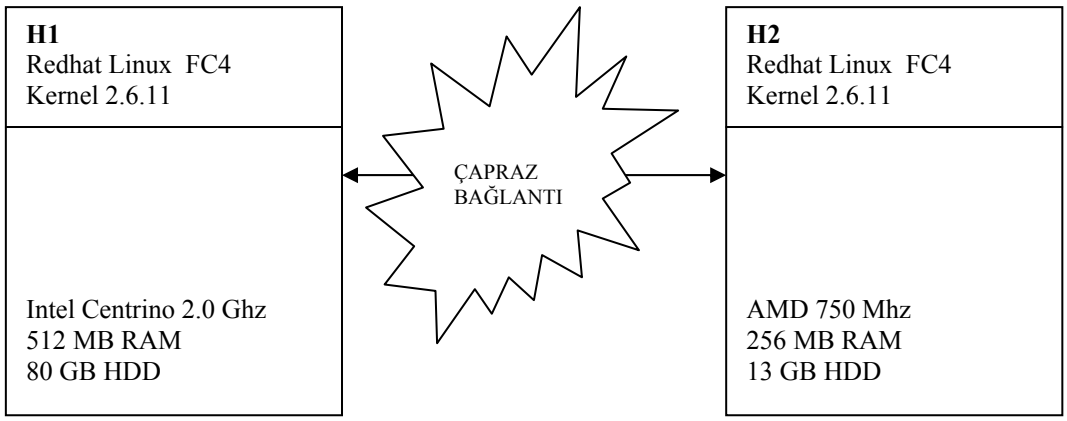
Sistemin güvenli veri iletişimi için kullanacağı bu dosyanın formatı alt düzey kullanıcının da anlayacağı dilde tasarlanmıştır. Dosya yapısı için seçilen etiketlerden oluşan bir örnek Şekil 8.1 belirtilmiştir. Kural etiketi hariç diğer etiketler değişik sırada yazılabilir komut satırı için # işareti kullanılarak hatırlatma bilgileri dosya içinde saklanabilir. Dosya yapısı için prototip aşağıdadır. Ayrıca örnek dosya yapısı Şekil 8.1'de gösterilmiştir.
[etiket] [=] [başlangıç değeri – bitiş değeri]

```
RULE = 1
SECURITY = TRUE
IP = 193.140.1.3 – 193.140.1.10
PORT = 20 – 22
ENCRYPTION =  1
TYPE =SERVER
# Elektrik bölümüne ait bilgisayarlar ile ftp ve ssh
# uygulamaları için şifreli konuşma kuralı
```

Şekil 8.1 Kural dosyası

9. SİSTEMİN TEST EDİLMESİ

Gerçeklenen sistemin test edilmesi için, bir yerel ağ oluşturulmuştur. İki bilgisayarın birbirine çapraz bağlanması ile oluşturulan bu ağ yapısında PILSC sisteminin, tasarlanan test senaryolarında davranışı analiz edilmiştir. Kullanılan bilgisayarların donanım ve yazılım yapılandırması Şekil 9.1'de verilmiştir. Her iki bilgisayar da istemci/sunucu modunda çalışabilmektedir.



Şekil 9.1 Güvenli iletişim sistemi için kullanılan test altyapısı

Şekil 9.1'de gösterilen H1 isimli konakçı (host) 192.168.1.3, H2 isimli konakçı (host) 192.168.1.4 ara yüzü üzerinden haberleşmektedir. Test aşamasında genel olarak, H1 isimli konakçı üzerinde çalışan uygulamalar sunucu modunda, H2 isimli konakçı üzerinde çalışan uygulamalar istemci modunda çalıştırılmıştır.

Test aşamasında, geliştirilen PILSC sisteminin kullanımına bağlı olarak çeşitli değerler elde edilmiştir. Bu değerler test senaryolarına bağlı olarak incelenmiştir.

9.1 Test Senaryoları

Test işlemleri için çeşitli senaryolar oluşturulmuştur:

1. Çeşitli boyutlardaki dosyaların FTP protokolü kullanarak transfer sürelerinin ölçülmesi
2. Kural dosyasındaki kural sayısına bağlı olarak yapılan dosya transferlerinin sürelerinin ölçülmesi
3. Farklı şifreleme algoritmaları kullanılarak dosya transfer sürelerinin ölçülmesi
4. Şifreleme algoritmalarının parametrelerinin kullanılarak transfer sürelerinin ölçülmesi
5. Paket sıklığı arttırılarak PILSC sisteminin bu stres testine verdiği cevabın ölçülmesi
6. Şifreleme algoritmalarının çekirdek ve kullanıcı seviyesinde çalışma sürelerinin ölçülmesi

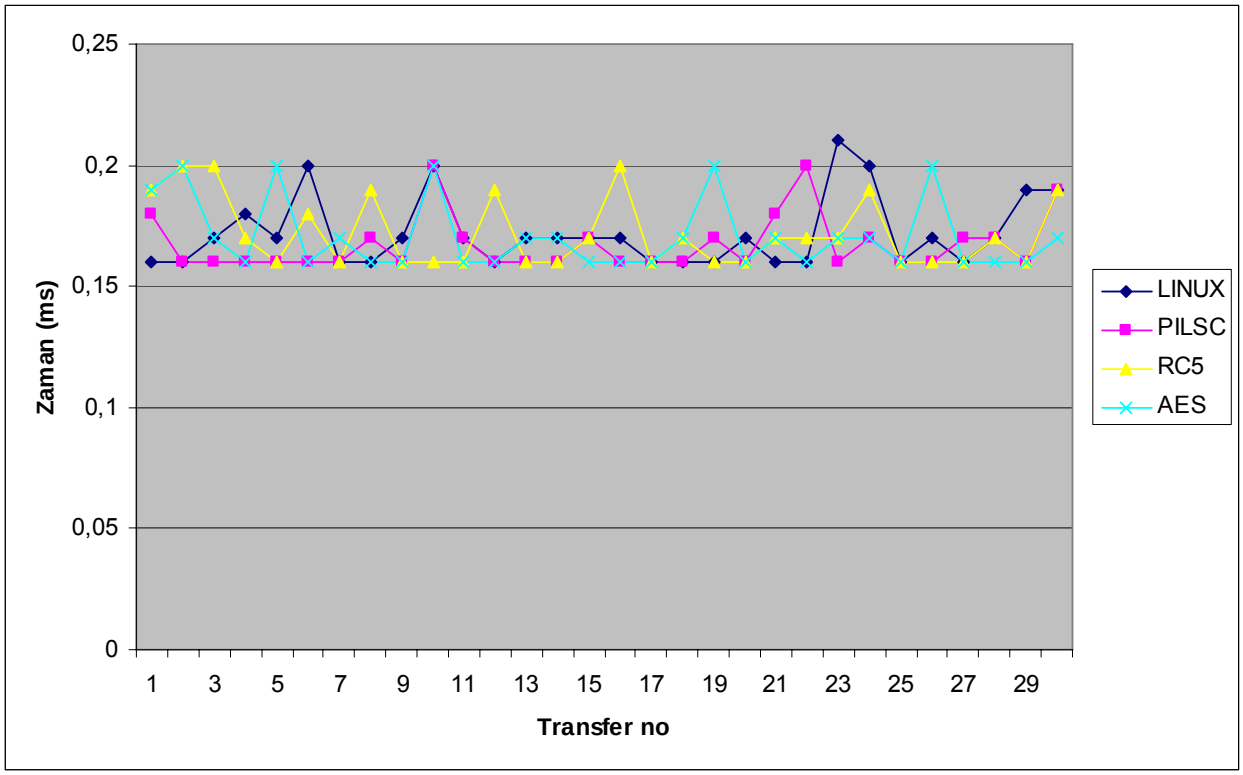
Yapılan tüm ölçümlerde aksi belirtilmediği sürece RC5 için anahtar değeri 64-bit, round sayısı 8, blok büyüklüğü 8, AES içinse anahtar değeri 128-bit ve blok büyüklüğü 16 olarak kullanılmıştır.

9.1.1 Dosya Boyutlarına Göre Alınan Değerler

İlk olarak çeşitli boyutlardaki dosyaların FTP protokolü üzerinden transfer süreleri ölçülmüştür. 1KB, 10KB, 1MB ve 10MB dosyaların FTP protokolü ile dört farklı durum için transfer süreleri ölçülmüştür. Transfer süreleri Şekil 9.2, Şekil 9.3, Şekil 9.4 ve Şekil 9.5'te gösterilmiştir. Grafiklerde "30" adet ölçümün transfer süreleri gösterilmiştir.

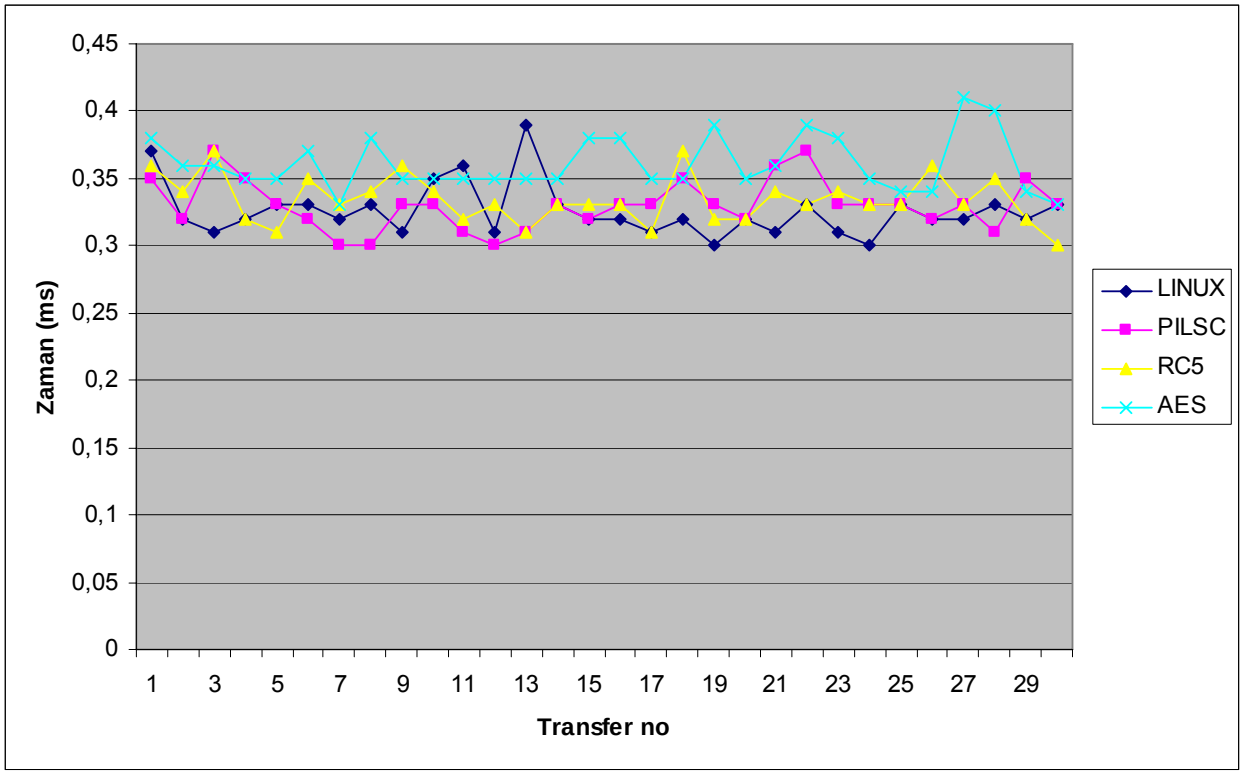
Ölçüm için kullanılan dört farklı durum şu şekildedir:

1. Linux paket işleyicisi
2. PILSC
3. PILSC (RC5 şifreleme algoritması)
4. PILSC (AES şifreleme algoritması)



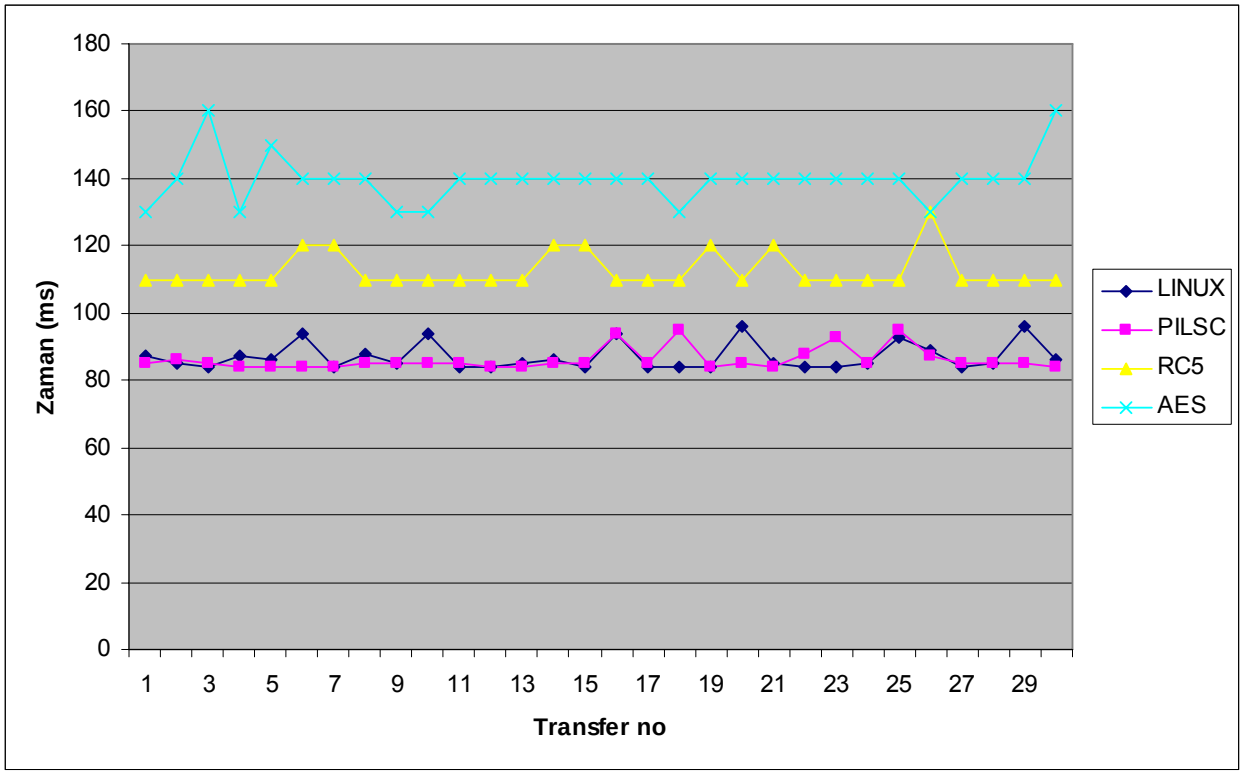
Şekil 9.2 1KB dosyanın transfer süreleri

Şekil 9.2'de 1KB boyutundaki dosyanın dört farklı durumda ölçülen transfer süreleri gösterilmiştir. Genel olarak dört durum içinde aynı süreler elde edilmiştir. Dosya boyutu küçük olduğu için kriptografik işlemlerin oluşturduğu zaman kaybı transfer süresini etkilememektedir.



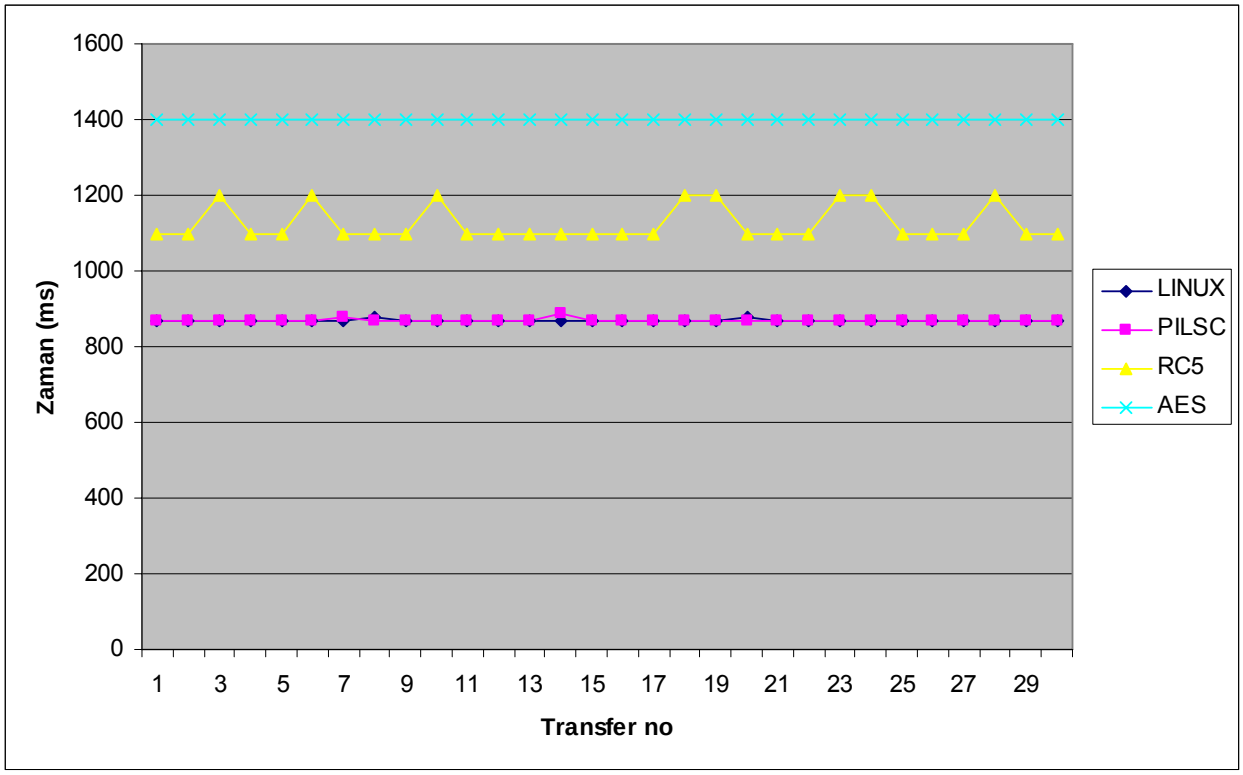
Şekil 9.3 10KB dosyanın transfer süreleri

Şekil 9.3'te 10KB dosyanın dört farklı durumda ölçülen transfer süreleri gösterilmiştir. Bu grafikten çıkarılan sonuç transfer sürelerinin belirli bir aralıkta salınım gösterdiği ve dosya boyutu büyük olmadığı için şifreleme algoritmaları kullanılmasına rağmen dosya transfer sürelerinin hemen hemen aynı olduğudur. Fakat grafik incelediğinde RC5 ve AES algoritmalarının ihmal edilebilecek kadar az da olsa transferi etkilediği gözlemlenmiştir.



Şekil 9.4 1MB dosyanın transfer süreleri

Şekil 9.4'te 1MB dosyanın dört farklı durumda ölçülen transfer süreleri gösterilmiştir. Grafikte PILSC sistemi ile Linux paket işleyicisi arasında paralellik ve çakışma görülmektedir. Bu da PILSC sisteminin herhangi bir gecikmeye neden olmadığını göstermektedir. Fakat şifreleme algoritmaları kullanıldığında durum değişmektedir. Dosyanın boyutu büyüdüğü için şifrelenen paket sayısı da artmaktadır. Bu da beraberinde dosya transferlerinin RC5 ve AES şifreleme algoritmaları için belirli oranda (%22-%37) gecikmeli olmasına neden olmaktadır.



Şekil 9.5 10 MB dosyanın transfer süreleri

Şekil 9.5'te 10MB dosyanın dört farklı durumda ölçülen transfer süreleri gösterilmiştir. Grafik incelendiğinde PILSC sisteminin dosya boyutu büyümesine rağmen Linux paket işleyicisi ile aynı hızda çalıştığı gözlemlenmiştir. RC5 ve AES algoritmaları kullanılarak yapılan transferlerde ise kriptografik işlemlerden kaynaklı gecikmeler gözlenmiştir. Ayrıca dosya boyutu büyüdüğünde dosyanın transfer süresindeki salınımın azaldığı ve belirli bir değere oturduğu gözlemlenmiştir.

Yapılan 100 adet ölçümün transfer sürelerinin ortalamaları alınarak Çizelge 9.1 ve Çizelge 9.2 oluşturulmuştur. İki çizelge yorumlandığında PILSC sisteminin paket değerlendirme sırasında yaptığı işlemlerin ihmal edilebilecek kadar az yük getirdiği gözlemlenmiştir. Ayrıca transfer edilen dosyanın boyutunun küçük olduğu durumlarda kriptografik işlemlerin oluşturduğu ek sürenin %1-%5 civarında gecikmeye neden olduğu tespit edilmiştir. 1MB ve üzerindeki dosya boyutlarında 64-bit şifreleme yapan RC5'in 128-bit şifreleme yapan AES algoritmasına göre daha hızlı çalıştığı gözlenmiştir.

Çizelge 9.1 FTP protokolü ile yapılan dosya transferlerinin ortalama süreleri (ms)

Dosya Boyutu / Ortalama süreler	LINUX	PILSC	RC5	AES
1KB	0,171	0,172	0,172	0,173
10KB	0,326	0,330	0,334	0,360
1MB	87	87,1	112,6	139,6
10MB	870,6	871	1126	1400

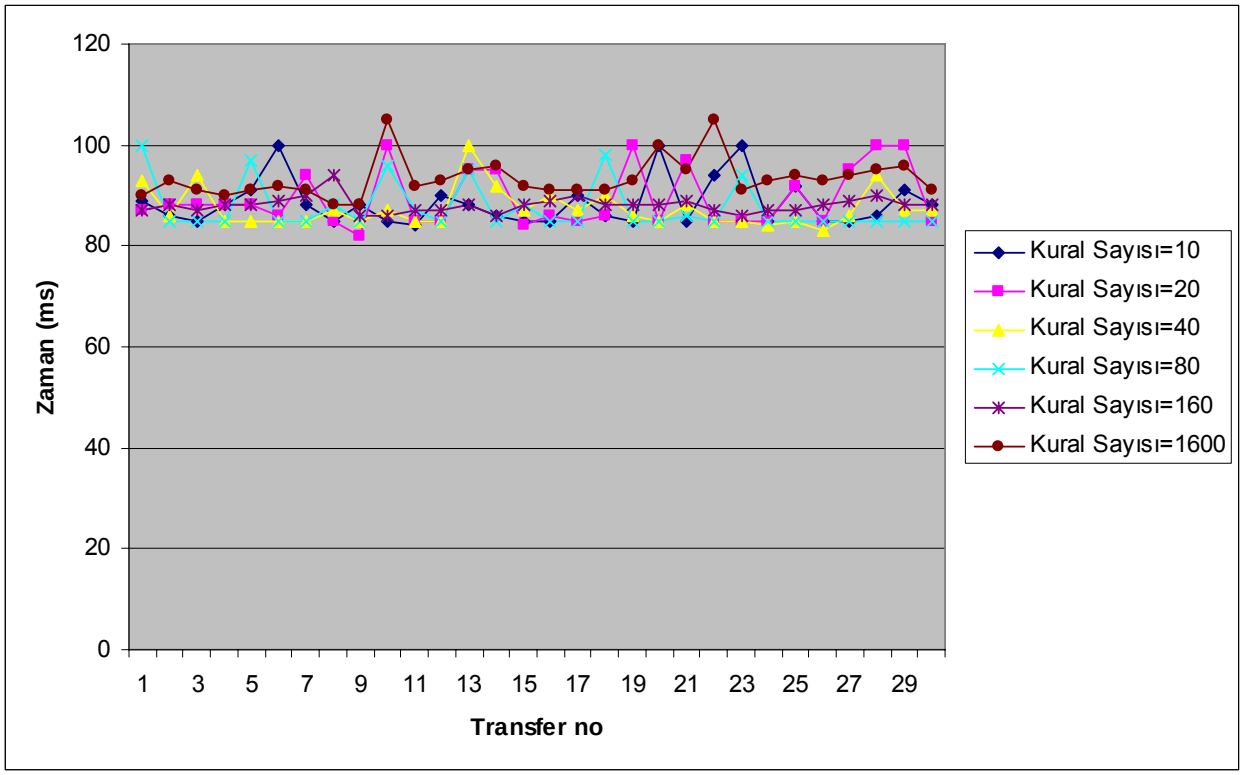
Çizelge 9.2 Dosya transfer sürelerinin Linux paket işleyicisine göre gecikme oranları

Dosya Boyutu / Gecikme Oranları	PILSC	RC5	AES
1KB	%0,005	%0,05	%1,2
10KB	%1,3	%2,4	%5,4
1MB	%0,002	%19,3	%37,5
10MB	%0,0001	%19,5	%37,7

9.1.2 Kural Sayısına Göre Alınan Değerler

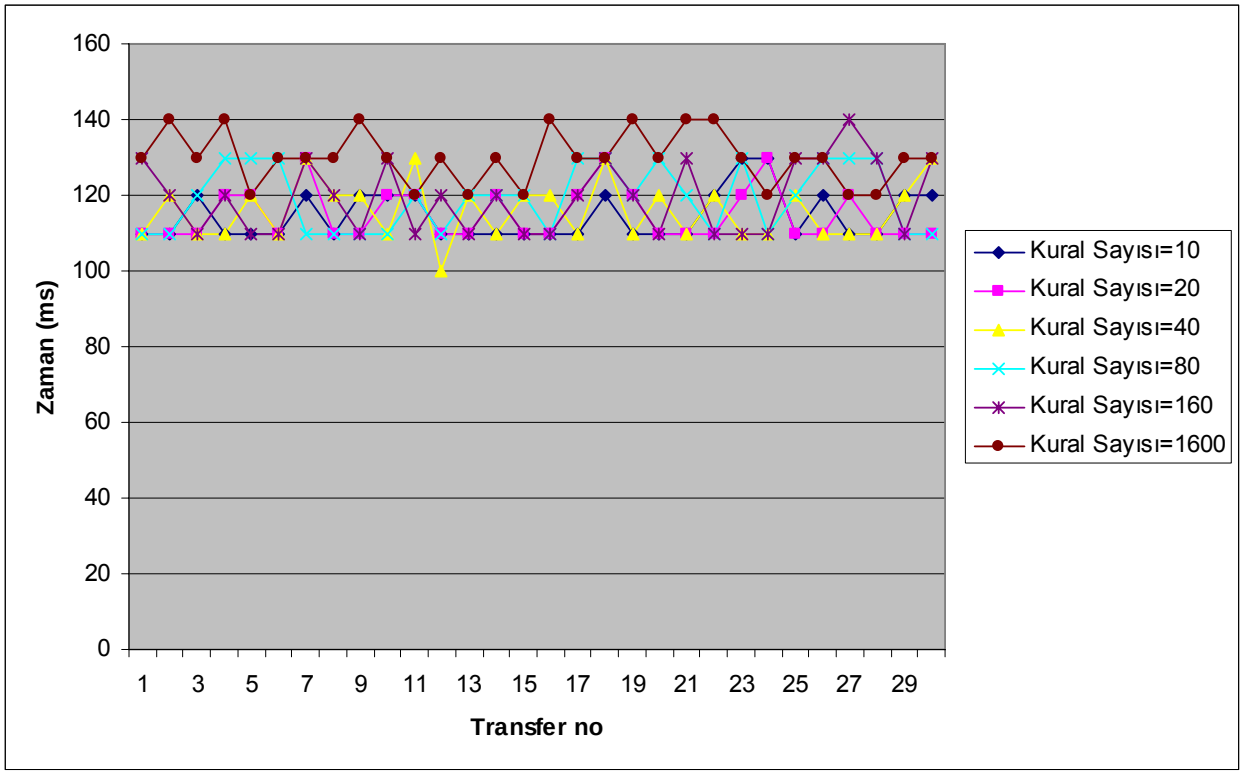
Kural dosyasında yer alan kuralların sayısı değiştirilerek üç farklı durum için dosya transfer süreleri ölçülmüştür. Kural sayıları istemci ve sunucu için eşit olarak seçilmiştir. Test aşamasında 1MB büyüklüğünde dosya FTP protokolü kullanılarak 100 kez transfer edilmiştir. Şekil 9.6, Şekil 9.7, Şekil 9.8 ve Şekil 9.9'da elde edilen test sonuçları grafik olarak gösterilmiştir. Ölçüm için kullanılan üç farklı durum şu şekildedir:

1. PILSC
2. PILSC (RC5 şifreleme algoritması)
3. PILSC (AES şifreleme algoritması)



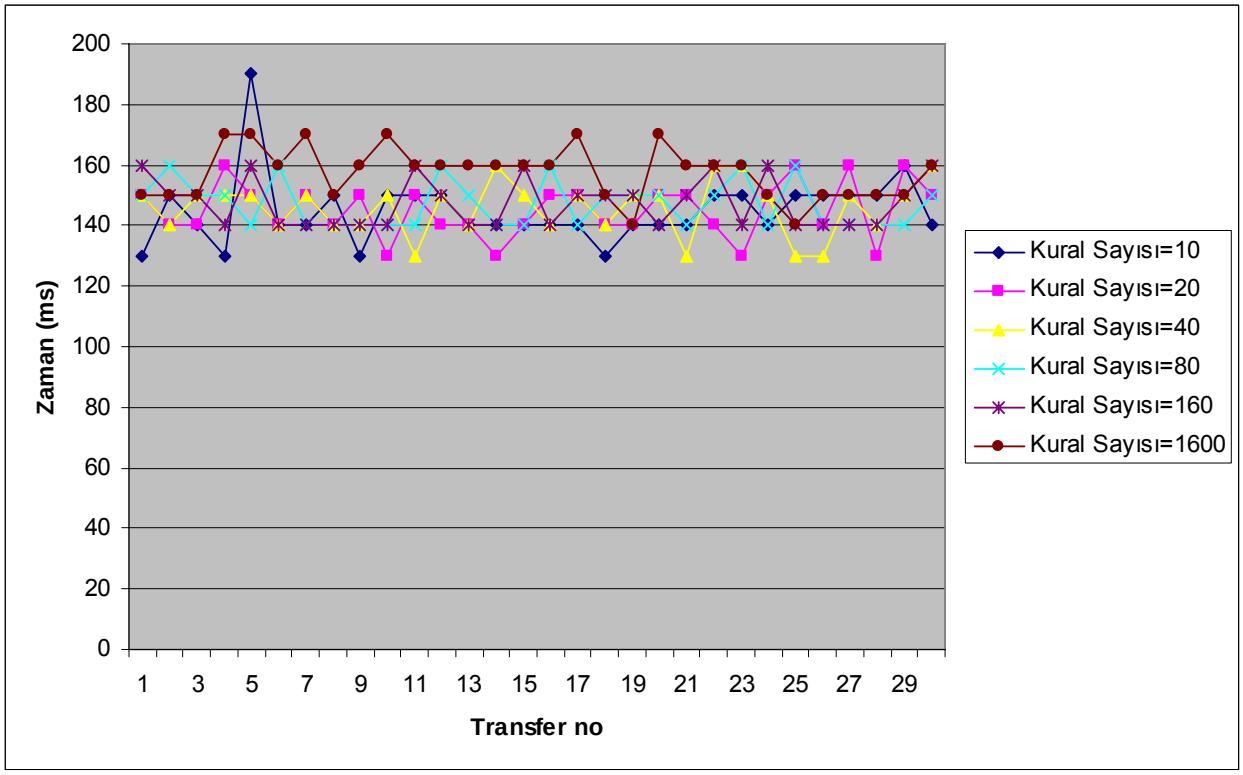
Şekil 9.6 PILSC Sisteminin kural dosyasındaki kural sayısına göre transfer süreleri (ms)

Şekil 9.6'da PILSC sistemi çalışırken transfer edilen dosyanın kural sayısına bağlı olarak transfer süreleri gösterilmektedir. Değerlerin belirli bir aralıkta salınım gösterdiği gözlemlenmiştir. Kural sayısı "80"e çıktığında tepe noktalarının biraz daha arttığı fakat dosya transfer sürelerinde büyük bir değişim olmadığı tespit edilmiştir. Dosya transfer sürelerinin kural sayısı "1600" e çıktığında belirgin düzeyde (%10) arttığı gözlemlenmiştir.



Şekil 9.7 PILSC sisteminin kural sayısına göre RC5 algoritmasını kullanarak elde edilen sonuçlar

RC5 algoritması kullanılarak yapılan dosya transfer sürelerinin kural sayısına bağlı ölçümleri Şekil 9.7'te gösterilmiştir. Grafikler incelediğinde kural sayısı arttığında genel olarak transfer süresinin de arttığı belirlenmiştir. Özellikle kural sayısının "80" ve "160" olduğu durumlarda daha çok tepe noktası gözlenmiştir. Kural sayısı "1600" olduğunda ise ölçülen süreler gözle görülecek oranda artış göstermiştir.



Şekil 9.8 PILSC sisteminin kural sayısına göre AES algoritmasını kullanarak elde edilen sonuçlar

Şekil 9.8'de AES algoritması kullanılarak farklı kural sayıları için dosya transfer sürelerinin değerleri grafik olarak gösterilmiştir. Kural sayıları günlük hayatta kullanılan seviyelerde olduğu sürece ihmal edilebilir gecikmeler meydana gelmiştir. Gecikme sürelerinde meydana gelen belirgin farklılık kural sayısının "1600" olduğu durumda tespit edilmiştir.

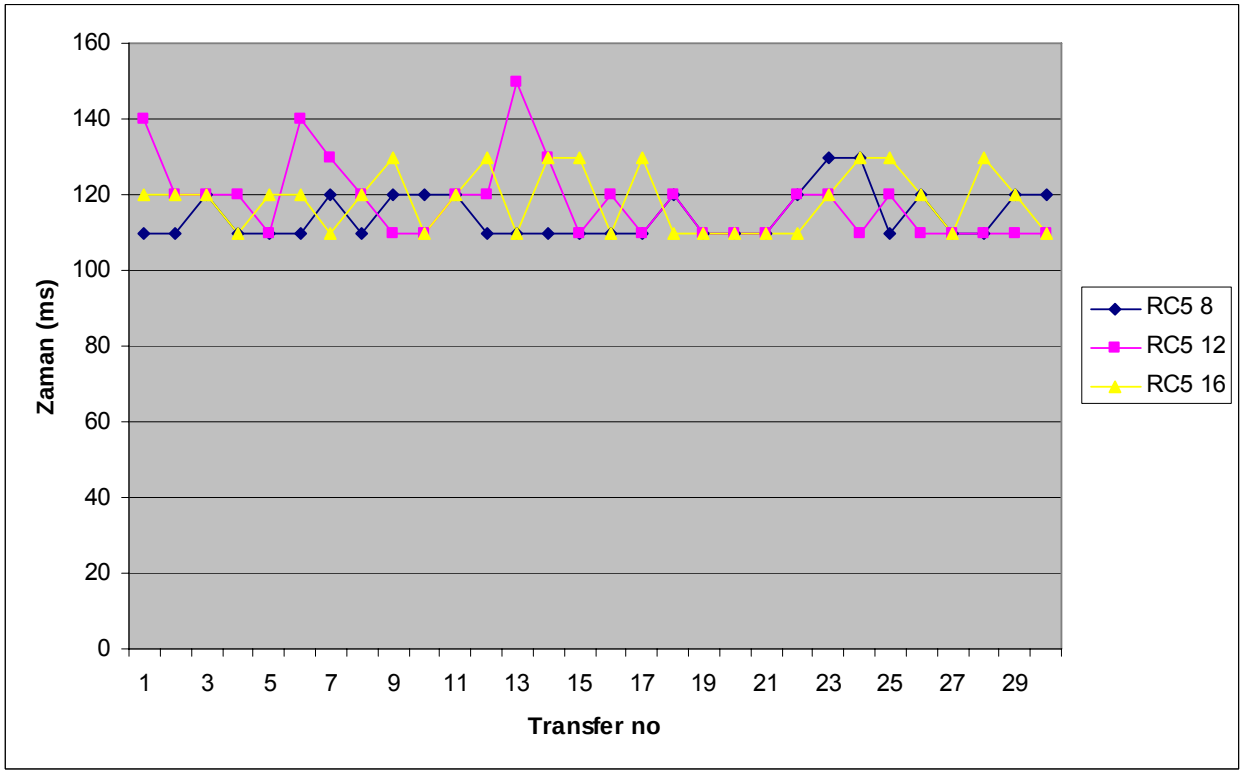
Çizelge 9.3'te 1MB dosyanın üç değişik durum için faklı kural sayılarına göre yapılan "100"er adet ölçümün ortalama sonuçları verilmiştir. Kural sayılarındaki artış RC5 ve AES algoritmaları ile yapılan ölçümlerde net olarak görülmektedir. Kural sayısı ile transfer süreleri arasında doğru orantı gözlenmiştir. En büyük sıçrama ise kural sayısı "1600" olduğunda tespit edilmiştir.

Çizelge 9.3 Kural sayısına göre 1MB dosyanın transfer süreleri (ms)

Yöntem / Kural Sayısı	PILSC	RC5	AES
10	87,5	114,6	144,6
20	87,7	115	145
40	87,8	116	146
80	87,9	119	147
160	88	119,3	147,3
1600	93,3	130	157,3

9.1.3 RC5 Algoritmasının Round Değişkenine Göre Dosya Transfer Süreleri

Kriptografik bir algoritmanın çeşitli parametreleri vardır. Bunlardan biri de karmaşıklığı arttırmak için veri üzerinde uygulanan kriptografik işlemlerin tekrar sayısının arttırılmasına dayanır. RC5 algoritmasında bu işlem için "round" değişkeni kullanılmaktadır. Round değişkeni 0-255 arasında belirlenebilir. Yapılan testte üç değer için 1MB dosyanın transfer süreleri elde edilmiştir ve Şekil 9.9'da gösterilmiştir.



Şekil 9.9 PILSC sisteminde RC5 algoritmasında kullanılan round sayısına göre elde edilen sonuçlar

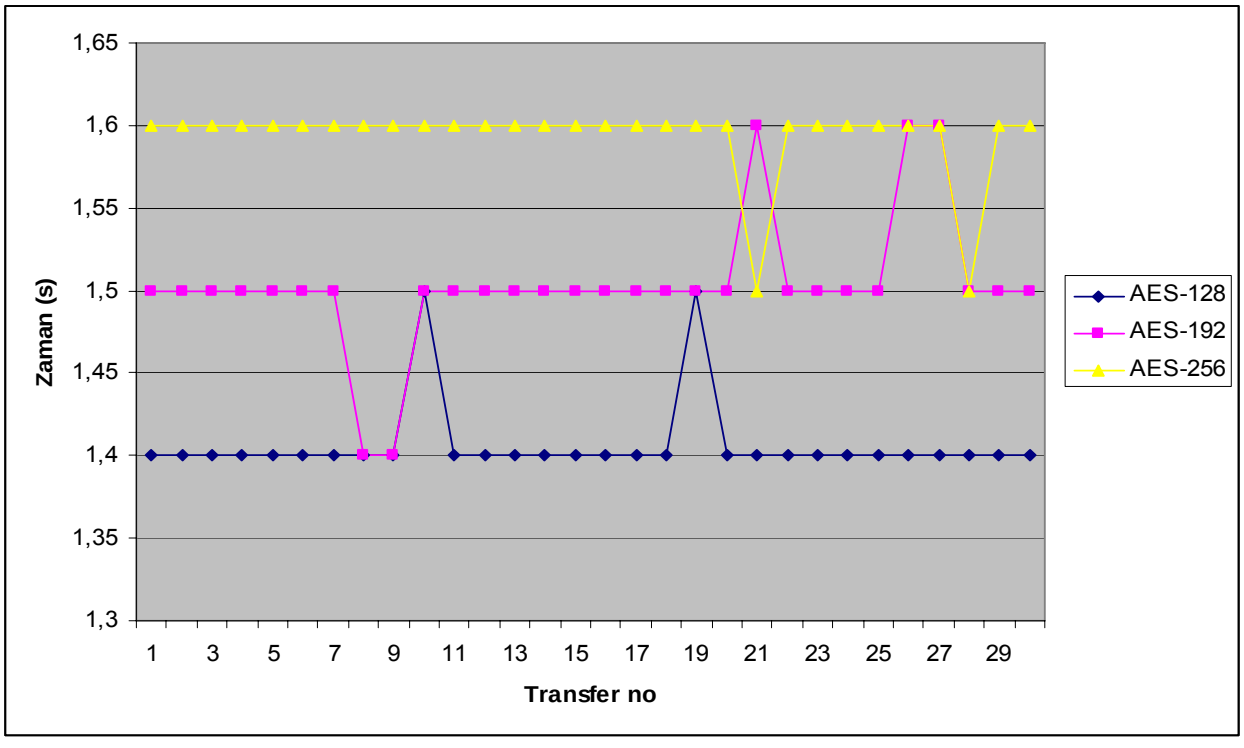
Şekil 9.9'da round sayısı arttığında transfer sürelerinin de arttığı gösterilmektedir. Çizelge 9.4'te ise round sayılarına ait ortalama transfer süreleri verilmiştir.

Çizelge 9.4 Round Sayısına göre ortalama dosya transfer süreleri (ms)

Round Sayısı	RC5
8	114,6
12	118,3
16	118,6

9.1.4 AES Algoritmasında Kullanılan Anahtar Değerine Göre Dosya Transfer Süreleri

Kriptografik işlemlerde verinin güvenliğini arttırmak, şifrenin kırılmasını zorlaştırmak için anahtar değerlerinin büyük olması istenir. Bu testte AES algoritması için üç farklı anahtar değeri kullanılmıştır. 1MB dosyanın transfer süreleri 128, 192, 256 bit anahtar değerleri için ölçülmüştür. Şekil 9.10'da beklenildiği gibi kullanılan anahtar değeri büyüdükçe dosya transfer süreleri artmıştır. Kriptografik işlemlerin getirdiği işlem yükünün zamana olan yansıması bu grafikte görülmektedir.

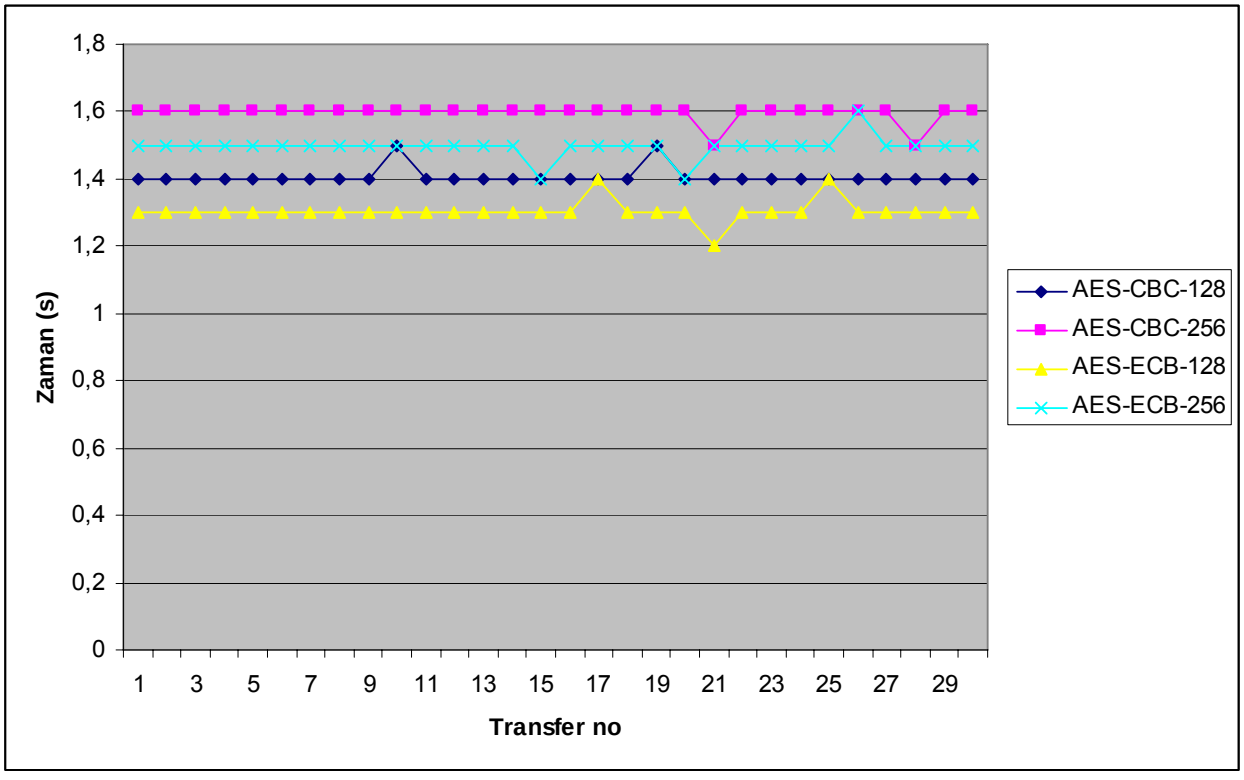


Şekil 9.10 AES algoritmasındaki anahtar değerlerine göre dosya transfer süreleri

9.1.5 AES Algoritmasında Kullanılan Farklı Şifreleme Yapılarının İncelenmesi

Şifreleme algoritmaları çeşitli ağ yapıları kullanırlar. Bunlardan en bilinenleri Cipher Block Chaining (CBC) ve Electronic Code Booking (ECB)'dir. CBC şifreleme sırasında aritmetik işlemler sırasında giriş değeri olarak bir önceki veriden yararlanırken, ECB giriş değeri almaz. Bu nedenle CBC yönteminin karmaşıklığının artmasından ötürü işlem süresi de uzar. AES algoritması kullanılarak yapılan ölçümlerde Şekil 9.11'deki grafik elde edilmiştir. Test sırasında 1MB dosyanın transfer süreleri ölçülmüştür. Şekil 9.11'de 128-bit ve 256-bit şifreleme için CBC ve ECB yöntemleri karşılaştırılmıştır. Sonuçta CBC yönteminin daha yavaş çalıştığı gözlenmiştir.

Grafik detaylı incelediğinde ECB yöntemi eşit anahtar değerlerinde daha hızlı çalışmasına rağmen ECB yönteminin ile 256-bit şifreleme kullanırken, 128-bit şifreleme yapan CBC yöntemine göre daha yavaş çalıştığı sonucuna varılmıştır.



Şekil 9.11 AES-CBC ve AES-ECB yöntemlerinin 128-bit ve 256-bit anahtar değerleri ile yapılan dosya transfer süreleri

9.1.6 ICMP Paketlerinin Büyüklüğüne ve Miktarına Göre Alınan Değerler

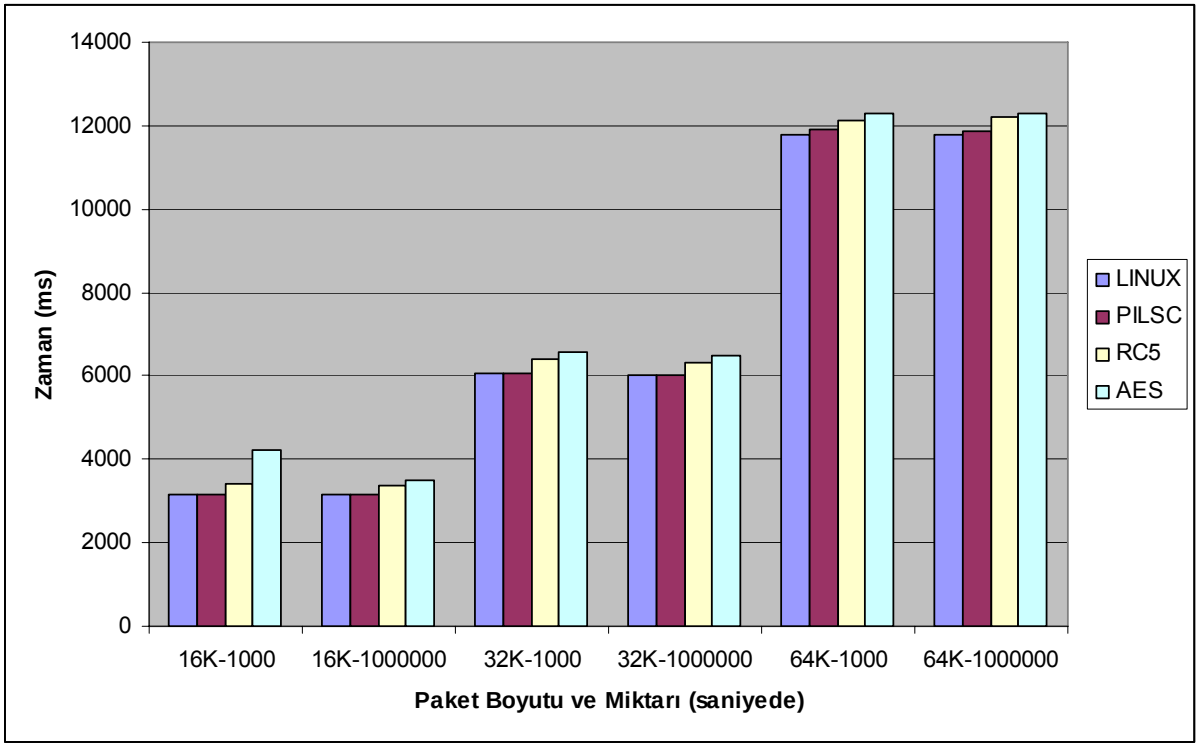
ICMP paketlerinin boyutları ve saniyede gönderilen paket sayısı değiştirilerek sisteme stres testi uygulanmıştır. Test sonuçları 5 dakika sonucunda elde edilen değerlerdir. Test sırasında uygulanan altı farklı senaryo şu şekildedir:

1. 16 KB büyüklüğünde saniyede 1000 ICMP paketi
2. 16 KB büyüklüğünde saniyede 1000000 ICMP paketi
3. 32 KB büyüklüğünde saniyede 1000 ICMP paketi
4. 32 KB büyüklüğünde saniyede 1000000 ICMP paketi
5. 64 KB büyüklüğünde saniyede 1000 ICMP paketi
6. 64 KB büyüklüğünde saniyede 1000000 ICMP paketi

Yukarıda belirtilen senaryolar dört farklı durum için değerlendirilmiştir.

1. Linux Paket İşleyicisi
2. PILSC
3. PILSC (RC5 şifreleme algoritması)
4. PILSC (AES şifreleme algoritması)

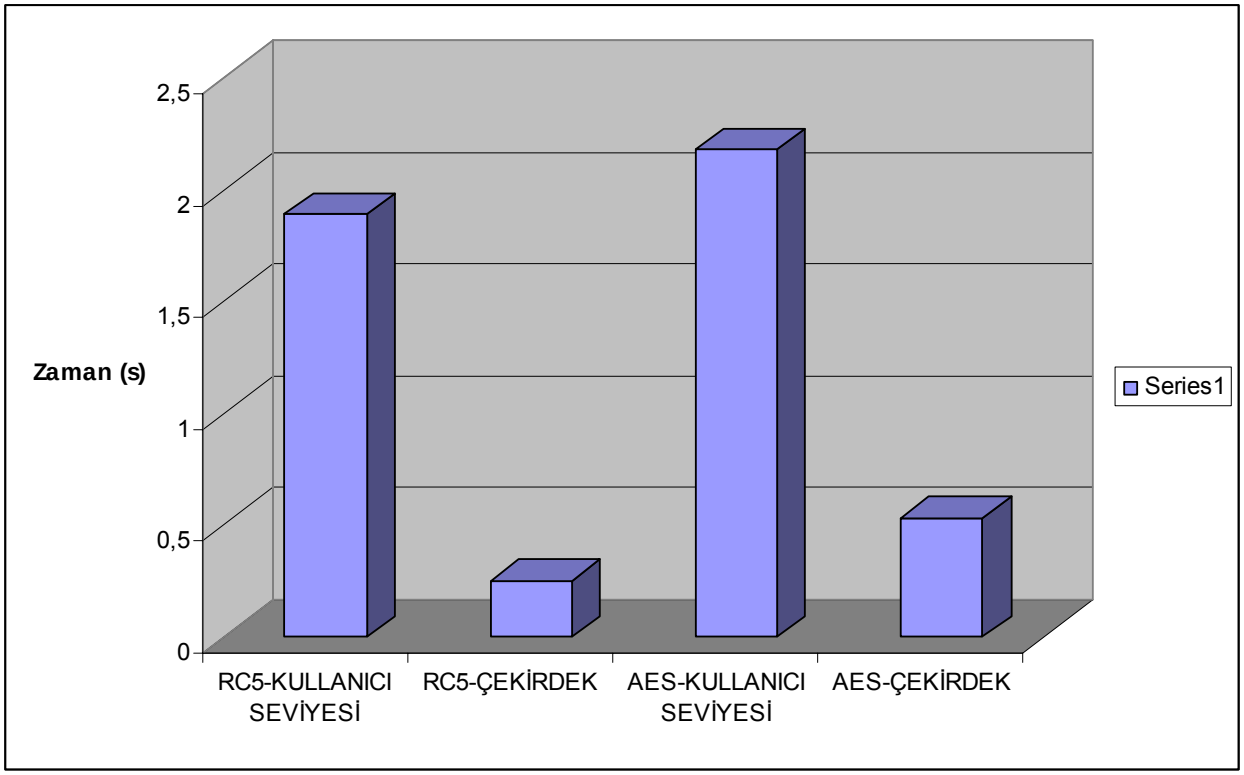
Şekil 9.12 incelediğinde Linux paket işleyicisi ile PILSC sisteminin stres testinde hemen hemen aynı performansı gösterdikleri görülmektedir. Paketlerin boyutları ve gönderim sıklığı artmasına rağmen dört farklı durumun birbirleri ile orantısında önemli bir değişim görülmemiştir. Beklenildiği gibi tüm senaryolarda en yavaş çalışan AES algoritması olmuştur.



Şekil 9.12 ICMP paketlerinin boyutlarına ve miktarlarına göre ölçülen değerler

9.1.7 RC5 ve AES algoritmalarının Kullanıcı Seviyesinde Ölçülen Şifreleme Süreleri

Bu tez çalışmanın en önemli amaçlarından biri güvenli veri iletişimini çekirdek seviyesinde sağlamaktır. Bunun nedeni çekirdek seviyesinde geliştirilen uygulamaların kullanıcı seviyesine göre çok daha hızlı çalışmasıdır. Bunu gösterebilmek için yapılan testte 10MB büyüklüğündeki verinin şifreleme ve deşifreleme süreleri ölçülmüştür. Çekirdek seviyesindeki kriptografik işlem süreleri ile karşılaştırmalı olarak Şekil 9.13'te gösterilmiştir.

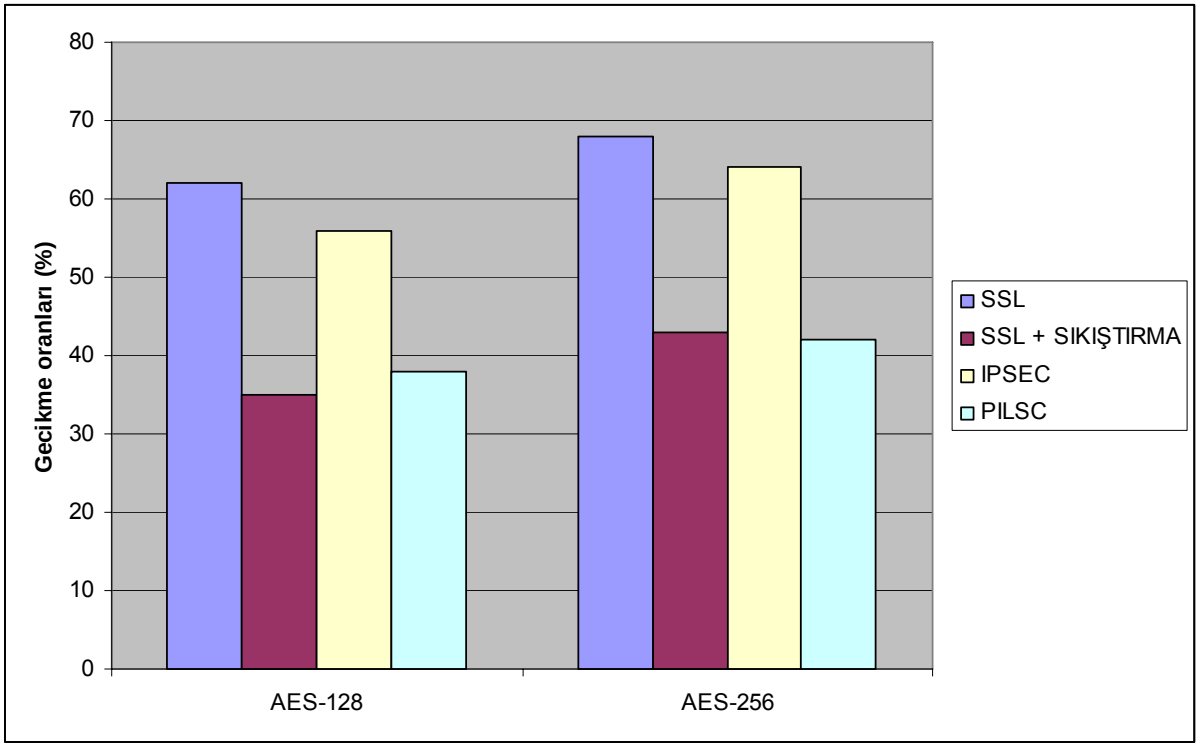


Şekil 9.13 Şifreleme algoritmalarının kullanıcı ve çekirdek seviyesinde çalışma süreleri (s)

Şekil 9.13'te görüldüğü gibi şifreleme algoritmalarının çekirdek seviyesinde çalışma süreleri çok büyük oranda azalmıştır. RC5 algoritması %87 oranında daha hızlı çalışırken, AES algoritması çekirdek seviyesinde %76 oranında daha hızlı kriptografik işlem gerçekleştirmiştir.

9.1.8 PILSC Sisteminin IPSec ve SSL Protokolleri ile Karşılaştırılması

PILSC sisteminde AES-128 bit ve AES-256 bit ile yapılan ölçümlerin değerleri ile IPSec ve SSL için aynı algoritmaların kullanılması ile elde edilen değerler karşılaştırılmıştır. Sonuçlar Şekil 9.14'de gösterilmektedir. IPSec ve SSL ile ilgili değerler Alshamsi ve Saito (2005) tarafından yapılan test sonuçları analiz edilerek elde edilmiştir. IPSec ve SSL protokolleri kimlik doğrulama (authentication) için hash fonksiyonu kullandıkları için PILSC ile karşılaştırmaları çok doğru olmasa da Şekil 9.14'de PILSC sisteminin performansı hakkında bir fikir vermektedir. Hash algoritmalarının sisteme çok yük getirmediği düşünüldüğünde PILSC sisteminin aynı algoritmalar için daha başarılı sonuçlar ürettiği söylenebilir.



Şekil 9.14 Farklı güvenlik protokollerinin paket transferinde gösterdikleri gecikme oranları

10. SONUÇ

Günümüzde güvenli veri iletişimini sağlayan uygulamaların, tüm IP paketlerine destek verebilmesi, olabildiğince çok şifreleme algoritması ile uyumlu çalışması, kolay yönetilebilir olması ve veri iletişiminde meydana gelebilecek gecikmeleri en az düzeyde tutabilmesi beklenmektedir. Gerçekleştirilen tez çalışmasında bu amaca yönelik bir mimari tasarlanmıştır. Bu mimari Linux işletim sistemi üzerinde, C programlama dili ve "Yüklenebilir Çekirdek Modülü" kullanılarak tasarlanmıştır. Oluşturulan sistem, kurulan test altyapısı üzerinde belirlenen senaryolar uyarınca test edilmiş, yapılan ölçümler analiz edilerek aşağıdaki sonuçlara varılmıştır.

1. Gerçeklenen sistem çekirdeğe dinamik olarak yüklenebilen bir modül şeklinde geliştirilmiştir. Bu yapı sistemin çalışmasını çekirdekten bağımsız hale getirmektedir.

2. "Yüklenebilir Çekirdek Modülü" kullanılarak tasarlanan sistemin işletim sisteminde çalışan herhangi bir servis gibi hızla yüklenebilmesi ve kaldırılabilmesi sağlanmıştır.

3. Sistemdeki kuralların yapılandırma dosyaları kullanılarak tanımlanması sistemin yönetimini kolaylaştırmıştır. Bu bilgilerin dosyalardan okunması sonucunda, sistemin devreye alınmasında oluşabilecek parametre karmaşası ortadan kaldırılmıştır.

4. Sistemin gerçeklenmesi sonucu oluşan güvenli veri iletişiminde, sadece tanımlı kurallara uygun paketler kriptografik işlem görürken, diğer paketlerin gecikme olmadan hizmet almaları sağlanmıştır.

5. Tasarımın gerçeklenmesinde kullanılan arama algoritmaları ile paketin kurallar içinde bulunması ve güvenlik alanı kontrol işlemleri en kısa sürede gerçekleşmektedir. Testler sonucunda PILSC sisteminin Linux paket işleyicisi ile hemen hemen aynı sürelerde hizmet verdiği belirlenmiştir.

6. Testler sonucunda PILSC sisteminin görülen en önemli avantajı güvenli veri iletişiminin çekirdek seviyesinde gerçekleştirilmesinin getirdiği performans artışıdır. Kriptografik işlemlerin çekirdek seviyesinde gerçekleştirilmesi %75-%90 oranında performans artışı sağlamıştır.

7. PILSC sisteminin kriptografik algoritmalardan bağımsız çalışabilme özelliği sistemin esnekliğini göstermektedir.

8. PILSC sisteminin IPSec ve SSL ile karşılaştırıldığında daha hızlı çalıştığı tespit edilmiştir. PILSC güvenli veri transferini IPSec ve SSL'e göre %20-%25 daha hızlı gerçekleştirir.

9. PILSC sistemi IP adresi, protokol ve port bazında farklı anahtar değerlerini destekleyebilmektedir.

Yapılan testler ve sistemin çalışması göz önüne alındığında, sistemde aşağıdaki özellikler dikkate alınarak ileriye yönelik geliştirmeler yapılabileceği sonucuna varılmıştır.

Sisteme anahtar dağıtım merkezi ile uyumlu çalışabilecek bir yapı eklenerek anahtar değişiminin güvenliğinin de kontrol edilebilmesi sağlanabilir. Mevcut sistem anahtar dağıtım merkezleri ile uyumlu hale getirilecek bir yapıya sahiptir.

KAYNAKLAR

Alshamsi, A. ve Saito T., (2005), "A technical Comparison of IPSec and SSL", Advanced Information Networking and Application 19. International Conference, AINA, 2005

Abramson, R., (2001) "Linux Looms Larger Than Thought", www.thestreet.com ,2001

Apostolopoulos, G., Peris, V. ve Saha, D., "Transport Layer Security: How much does it really cost?", 18. Annual Joint Conference of the IEEE Computer and Communications Societies Proceedings, INFOCOM, 1999

Bradford, Dr. E. G., "High-performance programming techniques on Linux and Windows – Pipes in Linux, Windows 2000 and Windows XP", IBM DeveloperWorks, Ekim 2001

IDC Consulting Team, (2004), "The Linux Marketplace – Moving From Niche to Mainstream", http://www.osdl.org, 2004

Ge, L., Scott, L. ve Vanderwiele, M. , (2004), "Putting Linux Reliability to the Test", www.opensourcetutorials.com, 19 Ocak 2004

Harrin, G., (2000), "Linux IP Networking: A Guide to the Implementation and Modification of the Linux Protocol Stack", 31 Mayıs 2000

Henderson, B., (2005), "Linux Loadable Kernel Module HOWTO", www.tldp.org, 20 Temmuz 2005

Kant, K., Iyer, R. ve Mohapatra, P.,(2000), "Architectural Impact of Secure Socket Layer on Internet Servers", 2000 International Conference on Computer Design 2000. Proceedings, 2000

Kossak, (1999), "Building Into The Linux Network Layer", Phrack Magazine, 9 Eylül 1999

Legard, D., (2004), "Rapid Linux Growth boosts server market" http://www.infoworld.com

Oppliger, R.,(1998), "Security at the Internet Layer", Volume 31, Issue 9, IEEE Computer Society, 1998

Pragmatic, (1999), "Nearly Complete Guide to Loadable Kernel Modules", Mart, 1999

Reilly R., (2003), "The Basics of Linux Network Security", www.linuxplanet.com

Saito, T., Kito, T., Umesawa, K. ve Mizoguchi, F., (2002), "Architectural Defects of the Secure Shell", 2002 Proceedings of 13th International Workshop on Database and Expert Systems Application, 2002

Schneier, B., (1996), "Applied Cryptography Protocols, Algorithms and Source Code in C", Second Edition, John Wiley & Sons, 1996

Tanenbaum, A., (1999), "Computer Networks", Third Edition, 1999

Yasinsac, A. ve Childs J., (2001), "Analyzing Internet Security Protocols", Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering, 2001

INTERNET KAYNAKLARI

[1] Mitchell J, http://theory.stanford.edu/people/jcm/home.html
[2] http://www.vandyke.com
[3] "TLS Protocol Version 1.0", RFC 2246
[4] http://www.networksorcery.com
[5] "Why Linux is a Better Choice than Windows", http://www.novell.com
[6] "Linux Marketplace Acceptance", http://helpdesk.du.ac.in
[7] http://vger.kernel.org/
[8] http://www.openssl.org
[9] http://www.ethereal.com
[10]http://en.wikipedia.org

ÖZGEÇMİŞ

Doğum tarihi	17.06.1978	
Doğum yeri	İstanbul	
Lise	1989-1997	İstanbul Erkek Lisesi
Lisans	1997-2002	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fak. Bilgisayar Mühendisliği Bölümü

Çalıştığı Kurumlar

2002-2003	YTÜ Bilgi İşlem Daire Başkanlığı
2003-Devam ediyor	YTÜ Elektrik-Elektronik Fakültesi Araştırma Görevlisi