Uygulama Katmanında Desen Arama Tabanlı Güvenlik Duvarı

Tolga Kızılkaya

T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

UYGULAMA KATMANINDA DESEN ARAMA TABANLI GÜVENLİK DUVARI

Tolga Kızılkaya
0000-0002-4822-0155

Doç. Dr. Murtaza CİCİOĞLU
(Danışman)

Dr. Cengiz TOĞAY
(İkinci Danışman)

YÜKSEK LİSANS
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BURSA – 2023

TEZ ONAYI

Tolga Kızılkaya tarafından hazırlanan "Uygulama Katmanında Desen Arama Tabanlı Güvenlik Duvarı" adlı tez çalışması aşağıdaki jüri tarafından oy birliği ile Uludağ Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman: Doç. Dr. Murtaza CİCİOĞLU

Başkan :	Doç. Dr. Murtaza CİCİOĞLU 0000-0002-5657-7402 Bursa Uludağ Üniversitesi, Mühendislik Fakültesi, Bilgisayar Yazılımı Anabilim Dalı	İmza
Üye :	Doç. Dr. Ali ÇALHAN 0000-0002-5798-3103 Düzce Üniversitesi, Mühendislik Fakültesi, Bilgisayar Yazılımı Anabilim Dalı	İmza
Üye :	Doç. Dr. PINAR KIRCI 0000-0002-0442-0235 Uludağ Üniversitesi, Mühendislik Fakültesi, Bilgisayar Bilimleri Anabilim Dalı	İmza

Yukarıdaki sonucu onaylarım
Prof. Dr. ……..…..
Enstitü Müdürü
../../….(Tarih)

Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

…/…/……

Tolga Kızılkaya

TEZ YAYINLANMA
FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezin/raporun tamamını veya herhangi bir kısmını, basılı (kâğıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma izni Bursa Uludağ Üniversitesi'ne aittir. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet hakları ile tezin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları tarafımıza ait olacaktır. Tezde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanılması zorunlu metinlerin yazılı izin alınarak kullandığını ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederiz.

Yükseköğretim Kurulu tarafından yayınlanan "**Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge**" kapsamında, yönerge tarafından belirtilen kısıtlamalar olmadığı takdirde tezin YÖK Ulusal Tez Merkezi / B.U.Ü. Kütüphanesi Açık Erişim Sistemi ve üye olunan diğer veri tabanlarının (Proquest veri tabanı gibi) erişimine açılması uygundur.

Doç.Dr. Murtaza CİCİOĞLU	Tolga KIZILKAYA
09.05.2023	09.05.2023
İmza	İmza
Bu bölüme kişinin kendi el yazısı ile okudum anladım yazmalı ve imzalanmalıdır.	Bu bölüme kişinin kendi el yazısı ile okudum anladım yazmalı ve imzalanmalıdır.

ÖZET

Yüksek Lisans

UYGULAMA KATMANINDA DESEN ARAMA TABANLI GÜVENLİK DUVARI

Tolga Kızılkaya

Bursa Uludağ Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Murtaza CİCİOĞLU

FreeBSD, servis sunucusu (web, http, ftp vb.), masaüstü kullanımı (X ile kde veya xfce) ve güvenlik duvarı (ipfw) olarak kullanılan bir işletim sistemidir. Freebsd işletim sistemi güvenlik duvarı olarak endüstride yaygın bir kullanıma sahiptir. pfSense ve OPNsense dağıtımları, Freebsd işletim sistemini kullanan ve endüstride birçok firma tarafından tercih edilen (Google, US Department of Homeland Security, Shopify ve NASA) açık kaynak kodlu güvenlik duvarlarıdır. Freebsd işletim sisteminde bulunan güvenlik duvarı IP ve Port adresi filtreleme işlemlerini yapabilmektedir. Ancak OSI modelinin yedinci katmanı olan uygulama katmanında filtreleme özelliğine sahip değildir. Pfsense ve Opnsense bu kısıtlamayı aşmak için snort ve suricata gibi açık kaynak kodlu yazılımları kullanmaktadır. Bu yazılımlar da işletim sistemlerinin kullanıcı uzayında (user-space) çalışabilmektedir. Bununla birlikte kullanıcı uzayında çalışan uygulamaların, çekirdek (kernel) seviyesinde çalışan uygulamalara göre daha fazla kaynak tükettiği de bilinmektedir. Bu tez çalışmasında Freebsd işletim sistemine ait güvenlik duvarı olan PF'in Pfill hook yöntemi kullanılarak uygulama katmanında filtreleme işlemi çekirdek seviyesinde gerçekleştirilmiştir. Paket içerisinde desen araması yapmak için Aho-Corasick algoritması kullanılmıştır. Önerilen teknik pfsense işletim sistemi üzerinde suricata yazılımı ile aynı kural sayısında bant genişliğinin kullanımı ve işlemci tüketimi açısından karşılaştırılmıştır. Karşılaştırma sonucunda bu tez kapsamında geliştirilen uygulamanın aynı kural sayısında yaklaşık 2 kat daha fazla bant genişliğinde filtreleme yapabildiği ve aynı kural sayısı ile aynı bant genişliğinde yaklaşık %20 daha az işlemci gücü tükettiği tespit edilmiştir. Ayrıca önerilen teknik son zamanlarda bilgisayar ağ dünyasında yaygın bir kabul gören yazılım tanımlı ağ (YTA) mimarisine uygulanmış ve literatürde YTA mimarisinin uygulandığı çalışmalarla karşılaştırılmıştır. Literatürdeki diğer çalışmalardan uygulama kolaylığı ve kaynak kullanımı açısından daha iyi olduğu ortaya konmuştur.

Anahtar Kelimeler: Desen Arama Algoritmaları; Çekirdek Programlama; FreeBSD; Güvenlik Duvarı; Katman 7 Filtreleme; Yazılım Tanımlı Ağlar
2023, vii + 38 sayfa.

ABSTRACT

MSc

PATTERN SEARCH-BASED FIREWALL IN APPLICATION LAYER

Tolga Kızılkaya

Bursa Uludağ University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Murtaza CİCİOĞLU

FreeBSD is an operating system used as a service server (web, http, ftp etc.), desktop use (kde or xfce with X) and firewall (ipfw). Freebsd is widely used in the industry as an operating system firewall. pfSense and OPNsense distributions are open source firewalls that use the Freebsd operating system and are preferred by many companies in the industry (Google, US Department of Homeland Security, Shopify and NASA). The firewall in the Freebsd operating system can perform IP and Port address filtering. However, it does not have filtering in the application layer, which is the seventh layer of the OSI model. Pfsense and Opnsense use open source software such as snort and suricata to overcome this limitation. These software can also work in the user-space of operating systems. However, it is also known that applications running in user space consume more resources than applications running at kernel level. In this thesis, the filtering process in the application layer is carried out at the kernel level by using the Pfill hook method of PF, which is the firewall of the Freebsd operating system. Aho-Corasick algorithm was used to search for patterns in the package. The proposed technique was compared with the suricata software on the pfsense operating system in terms of bandwidth usage and processor consumption in the same rule number. As a result of the comparison, it has been determined that the application developed within the scope of this thesis can filter approximately 2 times more bandwidth in the same number of rules and consumes approximately 20% less processing power in the same bandwidth with the same number of rules. In addition, the proposed technique has been applied to the software-defined network (YTA) architecture, which has been widely accepted in the computer networking world, and has been compared with the studies in the literature where YTA architecture has been applied. It has been shown to be better than other studies in the literature in terms of ease of application and resource use.

Key words: Computer Networks; Pattern Searching Algorithm; Kernel Programming; FreeBSD; Firewall; Layer 7 Filtering; SDN

2023, vii + 38 pages.

ÖNSÖZ VE/VEYA TEŞEKKÜR

Çalışmamız için deneyimlerini, akademik bilgi ve tecrübelerini aktaran danışmanım Doç. Dr. Murtaza CİCİOĞLU'na sonsuz teşekkürlerimi sunarım.

Çalışmamızın geliştirme ve yazım süreçlerinde siber güvenlik ve yazılım sektöründeki bilgi ve tecrübelerini paylaşan Dr. Cengiz TOĞAY'a teşekkürlerimi sunarım.

Yüksek lisans süresince hep yanımda olan ve desteklerini esirgemeyen eşim Esra Kızılkaya'ya en içten duygularımla sonsuz sevgilerimi ve teşekkürlerimi sunarım.

Yüksek lisans yapmak için beni teşvik eden ve yüksek lisans süresince maddi ve manevi desteklerini esirgemeyen çalıştığım kurum olan BG-Tek Bilgi Güvenliği Teknolojileri firmasına teşekkürlerimi sunarım.

Tolga Kızılkaya
09/03/2023

İÇİNDEKİLER

Sayfa

SİMGELER ve KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
AC	Aho–Corasick
BM	Boyer-Moore
BMH	Boyer–Moore–Horspool
CDAWG	Compact Direct Acyclic Word Graph
QWM	Improved Wu-Manber
WM	Wu-Manber
CW	Commentz-Walter
HBMH	Hash Boyer Moore Horspool
Nw2	Improved Wu-Manber
SBMH	Set-wise Boyer-Moore-Horspool
FvH	SBMH ile aynı makalede farklı ifade edilmiş
SWBM	Set-wise Boyer-Moore
AC_BM	Boyer-Moore like algorithm
E2xB	Exclusion-based string matching
NFA	Non-Deterministic finite automata
DFA	Deterministic Finite Automat
YTA	Yazılım Tanımlı Ağ
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
PDU	Protocol Data Unit
IPS	Intrusion Prevention System

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

Sayfa

1.GİRİŞ

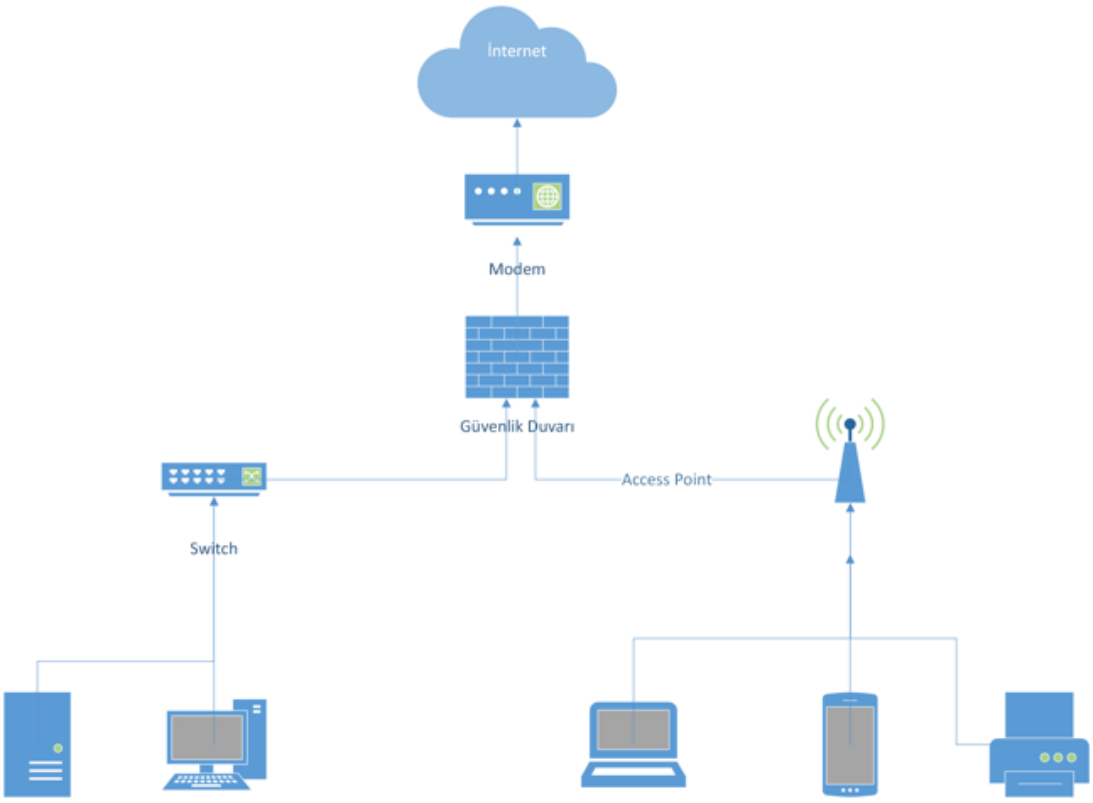
Siber Güvenlik, hassas bilgilerin ve bilgi sistemlerinin yetkisiz erişimlere, veri hırsızlığına, veri değişikliğine, veri bozulmasına karşı korunmasını olarak tanımlanmaktadır. Bilgisayar ağlarında güvenliğin (siber güvenlik) önemi her geçen gün artarak devam etmektedir. Özellikle, kurumsal ve kişisel bilgisayarlara sızılması, servis reddi (dağıtık hizmet engelleme (ddos)) saldırıları, oltalama (phishing) saldırıları, web sitelerinin ve kişisel verilerin ele geçirilmesi gibi siber güvenlik olayları bu konunun önemini ortaya koymaktadır. Bu olayların sonucunda birçok kurum ve kişi hem maddi hem de manevi zararlara maruz kalabilmektedir.

Dijital dünyanın bu zararlarından korunmak için yazılım ya da donanım şeklinde çözüm üretebilen çeşitli güvenlik duvarları bulunmaktadır. Güvenlik duvarları genel olarak, bir özel ağ ile internet arasına konularak, özel ağın internet tarafından gelecek saldırılara karşı korunmasını veya içeriden dışarıya çıkacak istenmeyen trafiğin engellenmesini sağlayan bir araç olarak ifade edilebilir. Gelen ve giden tüm ağ paketlerinin incelenmesi ve önceden tanımlı kurallara göre bu paketlerin kabul edilmesine veya engellenmesine karar vermek güvenlik duvarının en önemli işlevlerinden birisidir. (Gouda ve Liu, 2007).

Güvenlik duvarının bilgisayar ağ topolojisindeki konumu genel olarak Şekil 1.1'de gösterildiği gibi iç ağ ile internet arasındaki sınır bölgesindedir. Bu noktada güvenlik duvarı, üzerinden geçen trafiği inceleyebilmekte ve trafiğe müdahale edebilmektedir. Güvenlik duvarında gelen ve giden tüm paketlerin kontrol edilmesi, güvenlik duvarından geçecek paketlere karar verilmesi işlemine ise filtreleme adı verilmektedir. Filtreleme, mesai saatlerinde sosyal medya uygulamalarının açılmasının yasaklanması ya da virüslü bir dosyanın bilgisayara indirilirken engellenmesi gibi birçok kural tabanlı işlemlerden oluşmaktadır. Bir güvenlik duvarı filtreleme yaparken veri paketinin katmanlarını incelemektedir.

OSI modelinde gönderilecek mesaj her katmanda o katmana ait fonksiyonları barındıran bilgileri alıp kapsüllenerek bir alt katmana iletilmektedir. Aynı şekilde alıcı tarafta da her katman kendisine ait bilgileri kapsülden çıkararak ilgili kontrolleri gerçekleştirmekte ve

bir sorun yoksa bir üst katmana iletmektedir. OSI ve TCP/IP modeli güvenlik endişeleri barındırmadan geliştirilmiş protokol kümeleridir. Bu bağlamda güvenlik işlemleri sonradan eklenen yeni protokoller ile güçlendirilmektedir.



Şekil 1.1. Güvenlik Duvarı

Her katmanın bilgisine Protocol Data Unit (PDU) denir. Verilerle birlikte protokole özel kontrol bilgilerini bulundurur. Bir PDU her katmanda kendi protokol bilgilerini ekler veya siler. PDU'ya rolünü temsil etmesi için her katmanda farklı isimler verilir. OSI modeline göre katman 1'de bit, katman 2'de frame, katman 3'te paket, katman 4'te eğer bir TCP paketi ise segment, UDP paketi ise datagram, katman 5 ve üzeri ise veri olarak isimlendirilir.

OSI modelinin katmanlarında, güvenlik duvarına girilen kurallara göre filtrelemeler yapılmaktadır. Veri paketinin başlık bilgisinde bulunan MAC adresi katman 2'de, IP adresi katman 3'te ve port bilgisi ise katman 4'te yer almaktadır. Örneğin, önceden tanımlı bir IP adresinin filtreden geçilmesine engel olabilmek için bir kural

tanımlanabilmekte ve güvenlik duvarı paketin yalnızca katman 3'te IP başlık bilgisine bakarak bu filtrelemeyi gerçekleştirebilmektedir. Ancak 2, 3 ve 4. katmanda yapılan bu filtrelemeler trafik sınıflandırma ve uygulama tespiti için yeterli olmamaktadır (Dragos ve Avram, 2010).

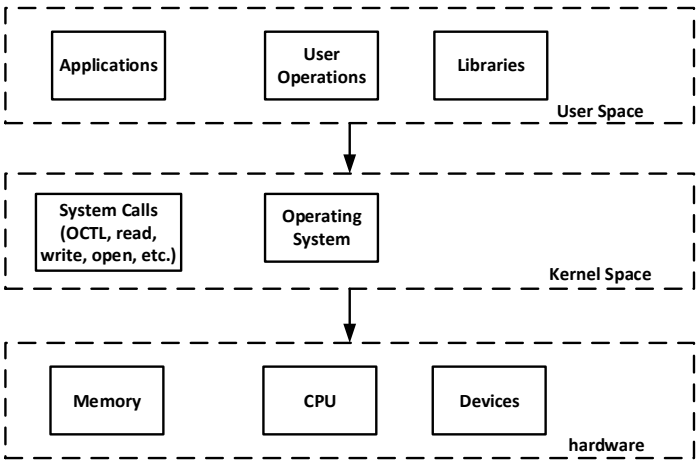
Bir diğer taraftan uygulama katmanı olan katman 7'de uygulamaya göre filtreleme yapılabilmektedir. Katman 7'de ip başlığının veri kısmına bakılarak uygulama tanımlanabilmektedir. Bu sayede daha detaylı bir filtreleme sağlanmaktadır. Ancak katman 7'de filtreleme yapmak diğer katmanlara göre daha maliyetli olabilmektedir. Bunun sebebi, veri paketinin tamamının incelenmesi ve içerisinde kuralların aranmasıdır. Bu durum 7. katmanda yapılan filtrelemeleri kaynak tüketimi açısından verimsiz hale dönüştürmektedir.

Güvenlik duvarları hem donanım hem yazılımla birlikte veya sadece yazılımsal olarak özel ağlara entegre edilebilmektedir. Ağda bulunduğu konum itibarıyla hızlı ve güvenilir işletim sistemleri üzerinde çalışması oldukça önemlidir. Freebsd işletim sistemi endüstri tarafından tercih edilen uzun yıllardır geliştirilmiş ve geliştirilmeye devam edilen tüm ayarlamaları iyi bir şekilde yapılmış son derece güvenli ve kararlı bir işletim sistemidir (Stewart ve Healy, 2010). Bu nedenle güvenlik duvarı uygulamalarında sıkça kullanılmaktadır. Hatta pfsense ve opnsense gibi birçok açık kaynak kodlu güvenlik duvarı da freebsd işletim sisteminin güvenlik duvarı olan paket filtreleme (PF) kullanmaktadır.

Freebsd ve linux açık kaynak kodlu olmasından dolayı benzer işletim sistemleri olarak görülebilmektedir. Fakat freebsd işletim sistemleri unix çekirdeğini kullanırken linux işletim sistemi ise linux çekirdeğini kullanmaktadır. Ayrıca freebsd işletim sistemi linux işletim sistemine göre daha hızlı ve kararlı olmakla birlikte, birçok uygulama freebsd işletim sisteminde daha hızlı ve kararlı çalışabilmektedir. (Reinholz Dan ve Reinholz Kevin, 2022). Sadece grafik tabanlı bazı uygulamalar linux işletim sistemi tabanlı yazıldığından freebsd işletim sistemine taşındığında daha yavaş çalışabilmektedir. Ayrıca uygulamaların (iki işletim sistemi tarafından desteklenen açık kaynak kodlu uygulamalar) güncel sürümlerinin freebsd işletim sistemine uyumlu hale dönüşmesi de zaman

alabilmektedir. Bu durumun en önemli sebebi freebsd işletim sisteminin kararlı hale gelen uygulamaları entegre etmesidir (Reinholz Dan ve Reinholz Kevin, 2022). Bir diğer taraftan freebsd tabanlı macosx ele alındığında ise macosx'in kullanımının daha kolay olduğu görülmektedir (Reinholz Dan ve Reinholz Kevin, 2022). Macosx lisanslı bir sistem yazılımı olmakla birlikte özel sistemlerde çalışmaktadır. Ayrıca freebsd işletim sisteminden gelen iyileştirmeleri alan ve aynı zamanda kendi iyileştirmelerini de gönderen bir etkileşim bulunmaktadır. (Reinholz Dan ve Reinholz Kevin, 2022). Bu çalışmada ücretsiz, güvenli ve kararlı olmasından dolayı freebsd işletim sistemi tercih edilmiştir.

Şekil 1.2'de görüldüğü gibi işletim sistemleri sanal bellek alanını kullanıcı uzayı (user-space) ve çekirdek uzayı (kernel space) alanı olarak ikiyi ayırmaktadır. Çekirdek uzayı işletim sistemi çekirdeğini, çekirdek uzantılarını ve aygıt sürücülerini barındıran bellek alanıdır. Kullanıcı uzayı ise kullanıcı uygulamalarının çalıştığı bir bellek alanıdır (Bellevue Linux Users Group, 2005). Çekirdek uzayında çalışan ağ uygulamaları kullanıcı uzayında çalışan ağ uygulamalarına göre daha az kaynak (işlemci) tüketmektedirler ve paket aktarımında daha yüksek bant genişliğine ulaşabilmektedirler (Minghao vd., 2007). Bu nedenle bu tez çalışmasında çekirdek uzayında çalışacak uygulama üzerinde yoğunlaşılmıştır.



Şekil 1.2. Kullanıcı ve Çekirdek Uzayı

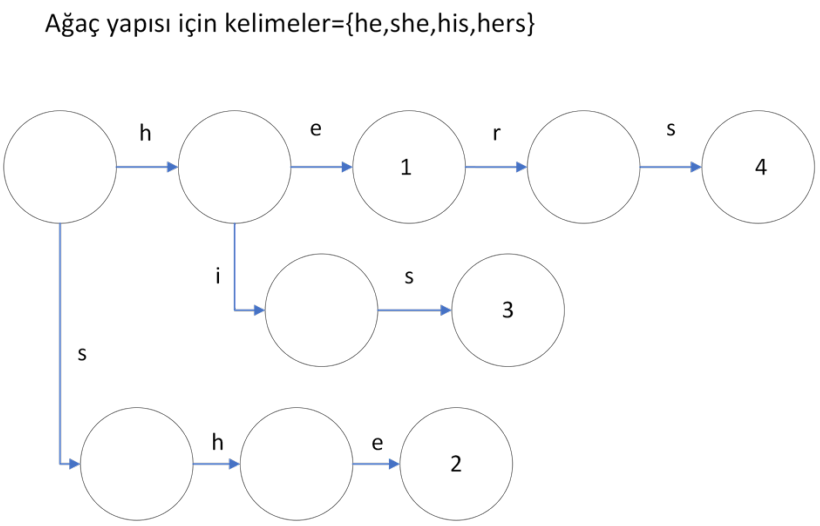
Bir diğer taraftan bu tez çalışmasında veri paketi içerisinde arama yapmak için çekirdek uzayında desen arama algoritmaları kullanılmıştır. Desen arama algoritmalarına dizi arama algoritmaları da denir ve dizi arama algoritmalarının bir parçası olarak kabul edilir. Bu algoritmalar, başka bir desen içinde bir desen arama durumunda kullanılır.

Desen arama işlemi ağ güvenliğinde imza tabanlı saldırı tespiti (IDS/IPS), arama motorları, tıp bilimi, biyoinformatik, metin editörleri, veri tabanı sorguları, görüntü işleme, web filtreleme yazılımları, anti-virüsler, anti-spam yazılımları gibi çok geniş uygulama alanına sahiptir (Le Dang vd., 2016). Bu çalışmada uygulama imzalarını akan trafik içerisindeki veri paketlerinde gerçek zamanlı olarak aramak için desen arama algoritması kullanılmıştır.

Desen arama algoritmalarından biri olan aho-corasick algoritması, Alfred V. Aho ve Margaret J. Corasick tarafından geliştirilmiştir. Bu algoritma bir giriş metnindeki sonlu dizeleri bulur. Desen sözlüğü önceden bilindiğinde (bilgisayarlarda virüslerin imza veri tabanı gibi), arama işlemi çevrimdışı olarak gerçekleştirilebilir ve sonuçlar daha sonra kullanılmak üzere saklanabilir. Bu durumda, çalışma süresi, girişin uzunluğu artı eşleşen girişlerin sayısı bakımından doğrusaldır. Aho–Corasick desen arama algoritması, unix/linux sistemlerdeki fgrep komutunun temelini oluşturur (Govil Ayush, 2022).

Aho-Corasick algoritması tüm anahtar kelimeleri içeren bir ağaç yapısı kullanır. Aho-Corasick algoritmasının üç farklı aşaması vardır. Bunlar; Go-to, Failure, ve Output'tur. Go-to aşamasında, tüm anahtar kelimeler kullanılarak ağaç oluşturulur. Sonraki aşama olan Failure aşamasında, bazı anahtar kelimelerin uygun bir son ekini almak için geriye doğru geçişi bulmaya çalışır. Output aşamasında, otomatın her "s" durumu için, "s" durumunda biten tüm kelimeleri bulur.

Aho-corasick algoritmasındaki örnek ağaç yapısı Şekil 1.3'te verilmiştir. Ağaç oluşturulduktan sonra doğrusal zamanlı arama için bir otomata dönüştürülür. Bir dizi içerisinde başarılı ve başarısız geçişler saklanır (Govil Ayush, 2022).



Şekil 1.3. Aho-corasick ağaç yapısı

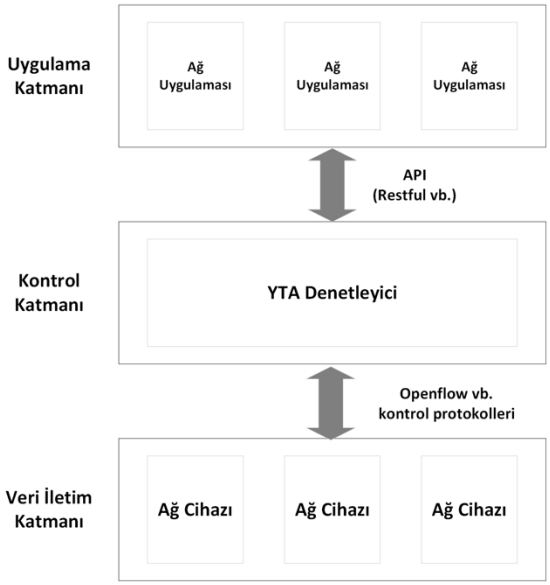
Bir diğer desen arama algoritması ise Boyer-Moore(BM) algoritması Robert Boyer ve J Strother Moore tarafından 1977'de yazılan bir desen arama algoritmasıdır. BM desen arama algoritması verimli bir algoritmadır ve yazıldığı zamandan beri desen arama algoritmaları için standart bir kıyaslama görevi görmüştür (Hume ve Sunday, 1991).

BM algoritması 'geriye doğru' bir yaklaşım benimsemektedir: Desen dizisi (P), metin dizesinin (T) başlangıcı ile hizalanmaktadır ve ardından bir desen karakterlerini sağdan sola, en sağdaki karakterden başlayarak karşılaştırmaktadır.

Uyumsuz karşılaştırma denk gelirse, olası kaydırmalara karar vermek için, BM algoritması aynı anda iki ön işleme stratejisi kullanır. Bir uyumsuzluk meydana geldiğinde, algoritma her iki yaklaşımı kullanarak bir varyasyonu hesaplar ve her durum için en etkili stratejiyi kullanır. Böylece daha önemli olan kaymayı seçer. Bu yaklaşımlar iyi sonek (good suffix) ya da kötü eşleşme (bad match) olarak bilinir (Gharaee vd., 2014).

Yazılım tanımlı ağ (YTA), görevlerine göre yerleştirilmiş ağ cihazlarından oluşan ve her cihazın yapılandırmasının ayrı ayrı yapıldığı geleneksel ağları, programlanabilen yazılım tabanlı, esnek, merkezi bir yönetim ortamına dönüştüren yeni nesil bir ağ teknolojisi yaklaşımıdır. Geleneksel ağlarda ağ aygıtları kontrol ve veri düzlemi fonksiyonlarını bütünleşik bir biçimde barındırmakta ve üretici bağımlılığına sahiptir. YTA yaklaşımında

ise Şekil 1.4'te görüldüğü gibi bütünleşik bir mimari bulunmaktadır. YTA mimarisinde kurallar mantıksal olarak yapılandırılmış merkezi bir noktadan yönetildiği için ağ cihazları üzerinde kural karmaşıklığı azalmaktadır. YTA mimarisinde de geleneksel ağlardaki zafiyetler (malware, trojan, dos vs.) olmakla birlikte denetleyici ya da denetleyicilere yapılacak saldırıların da olabileceği görülmektedir. Ancak geleneksel ağların aksine, YTA mimarisinde tek bir noktada tespit edilen yazılım mimariye dahil olan tüm ağda engellenebilmektedir.



Şekil 1.4. YTA mimarisi

Bu tez çalışmasında güçlü, kararlı ve güvenilir bir işletim sistemi olan FreeBSD üzerinde güvenlik duvarı için çekirdek seviyesinde çalışan katman 7 filtrelemesi geliştirilmiştir. Buradaki en önemli amaç kaynakların daha verimli bir şekilde kullanılmasını sağlayabilmektir. Çekirdek seviyesinde çalışan uygulamada daha az CPU tüketirken daha yüksek bant genişliğinde filtreleme yapılabilmektedir. Önerilen uygulamada Aho-Corasick ve Boyer-Moore dizi arama algoritmaları kullanılmış ve hem sanal hem de gerçek ortamda test edilmiştir. Ayrıca önerilen teknik açık kaynak olan ve kullanıcı uzayında çalışan IDS/IPS olarak kullanılan suricata yazılımı ile de karşılaştırılmıştır. Geliştirilen teknik freebsd üzerinde çalışabildiği gibi pfsense, opnsense gibi endüstri dünyasında sıkça kullanılan freebsd işletim sistemi temelli güvenlik duvarları üzerinde de çalışabilmektedir. Bunun yanı sıra verilen kurallara göre içeriden dışarıya

kullanıcıların internet filtrelemesi, dışarıdan içeriye IDS/IPS, WAF, SIP güvenlik duvarı uygulamaları olarak da kullanılabilecek genel bir yapı oluşturulmuştur. Ayrıca oluşturulan yapı YTA mimarisine uygulanarak farklı senaryolar ile de analiz edilmiştir.

Bu tez çalışmasının ana katkılarını sıralayacak olursak;

- Açık kaynak kodlu freebsd işletim sistemine katman 7 filtreleme özelliği eklenmesi
- Aho-corasick algoritmasının çekirdek uzayında uygulanması
- Çekirdek uzayında filtreleme yapılarak daha fazla bant genişliğinde filtreleme yapılabilmesi
- Çekirdek uzayında filtreleme yapılarak CPU tüketiminin düşürülmesinin sağlanması
- Çekirdek uzayında geliştirilen teknik ile kullanıcı uzayındaki suricata yazılımının karşılaştırılması
- YTA mimarisinde katman 7 filtrelemesinin uygulanması

2. KAYNAK ARAŞTIRMASI

2.1. Algoritmalar

IDS (intrusion detection system), internet üzerinden gelebilecek kötü niyetli saldırıları savunmak için kullanılan güçlü bir yazılımdır. Saldırıyı çevrimiçi olarak tespit etmek amacıyla IDS, kötü amaçlı kodu açığa çıkarmak için paketleri çok hızlı bir şekilde incelemektedir. Desen arama algoritmaları, IDS'lerde sıkça kullanılmaktadır. Yang Dong ve diğerleri (2006), desen arama algoritmasının performansını artırmak için, geliştirilmiş bir Wu-Manber algoritması olan QWM (Improved Wu-Manber Algorithm) algoritmasını önermişlerdir. Bu çalışmada önerilen QWM algoritması, Aho-Corasick, Commentz-Walter ve Wu-Manber algoritmalarıyla karşılaştırılmıştır. Çalışmada Aho-Corasick algoritmasının karmaşıklığı O(nlogx) olarak verilmiştir. X parametresi, alfabenin boyutu olarak ele alınmıştır. Bu yüzden Aho-Corasick algoritmasının çalışma süresinin, desen sayısından bağımsız olduğu belirtilmiştir. Çalışmada karşılaştırma için kullanılan Commentz-Walter algoritması da Boyer-Moore tekniğini Aho-Corasick algoritması ile birleştirmiştir. Çalışmada test sonuçları, İngilizce metin ve Çince metin gibi büyük alfabelerde, QWM algoritmasının daha iyi performansa sahip olduğunu, diğer üç algoritmadan daha hızlı olduğunu göstermiştir. Ayrıca QWM algoritmasının bellek kullanımının diğer algoritmalara göre daha az olduğu tespit edilmiştir.

İmza tabanlı ağ saldırı tespit araçlarının performansında, paketlerin birçok imzaya karşı desen araması önemlidir. Fisk ve Varghese (2001), popüler saldırı tespit sistemi Snort'un farklı desen arama algoritmalarını kullanmak için en iyi şekilde nasıl optimize edilebileceği incelemiştir. Snort'un mevcut desen arama algoritması Boyer-Moore'un ve çeşitli alternatif algoritmaların performansı analiz edilmiştir. Varsayılan olarak Snort uygulamaları, yaygın olarak bilinen herhangi bir algoritmanın en iyi ortalama durum performansını sağlaması olarak kabul edilen Boyer-Moore desen arama algoritmasını kullanır. Algoritma, ortalama durumda alt doğrusal zamanda tek modellerin bulunmasına izin veren birkaç temel gözlemi temel alır. Boyer-Moore algoritmasını uygulamanın karmaşıklığı, daha fazla kullanılmamasının nedenlerinden biri olarak gösterilmiştir. Çalışmada, desen arama algoritmaları kullanılarak genel amaçlı bir kütüphane geliştirilmiştir. Bu kütüphane C programlama dili ile yazılmıştır ve diğer projeler

tarafından kullanılmak üzere serbestçe dağıtılacağı belirtilmiştir. Kütüphanede Boyer-Moore, Boyer-Moore-Horspool, Aho-Corasick ve SBMH algoritmaları uygulanmaktadır. Daha eşit bir karşılaştırma için, hem Aho-Corasick hem de SBMH algoritmalarının uygulamaları aynı trie (digital tree ya da prefix tree olarakta bilinir) veri yapısını kullanmaktadır. Ölçümler, 256KB yonga üzerinde önbelleğe sahip ve Linux 2.2.18 çekirdeği çalıştıran bir 733MHz Pentium III (Coppermine) sisteminde yapılmıştır. Desen eşleştirme kütüphanesi kullanmak için değiştirilmiş Snort 1.6.3-p2 sürümü kullanılmıştır. Kullanılan Snort kural seti Snort 1.6.3-p2 dağıtımından "ping" ve "arka kapı" kuralları devre dışı bırakılmış (ortak bir yapılandırma) tam kural setidir. Zamanlama bilgileri gprof aracı ile elde edilmiştir. Desen boyutu arttırılarak yapılan analizde, desen boyutu 100 karaktere kadar olan aramalarda çalışma süresi en az olan algoritma SBMH algoritması olarak bulunmuştur. Desen boyutu 100 karakterden uzun aramalarda çalışma süresi en az olan algoritma Aho-Corasick olarak bulunmuştur.

Vidanagamachchi ve diğerleri (2012), Commentz-Walter algoritmasını incelemiş ve bu algoritmayı AC ve BM algoritmaları ile karşılaştırmıştır. Çalışmaya göre Commentz-Walter algoritması hem Aho-Corasick hem de Boyer Moore'dan gelen fikirleri birleştirir. Çalışmada her iki algoritma da Visual Studio 2008 ortamında uygulanmış ve 4GB RAM ve 32bit işletim sistemine sahip Intel Core i7 2.2 GHz makinede çalıştırılmıştır. Aho-Corasick'in ortak inançların aksine Commentz-Walter'dan daha iyi performans gösterdiği sonucuna varılmıştır.

Anagnostakis ve diğerleri (2003) çalışmasında ağa İzinsiz Giriş Tespit Sistemlerinde (NIDS) desen eşleştirme sorunu ele almıştır. NIDS'lerde en büyük maliyet desen eşleme işleminden oluşmaktadır. Çalışmada NIDS'lerin verimliliğini ve kapasitesini attırmak amacıyla NIDS desen eşleştirmesinin belirli özelliklerine göre uyarlanmış yeni bir desen arama algoritması E2xB önermişlerdir. Yeni geliştirilen algoritma popüler bir açık kaynaklı NIDS olan snort aracına uygulanmış ve AC algoritması ile karşılaştırılmıştır. Çalışmada tüm testler için, 1.7 GHz hızında çalışan Pentium 4 işlemcili, 8 KB L1 önbelleği ve 256 KB L2 önbelleği ve 512 Mbyte ana belleğe sahip bir bilgisayar kullanılmıştır. Ölçülen bellek gecikmesi, L1 önbellek için 1 ns, L2 önbellek için 10.9 ns ve ana bellek için 170.4 ns olup, lmbench aracı kullanılarak ölçülmüştür. Ana işletim

sistemi Linux'tur (çekirdek sürümü 2.4.14, RedHat 7.3). Gcc sürüm 2.96 ile derlenen snort 1.9.0 (yapı 205) sürümü kullanılmıştır. Sonuçlar, tipik trafik kalıpları için yeni önerilen algoritmanın, NIDS performansını %10-%36 arasında artırdığını, bazı kural kümesi ve trafik kalıpları için desen eşleştirme performansının üç katına kadar iyileştirilebileceğini göstermiştir.

Desen eşleştirme algoritmaları birçok uygulamada önemli hale gelmiştir. Saldırı Tespit Sistemi (IDS) bunlardan biridir. IDS'lerde desen eşleştirme kötü niyetli faaliyetleri tespit etmek için modern ağ yapısının önemli bir parçası haline gelir. IDS'in algılama motoru, bu kötü niyetli etkinlikleri aramayı gerçekleştirmek için desen arama algoritmalarına kullanır, bu arama aşaması, IDS işlem süresinin yaklaşık %70'ini oluşturur. Günümüzde artan ağ trafikleri nedeniyle yeni kural politikalarını ihtiyaç duyulmakta ve bu nedenle arama süreleride artmaktadır. Daha hızlı çalışabilen bir yöntem geliştirerek IDS işlem süresini azaltmaya ihtiyaç vardır. Hasan ve diğerleri (2012), desen karşılaştırma süresini azaltmak için HBMH algoritması ile Boyer-Moore Horspool algoritmasını geliştirmeyi hedeflemiştir. BMH, Boyer-Moore algoritmasının biraz değiştirilmiş bir sürümüdür. Boyer-Moore algoritmasının aksine, BMH algoritması yalnızca bir tablo kullanır buda kötü karakter kayması tablosudur. Boyer-Moore algoritması ise iki tablo kullanır bunlar: kötü karakter kayması ve iyi son ek kayması tablolarıdır. Bu çalışmada anlatılan yöntem, BMH algoritmasının kaydırma tablosunun yanına hash fonksiyonunu ekleyerek BMH algoritmasını geliştirmeye çalışmaktadır. Hash fonksiyonu verilen metnin matematiksel işlemlerle sabit bir boyutta ifade edilmesidir. Hash işlevini kullanmanın ana yararı, her denemede BMH algoritması tarafından gerçekleştirilen karakter karşılaştırmalarının sayısını azaltmaktır. Böylelikle gerekli karşılaştırma süresini azaltır. Çalışmada elde edilen sonuçlara göre HBMH algoritmasının performansının BMH algoritmasına göre daha başarılı olduğu tespit edilmiştir. Bunun nedeni, hash fonksiyonunun BMH'deki 17 karşılaştırmadan karakter karşılaştırma sayısını HBMH'de yalnızca 8 karşılaştırmaya indirgemiş olması olarak ifade edilmiştir. Sonuç olarak, HBMH algoritmasının hızlı olduğu ve ağ güvenliğinde yararlı olabileceği önerilmiştir.

Saldırı tespit sistemleri, bilgisayar ve ağ sistemleri için koruyucu önlemlerin hayati unsurları olarak ele alınır. Ağ hızı ve algılama iş yüklerindeki büyük artış, yüksek verimli

NIDS araçlarına ihtiyacı arttırmaktadır. NIDS araçları trafikte geçen paketlerde önceden tanımlanmış kurallardaki bilinen saldırı modelini kontrol etmesi gerektiğinden, desen arama işlevi imza tabanlı NIDS'lerin en önemli parçası haline gelir. Anithakumari ve Chithraprasad (2009), ağ trafiğinin segmentasyonunu desteklemek ve parçalı saldırıları tespit etmek için, hem kısmi hem de tam desen eşleme yapan CDAWG (Compact Direct Acyclic Word Graph) isimli bir yöntem önermiştir. Bu çalışmada, CDAWG yapısını kullanarak verimli bir desen eşleştirme algoritması tasarlanmış ve uygulanmıştır. Bu yeni algoritmaya Dawgsearch ismi verilmiştir. Çalışmada yeni desen eşleştirme modülünün performansını değerlendirmek için kapsamlı bir dizi deney gerçekleştirilmiştir. Tüm deneyler, 512 MB RAM'e sahip ve Red Hat 8.0 linux işletim sistemi çalıştıran Pentium-IV 2.40GHz bir bilgisayarda gerçekleştirilmiştir. Önerilen algoritmanın Snort üzerindeki performansını incelemek için, saniye cinsinden süre miktarını hesaplamaya yarayan linux time komutu kullanılmış. Analizler sonucunda, önerilen algoritmanın Aho-Corasick algoritmasına göre 2,5 kat daha hızlı olduğu görülmüştür.

Suhendra (2016), Boyer-Moore ve Aho-Corasick algoritmalarının ağ saldırı tespit sistemindeki etkinliğini araştırmaktadır. Çalışmada C# programlama dili ile kodlanan algoritmalarda girdi boyutu arttırılarak AC ve BM algoritmaları ile arama yapılarak çalışma zamanları karşılaştırılmıştır. Sonuçlara göre Boyer-Moore algoritmasının ağ saldırı tespit sisteminde uygulanacak kadar hızlı olduğu fakat Aho-Corasick algoritmasının Boyer-Moore algoritmasından daha hızlı olduğu ifade edilmiştir.

Norton (2004), Aho-Corasick algoritmasının optimize edilmiş bir versiyonunu önermiştir. Optimize edilmiş tasarım, performansı önemli ölçüde artıran Aho-Corasick durum tablosunun optimize edilmiş vektör uygulamasını kullanır. Belleği verimli kullanan bir değişken, bellek gereksinimlerini azaltmak ve büyük kalıp gruplarında performansı daha da artırmak için seyrek matris depolama kullanır. Saldırı Tespit Sistemleri, çok yüksek ağ hızlarında ve zararlı ortamlarda gerçek zamanlı desen eşleştirme yetenekleri gerektiren çok özel uygulamalardır. Desen eşleştirme ve Saldırı Tespitinde dikkate alınması gereken önemli sorunların birçoğu, Snort Saldırı Tespit Sisteminde uygulandığı şekliyle Aho-Corasick algoritmasının kullanımına yönelik bir çerçeve oluşturmak için tartışılmıştır. Yazarların Aho-Corasick algoritmasının orijinal,

optimize edilmiş ve seyrek depolama sürümlerini karşılaştıran performans sonuçları sunulmaktadır. Testler, çeşitli sözlük testleri ve Snort tabanlı Saldırı Tespiti performans testi kullanılarak yapılmıştır. Desen grubu boyutlarının ve derleyici seçiminin performans üzerindeki etkisi de birkaç popüler derleyici kullanılarak gösterilmiştir. Optimize edilmiş Aho-Corasick, yazarın 2002'de Snort'ta yayınlanan orijinal versiyonunun geliştirilmiş halidir. Durum tablosu farklı şekilde yönetilmiş ve arama rutininin daha optimal bir talimat karışımına derlenmesi sağlanmıştır. Önceki durum tablosundaki her giriş, tek bir yapıda bulunan durum için bir geçiş vektörüne, tablonun NFA sürümü için bir hata göstericisine ve durum için bir eşleşen desen listesine sahiptir. Önceki durum tablosu artık bir durum geçiş tablosuna, durum başına eşleştirme desen listelerinden oluşan bir diziye ve NFA için her durum için ayrı bir hata gösterici listesine bölünmüştür. Test edilecek Aho-Corasick rutinlerinin her birinin birkaç farklı derlenmiş versiyonunu içeren bir test ortamı geliştirilmiştir. Derleyici olarak Microsoft VC 6.0, Intel C 6.0 ve Cygwin gcc 3.3.1 kullanılmıştır. Üç farklı derleyicinin kullanılması, arama algoritmasının ve depolama yöntemlerinin performansı ve önbelleğe alma davranışına ilişkin daha geniş bilgi sağlamıştır. Tüm sonuçlar Aho-Corasick algoritmasının DFA sürümü içindir. Bu testler, 1 GB bayt RAM içeren Dell 8100 1.7 GHz sistemde 16 bit durum değerleri kullanılarak gerçekleştirilmiştir. Optimize edilmiş algoritma, standart Aho-Corasick ve Wu-Manber algoritmasından önemli ölçüde daha hızlıdır. Snort'taki web kuralları çok sayıda küçük iki baytlık desen içerir. Bu, Wu-Manber algoritmasının kötü bir karakter kaydırma stratejisinden yararlanmasını engeller. Snort üzerinde yapılan testler, optimize edilmiş versiyonun yüzde 31 daha hızlı olduğunu, yüzde 46 daha fazla desen arama kapasitesi sağladığını ve algoritmanın orijinal versiyonunun belleğinin yarısını kullandığını göstermiştir.

2.2. Katman 7 Filtreleme

Açık kaynak kodlu bir yazılım olan snort linux ve freebsd çekirdeği tabanlı işletim sistemlerinde katman 7 filtreleme için kullanılmaktadır. İnan Aydın (2010), katman 7 filtreleme için snort yazılımını kullanmıştır. Snort yazılımda arama algoritması olarak Aho-Corasick algoritmasını kullanmıştır. Snort kullanıcı uzayında arama yapan bir

yazılımdır. Ayrıca çalışmada Aho-Corasick algoritmasının ram tüketiminin fazla olduğundan bahsedilmiştir.

Linux çekirdeğinde uygulama katmanında filtreleme vardır fakat performans olarak iyi değildir. Tahir Bilal (2011), Aho-Corasick algoritmasını Linux çekirdeğine uygulayarak uygulama katmanındaki filtreleme performansını arttırmıştır. Ek olarak çalışmadaki testler çekirdek uzayındaki filtrelemenin kullanıcı uzayındaki filtrelemeden daha verimli olduğunu göstermiştir.

Freebsd işletim sistemlerinde güvenlik duvarı olarak pf ve ipfw varsayılan olarak bulunmaktadır. Dragos ve Avram (2010), çalışmasında katman 7'de paket filtrelemek için freebsd işletim sisteminin pf güvenlik duvarının pfil_hook mekanizmasını kullanmıştır. Pfill_hook yöntemi ile gelen ve giden ağ paketleri ethernet kartından alınarak, incelenip engellenebilmekte ya da izin verilmektedir. Bu çalışmada sadece pfill_hook yöntemi aktif olmadan ve aktif edilerek 1000 paket ile gecikme ölçülmesi yapılmıştır. Bu çalışma daha çok kavramsal olarak ele alınmış ve sadece torrent trafiğinin tespit edildiği ifade edilmiştir. Paket içerisinde nasıl arama ve filtreleme yapıldığından bahsedilmemiştir.

Pfsense endüstride birçok firma tarafından kullanılan freebsd işletim sistemi tabanlı açık kaynak kodlu ve ücretsiz bir güvenlik duvarıdır. Patel ve Priyanka (2017), pfsense güvenlik duvarını incelenmiştir. Pfsense üzerinde katman 7 filtreleme olduğunu belirtmişler. Eski sürümlerdeki pfsense güvenlik duvarında ipfw-classifyd diye bir teknik kullanılmıştır. Bu teknik şu an yeni versiyonlarda çalışmıyor. Ayrıca tam işlevsel olmadığından ve kaynak tüketiminin fazla olduğundan bahsedilmiştir.

2.3. YTA ağlarında katman 7 filtreleme

YTA ağlarında güvenliği arttırmak için katman 7 mimarisinde filtreleme yapılabilir. Chin ve diğerleri (2018), katman 7 filtrelemesi yapmak için bir çekirdek modülü geliştirmiştir. Bu çekirdek modülü anahtar (switch) cihazı üzerine kurulmuştur. Yani YTA ağındaki her anahtar cihazı üzerine çekirdek modülü kurulumu gerekmektedir. Fakat kurumlardaki tüm anahtar donanımlarına ve yazılımlarına bunun kurulması gerçek dünyada çok

mümkün değildir. Anahtarların işletim sistemleri çekirdek modülünün geliştirildiği işletim sisteminden farklı olabilir ya da kapalı sistem olup kuruluma izin vermeyebilir. YTA için üretilen çözümün daha uygulanabilir olması gerekmektedir.

YTA ağlarında katman 7 filtreleme yapmak için gerçek zamanlı trafiğe müdahale edilmesine gerek yoktur. Nam ve Kim (2018), trafiğin bir kopyası anahtar cihazı üzerinden port mirroring ile alınmıştır. Port mirroring ile anahtar cihazı üzerindeki tüm trafik tek bir fiziksel porttan başka bir cihaza aktarılabilir. Port mirroring ile alınan trafik kopyası açık kaynak kodlu bir yazılım olan suricata ile incelenmiş, engellenmek istenen trafik ise anahtar gibi network aygıtlarına paket gönderilerek engellenmiştir.

3. MATERYAL ve YÖNTEM

3.1. Freebsd Üzerinde Çekirdek Seviyesinde Paket Filtreleme

Bir güvenlik duvarında ağdaki paketleri filtreleyebilmek için ağda gelen ve giden tüm veri paketlerinin hedefe gönderilmeden önce kontrolü yapılmalıdır. Bu kontrol kullanıcı ya da çekirdek uzayında çalışan uygulamalar ile yapılabilir.

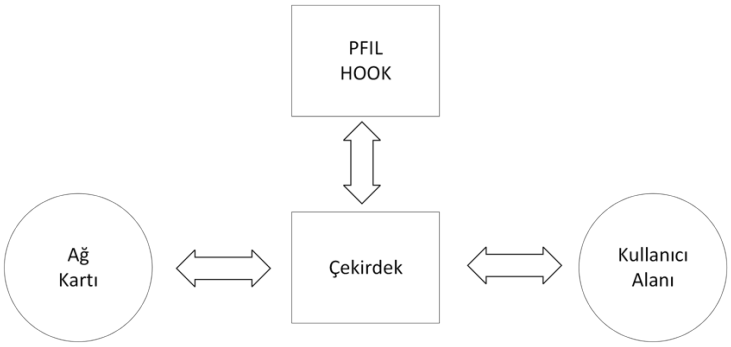
Çekirdek bilgisayarlarda donanım ve kullanıcı alanı arasında yer alarak, donanım aygıtları ve kullanıcı uzayındaki yazılımlar arasındaki bağlantıyı yönetir. Ayrıca İşletim sistemi ile haberleşerek ram bellek kullanımı, işlemci kullanımı, giriş çıkış (G/Ç) işlemleri gibi bilgisayarlardaki temel işlemlerin yürütülmesi ve kaynak paylaştırılması konusunda görevlidir. (Filozof, 2008). Yani, bir veri paketi ağ kartına gelmeden ya da ağ kartından kullanıcı alanına geçişi sırasında yönlendirme çekirdek uzayında yapılır. Şekil 3.1'de bir veri paketinin yolculuğu basit bir şekilde tasvir edilmiştir. Bu çekirdek alanında veri paketi manipüle edilip çekirdek üzerinde geçişi kontrol edilebilir.

Şekil 3.1. Bir veri paketinin yolculuğu

3.1.1. PFIL Kanca Atma (Hooks)

İşletim sistemlerinde paket filtreleme yapabilmek için bazı yöntemler geliştirilmiştir. Freebsd işletim sisteminde paket filtreleme yapmak için çekirdek üzerindeki fonksiyonlar kullanılabilmektedir. Bu sayede, paketi incelemek, sonrasında gerekli ise paketi düşürme veya geçişine izin verme işlemleri yapılabilmektedir. BSD çalışanları verimli ve kullanışlı bir yol olarak paket filtrelemesi için paket filtre kancaları (PFIL kanca) fikrini ortaya koymuşlardır (Balaban, 2005). Bu çalışmada güçlü, kararlı ve güvenilir olan ve endüstride sıklıkla kullanılan FreeBSD işletim sistemi tercih edilerek çekirdekte paket filtreleme için PFIL kanca atma yöntemi kullanılmıştır. Literatürde PFIL Hook olarak adlandırıldığından bu şekilde bahsedilmeye devam edilecektir.

PFIL, freebsd bir sistemde giriş/çıkış akışı için gelen ve giden paketlerin belirli işler için çağrılmasına olanak sağlayan bir yapıdır. Kullanıcıya, çekirdek arabiriminde belli noktalara kendi işlemlerini yapabilmesi için olanak tanır. Bir güvenlik duvarı uygulamasında paketi manipüle etmek için kullanılabilir (Freebsd Foundation, 2019). Şekil 3.2'de bir paketin çekirdeğe ulaştıktan sonra kanca atılıp incelenmesi tasvir edilmiştir. Burada PFIL Hook araya girer ve güvenlik duvarına işlem yapma fırsatı sunar.



Şekil 3.2. PFIL Hook

FreeBSD bize paketlere kanca atmak üzere PFIL Hook için bazı fonksiyonlar sunmuştur. Bu fonksiyonlardan bazıları (Freebsd Foundation, 2019);

1. pfh_inet = pfil_head_get(): Bu fonksiyon, paket filtresinin beklediği anahtarı ve bağlantı türünü döndürür. Fonksiyon sonucunda dönülen değer kanca atarken kullanılır.
2. pfil_add_hook(filter, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet): Bu fonksiyon, kanca görevlerini kurmamıza yardımcı olur. Parametreleri arasında PFIL_IN, yalnız giren paketlere kanca atılmasını gösteren parametredir. Kanca atıldıktan sonra, kanca filter fonksiyona kurulur ve kontroller gerçekleştirilir. Fonksiyonun ikinci parametresi, her çağırıldığında kanca atma işlemine bir gösterici atar (Stewart ve Healy, 2010). Burada, bu işleme gerek duymadığımızdan ikinci parametreye NULL verdik.
3. Pfil_remove_hook(filter, NULL, PFIL_IN | PFIL_WAITOK, pfh_inet): Bu fonksiyon, giren paketler üzerinde ki kancayı kaldırır.

Bu kodlar kullanılarak oluşturulan hook işlemi yapan uygulama derlenerek, FreeBSD'nin içerisinde bir yapı olan dinamik çekirdek bağlama (KLD) ile çekirdeğe atılır. Bu işlem

çekirdeğin işleyişinde normal şartlar altında bir sorun oluşturmamaktadır. Yalnız, modül içerisindeki bir hata çekirdeğin doğru çalışmasını engeller. Bu sebeple işletim sistemi kendini baştan başlatır.

3.1.2. Kuralları Yükleme

Filtreleme daha önce de bahsedildiği gibi bazı kurallar çerçevesinde gerçekleşir. Bu kuralların basit bir yoldan çekirdek alanına gönderilmesi ne yazık ki mümkün olmamaktadır. Bilindiği üzere kullanıcı alanında birçok kodlama dilinde kod içerisine dosyalar dâhil edilebilmektedir. Örneğin, C dilinde yalnız fopen ile bir .txt dosyası kod içerisine alınıp çözümlenir. Çekirdek alanında ise bu işlem bir aygıt oluşturularak yapılır. Bu işlemde yine çekirdeğe bir modül yazılır ve bu modül üzerinden, kullanıcı alanına bir girdi işlevi gören bir aygıt oluşturur. Bu aygıtlardan biri de "Karakter Aygıtı" adı verilen, verileri doğrudan çekirdeğe aktaran bir aygıttır. FreeBSD sisteminde /dev aygıtı altında konumlanır. Çizelge 3.1'de bir karakter aygıtındaki özellikler "cdevsw" yapısı içerisinde gösterilmiştir (Freebsd Foundation, 2022).

Çizelge 3.1. cdevsw yapısı

```
/* Karakter cihazı giriş noktaları */
static struct cdevsw echo_cdevsw = {
        .d_version = D_VERSION,
        .d_open = echo_open,
        .d_close = echo_close,
        .d_read = echo_read,
        .d_write = echo_write,
        .d_name = "echo",
};
```

Karakter aygıtı open(), close(), read(), write() gibi fonksiyonları içerisinde saklar. Aygıtı çalıştırmak için echo_open() fonksiyonu çağırılması yeterli olup, dosya sistemi içerisinde /dev/echo içerisine yazılan veriler, modül içerisinde echo_write() fonksiyonunu tetikler. Tetiklenen fonksiyon verileri bir tampona yazar ve modül tarafından okunması sağlanır.

Oluşturulan bu aygıta yazılabilecek veri miktarını modül içerisinde belirlenir. Belirlenen miktardan fazla veri yazılmaya çalışılırsa, çekirdek hataya düşebilir. Bu sebeple belirlenen miktardan fazla verinin yazılmamasına dikkat edilmelidir. Ayrıca 300 karakter üstü veriler de bazı durumlarda sağlıklı bir şekilde alınamamaktadır. Böyle bir durumla karşılaşıldığında veriler küçük boyutlara düşürülüp tekrar yollanmalıdır.

3.1.3. Kural Formatı

Güvenlik duvarlarında filtreleme ayarları için kurallar kullanılır. Bu kuralların doğru yazılması ve kullanılması güvenlik duvarının optimize çalışması için önemlidir. Önerilen çalışmada uygulama filtrelemenin yanında optimizasyon için port ve protokole göre de arama eklenmiştir. Çizelge 3.2'de örnek kural tablosu gösterilmiştir. Bu tabloya göre bazı ön tanımlar alınmış ve bu tanımların kullanımıyla güvenlik duvarı optimize edilmeye çalışılmıştır.

Çizelge 3.2. Kural Tablosu

Port	Protokol	Model
80	6	facebook
443	17	malware
443	-1	bg-tek.net
-1	-1	virus

Örneğin ağa, internetin 443 (https) portundan, 6 (TCP) nolu protokol numarasıyla gelen paket içerisinde "bg-tek.net" modelinin arandığı bir kuralımız bulunmaktadır. Görüldüğü üzere yalnızca model aranabilir. Ama bu noktada port ve protokol gibi paketin başlığına göre çok büyük olan tüm veri paketi incelenmek zorunda kalınabilir. Bunu önlemek için ulaşılmak istenen servisin hangi port ve protokol ile çalıştığı bilindiği takdirde doğru sonuca ulaşmak daha az masraflı olmaktadır. Bu mantığa göre bir model aranacağında listelenmiş port ve protokol numaraları modele hızlıca ulaşmamızı sağlar. Çizelge 3.2'de da bazı hücrelerin "-1" ile doldurulduğu görülmektedir. Buna göre o değeri alan yerlerde port veya protokol bilgisi bilinmediğinden değeri "-1" alınmıştır. Yani tüm protokol ve portlarda arama yapılacaktır.

Güvenlik duvarında kural eklerken modeli Çizelge 3.2'de olduğu gibi detaylı bir şekilde sınırlandırmak model aramanın masraflarını düşürür. Bu noktada alınan bu bilgiler veri paketinin başlığından temin edilebildiği için daha küçük bir alan içerisinde aranır. Paket başlığı içerisinden ayrılabilen IP adresi gibi diğer ayırt edici verilerde kural olarak eklenerek ağımız içerisinde kişi bazlı da haberleşmenin filtrelemesi sağlanabilir.

3.2. Desen Arama Algoritmaları

Güvenlik duvarı uygulamalarında, aranan desenin en hızlı bir şekilde bulunması oldukça önemlidir. Çok yüksek bant genişliği olan büyük işletmeler, oteller, açık alan internet hizmetleri gibi yerlerde çok fazla veri trafiği olmaktadır. Her bir veri paketi kontrol edildiğinden verinin uygun olup olmamasının kontrolü çok hızlı bir şekilde yapılmalıdır. Bu noktada desen arama algoritmalarının önemi ortaya çıkmaktadır. Desen arama ya da dizi arama algoritmaları, iki farklı dizinin birindeki yapının bir diğer dizi de aranması işlemidir (GeeksforGeeks, 2022).

Çizelge 3.3. Çalışılan Algoritmalar

	Algoritmalar												
Çalışmalar	**AC**	**BM**	**BMH**	**CDAWG**	**QWM**	**WM**	**CW**	**HBMH**	**Nw2**	**SWBM**	**AC_BM**	**FvH**	**E2xB**
(Yang Dong vd., 2006.)	X				X	X	X		X				
(Fisk ve Varghese, 2001)	X	X	X							X			
(Vidanagamachchi vd., 2012)	X						X						
(Coit vd., 2001)		X									X		
(Anagnostakis vd., 2003)		X										X	X
(Hasan vd., 2012)			X					X					
(Anithakumari ve Chithraprasad, 2009)	X			X									
(Suhendra, 2016)	X	X											
(Norton, 2004)	X					X							
(Antonatos vd., 2003.)	X					X							

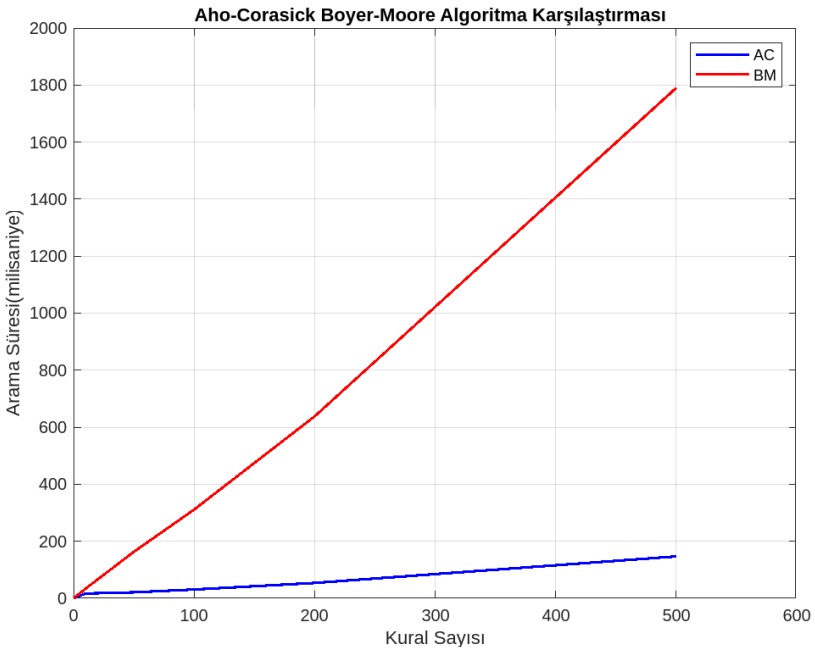
Bu çalışma için çeşitli algoritmalar incelenmiştir. İncelenen algoritmalar ve hangi makalelerde bulundukları Çizelge 3.3'te gösterilmiştir. Bu algoritmalar hem performans ve kaynak tüketimi hem de çekirdek seviyesinde kullanımı açısından değerlendirilmiştir.

Genel olarak saldırı tespit ve önleme sistemlerinde (IDS/IPS) kural sayısının artmasıyla veriminin daha yavaş azalmasından dolayı Aho-Corasick ve algoritmaları kıyaslamak için kullanılan Boyer-Moore algoritmalarının desen arama için sıklıkla kullanıldığı gözlenmiştir. Bu çalışmada IDS/IPS sistemlerde desen arama için yoğun olarak kullanılan bu iki algoritma yapılan simülasyon ile karşılaştırılmıştır.

4. BULGULAR

4.1. Desen Arama Algoritması Karşılaştırma

Bu bölümde seçilen Aho-corasick ve Boyer-Moore algoritmaları çalışma süresi cinsinden karşılaştırılmıştır. Tüm deneyler, 2 GB RAM'e sahip ve linux 5.4.0-kali3-686-pae işletim sistemi çalıştıran 4 çekirdekli Intel(R) Core(TM) i7-4770HQ CPU @ 2.20GHz işlemciye sahip bir sanal makinede gerçekleştirilmiştir. Kullanılan açık kaynak kodlu IDS/IPS olan snort versiyonu 2.9.16'dır. Snort ve algoritma kodları gcc 9.2.1 ile derlenmiştir. Programlama dili olarak C++ kullanılmıştır. Algoritmaların çalışma süresini ölçmek için linux sistemlerdeki time komutu kullanılmıştır.



Şekil 4.1. Aho Corasick ve Boyer-Moore algoritmalarının karşılaştırılması

Testler ağda bant genişliği sabit hızda tutularak aranan kural sayısı arttırılarak gerçekleştirilmiştir. Şekil 4.1'de Aho Corasick ve Boyer-Moore algoritmalarının karşılaştırılması verilmiştir. Elde edilen sonuçlara göre, çalışma süresi olarak Aho-Corasick algoritma performansının daha iyi olduğu görülmektedir. Kural sayısı arttıkça Aho-Corasick algoritmasının çalışma süresi daha yavaş artmaktadır.

Elde edilen sonuçlar literatürdeki farklı çalışmalar ile tutarlı olduğu görülmektedir. Suhendra (2016), çalışmasında Boyer-Moore algoritmasının IDS/IPS'ler için kabul edilebilir düzeyde bir hıza sahip olduğunu, fakat Aho-Corasick algoritmasının Boyer-Moore algoritmasından daha hızlı sonuç verdiğini ifade etmiştir. Bir diğer çalışma ise Fisk ve Varghese (2001), uygulanan kural sayısı arttıkça Aho-Corasick algoritmasının çalışma süresinin artış hızının daha yavaş, Boyer-Moore algoritmasının çalışma süresinin artış hızının ise daha hızlı olduğunu göstermiştir. Bu tez çalışmasında da benzer sonuçlar elde edilmiştir. Bu bağlamda yapılan analizlerin ve algoritma seçiminin tutarlı olduğu görülmektedir.

4.2. Suricata Yazılımı ile Önerilen Çalışmanın Karşılaştırılması

Testler aewin SCB-7906 donanımıyla yapılmıştır. Donanımın işlemcisi 4 çekirdek Intel(R) Celeron(R) CPU J1900 @ 1.99GHz'dır. Donanımda 8 gb ram bulunmaktadır. Cihaz üzerinde 6 tane 1Gbit Ethernet portu bulunmaktadır. Testler Freebsd işletim sistemi üzerinde yapılmıştır. İşletim sistemi versiyonu 11.4-STABLE amd64'tür. Suricata Pfsense 2.4.2 versiyonu üzerinde yapılandırılmıştır. Testlerde kullanılan suricata versiyonu 5.0.4'tür. Suricata uygulamasında paketlerin engellenebilmesi için suricata netmap ile inline modda çalıştırılmıştır.

Testlerde donanımın 2 portuna 2 farklı bilgisayar bağlanmıştır. Bilgisayarlar Ethernet portlarına cat5 kablo ile bağlanmıştır. İperf aracı ile bant genişliği testleri yapılmıştır. Bir bilgisayar sunucu bir bilgisayar istemci olarak kullanılmıştır (Dell Technologies, 2021). İstemci olarak kullanılan bilgisayarın bağlı olduğu arabirimde suricata ve önerilen çalışma kullanılarak filtrelemeler yapılmıştır.

Suricata ve önerilen çalışma aktif değilken yapılan testlerde 1mb tcp pencere boyutuyla 943mbit bant genişliği ölçülmüştür.

0.0- 5.0 sec 561 MBytes 943 Mbits/sec

Saniyede 943 megabit, zorunlu paketler arası boşluklar ve diğer protokol ek yükleri nedeniyle standart 1500 bayt yükleri kullanarak 1000BASE-T üzerinden IPv4 üzerinden

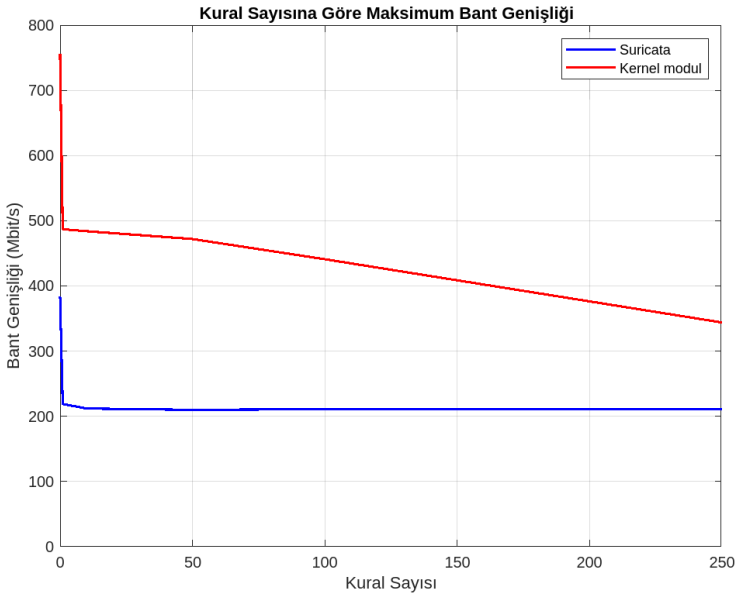
TCP ile elde edebileceğiniz teorik maksimum aktarım miktarıdır (IEEE Standard for Ethernet, 2018).

Sunucu tarafındaki bilgisayarda bir web servisi yapılandırılmış içerisine malware ifadesi geçen bir dosya konulmuştur. Suricata servisinde bu dosyayı engellemek için şu kural yazılmıştır: drop tcp any any -> any any (msg:"malware blocked"; content:"|6D 61 6C 77 61 72 65 2E 74 78 74|"; sid:1;)

Bizim çalışmamızda ise aynı engellemeyi yapan kural: echo "-1 -1 6 0 x6dx61x6cx77x61x72x65x2ex74x78x74" >> /dev/echo

Her iki uygulama için aynı şekilde rastgele 249 kural oluşturulmuştur. Testlerde son kurala yukarıdaki arama sonucunda bulunacak kural eklenmiştir. Sonuç karşılaştırmasının daha tutarlı olması için her iki uygulamada aho-corasick algoritması ile paketler içerisinde kural araması yapacak şekilde ayarlanmıştır.

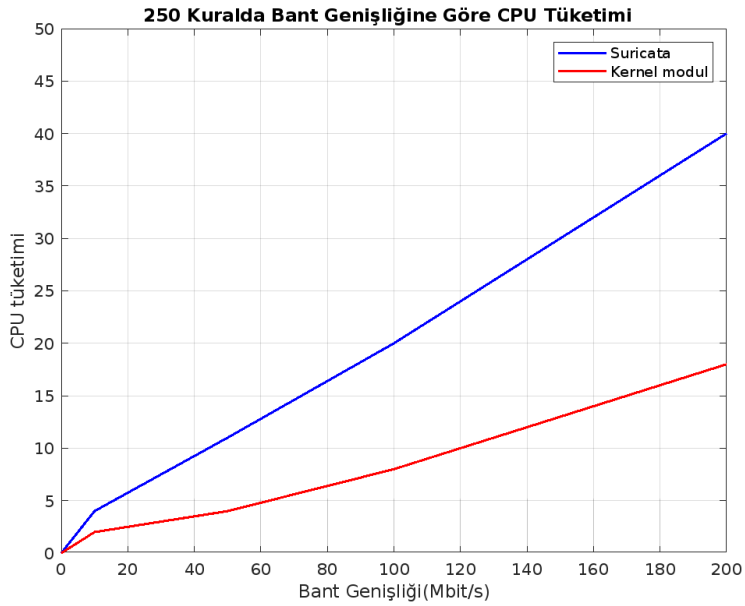
Filtrelemede kural sayısı arttırılarak donanım üzerinden geçen maksimum bant genişliği testi Şekil 4.2'de gösterilmiştir.



Şekil 4.2. Kural sayısına göre bant genişliği kullanımı

Bu testte iki uygulamada da kural sayısı arttırılarak (0,1,10,50,100,250 kural) cihaz üzerinden geçen maksimum bant genişliği ölçülmüştür. Şekil 4.2'de görüldüğü gibi kullanılan tüm kural sayılarında önerilen çalışma suricata uygulamasından daha fazla paket filtrelemiştir. Buda filtrelemenin çekirdek seviyesinde yapıldığında daha hızlı yapılabildiğini kanıtlamaktadır. Bu tez çalışmasında 100 kuraldan sonra bant genişliği miktarında suricataya göre daha hızlı bir azalma eğilimi gözükmektedir. Buda aho-corasick algoritmasının uygulanmasının suricata üzerinde daha başarılı yapıldığını göstermektedir.

250 kuralda bant genişliğine göre işlemci kullanımı Şekil 4.3'te görülmektedir.

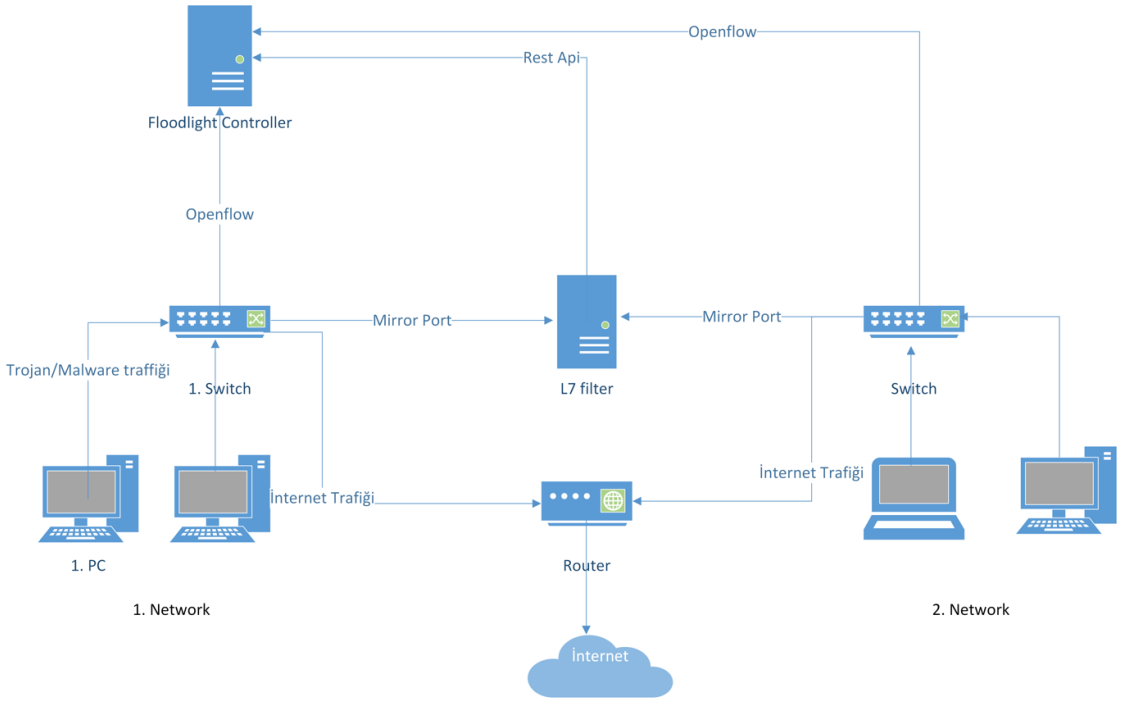


Şekil 4.3. Sabit kural sayısında bant genişliği arttırılarak CPU tüketimi

Bu testte her iki uygulamada da 250 kural ile filtreleme yapılarak 10,50,100 ve 200 Mbit bant genişliğinde cihaz üzerindeki işlemci kullanımları ölçülmüştür. (Suricata 250 kuralda 210Mbit geçirebildiği için maksimum 200Mbit ile ölçüm yapılmıştır.) Şekil 4.3'te görüldüğü gibi ölçülen her bant genişliğinde bizim çalışmamızın daha az işlemci kullandığı görülmüştür. Buda çekirdek seviyesindeki filtrelemelerin daha az işlemci kullandığını kanıtlamaktadır.

4.3. Önerilen Çalışmanın YTA mimarisine uygulanması

Bu çalışmada YTA mimarisinde ağ aygıtlarından mirror port ile trafik kopyası alarak, veri paketlerinin uygulama katmanında Aho-Corasick algoritması ile arama yapan ve YTA kontrolcü ile REST api aracılığı ile haberleşerek ağdaki tüm noktalarda tespit edilen ipleri engelleyen bir çekirdek modülü geliştirilmiştir. Örnek mimari Şekil 4.4'te görülmektedir.



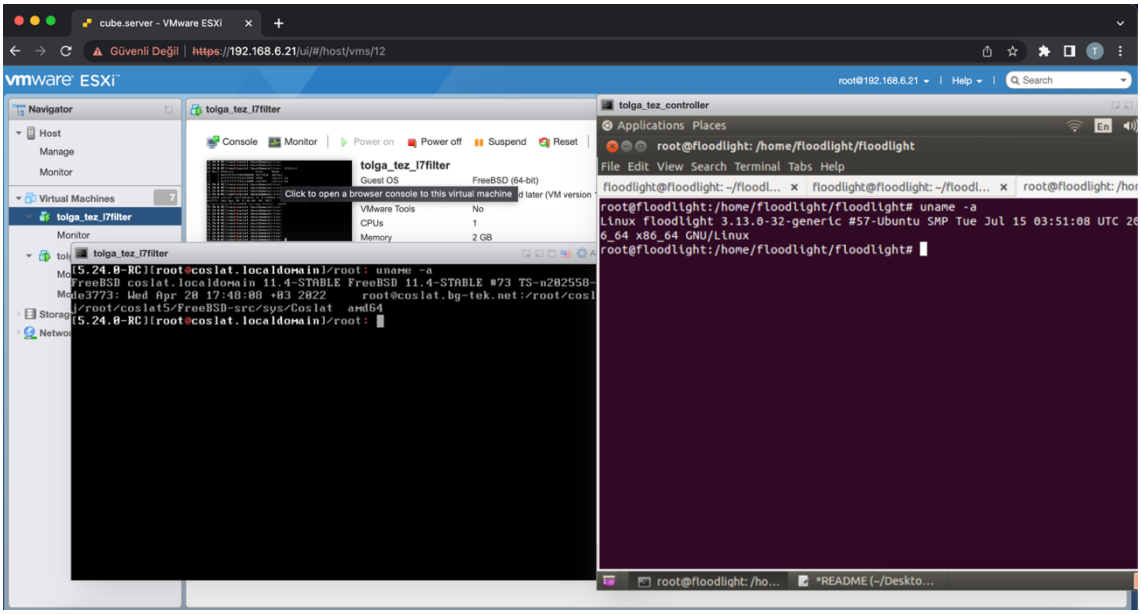
Şekil 4.4. Katman 7 filtrelemenin YTA mimarisine uygulanması

YTA denetleyicisi olarak mininet ile kolay entegre olduğundan ve Rest api ile yönetilmesi nedeniyle floodlight seçilmiştir. Senaryoya göre bir ya da daha fazla ağ anahtar cihazları üzerinden yönetilmektedir. Anahtarlar YTA mimarisinde floodlight kontrolcü ile openflow protokolü ile haberleşmektedir. Ayrıca üzerlerindeki trafiğin bir kopyasını mirror port ile önerilen çalışmaya göndermektedir. Önerilen çalışmada floodlight kontrolcü ile REST API üzerinden haberleşmektedir.

Ağdaki 1. bilgisayar üzerinden virüslü bir trafik akmaktadır. Aynı virüs firmadaki diğer bilgisayarlara bulaşarak yayılabilir ve kurumdaki tüm cihazlar ele geçirilebilir. 1. Anahtar tüm internet trafiğini mirror port ile freebsd işletim sistemi kurulu cihaza gönderir. Önerilen çalışma gelen zararlı trafiği freebsd işletim sisteminin PF mekanizmasının hook

yöntemini kullanarak çekirdek seviyesinde üzerine alır. Önceden tanımlanmış kurallara göre veri paketlerinde Aho-corasick algoritmasını kullanarak arama işlemi yapar. Zararlı trafiği tespit ettiğinde 1. bilgisayarın gitmeye çalıştığı hedef ip'yi REST API aracılığıyla floodlight kontrolcü üzerinden engeller. Floodlight kontrolcü ağdaki tüm cihazlara bu hedef ip'ye engel kuralı gönderir. Böylelikle ağdaki tüm istemcilerde bu zararlı yazılım bulunduran sunucuya erişim engellenmiş olur.

Uygulama sanal ortamda hazırlanmıştır. Sanal sistemler Şekil 4.5'te görülmektedir. Sanallaştırma için vmware ESXi platformu kullanılmıştır. Uygulama için 2 adet sistem kurulmuştur. İlk sistem uygulama katmanında filtreleme çekirdek modülünün çalıştığı freebsd işletim sistemidir. İkinci sistem ise floodlight kontrolcü, anahtar ve istemcilerin çalıştığı ubuntudur.



Şekil 4.5. Sanallaştırma ortamı

Freebsd işletim sistemine sahip sanal cihazda 2 adet arabirim bulunmaktadır. Em0 arabirimi yönetim için kullanılmaktadır. Vmx0 arabirimi ise anahtarlardan mirror port üzerinden trafiğin kopyasını almak için kullanılmaktadır. Freebsd işletim sisteminde vmx0 arabiriminin kendine ait olmayan trafiği alabilmesi için "ifconfig vmx0 promisc" komutu ile arabirim promiscuous moda geçirilmiştir.

Şekil 4.6'da görüldüğü gibi Freebsd platformuna özel olarak derlenen uygulama filtreleme modülü çekirdekte aktif hale getirilmiştir.

```
[5.24.0-RC][root@coslat.localdomain]/root: kldstat
Id Refs Address            Size     Name
 1    3 0xffffffff80200000 36f7950  kernel
 2    1 0xffffffff83a19000 10b0     cpuctl.ko
[5.24.0-RC][root@coslat.localdomain]/root: kldload ./modul.ko
[5.24.0-RC][root@coslat.localdomain]/root: kldstat
Id Refs Address            Size     Name
 1    5 0xffffffff80200000 36f7950  kernel
 2    1 0xffffffff83a19000 10b0     cpuctl.ko
 3    1 0xffffffff83a1b000 a2d344   modul.ko
[5.24.0-RC][root@coslat.localdomain]/root:
```

Şekil 4.6. Çekirdek modülünün yüklenmesi

Şekil 4.7'de görüldüğü gibi kurallar freebsd işletim sisteminin echo aygıtından gönderilmektedir. Kural diziminde ilk alan, kaynak port ikinci alan hedef port, 3. Alan protokol, 4. Alan grup, 5. Alan ise filtrelenecek hex ifadedir. 3 adet kural girilmiştir.

1. Kuralda tüm tcp portlarında malware.txt ifadesi aranmıştır.
2. Kuralda tüm tcp portlarında facebook ifadesi aranmıştır.
3. Kuralda tüm tcp portlarında coslat.com ifadesi aranmıştır.

Kurallardan sonra modülün freebsd işletim sisteminden üzerine alacağı trafik için arabirim ayarlanmıştır. Vmx0 üzerinde arama yapılacaktır. Kurallarda belirtilen 0. Grup için subnet tanımlaması yapılmıştır. Son olarak readyCoslat ifadesi ile modül üzerinde filtreleme başlatılmıştır.

```
[5.24.0-RC][root@coslat.localdomain]/root: echo "-1 -1 6 0 x6dx61x6cx77x61x72x65x2ex74x78x74" >> /dev/echo
[5.24.0-RC][root@coslat.localdomain]/root: echo "-1 -1 6 0 x66x61x63x65x62x6fx6fx6b" >> /dev/echo
[5.24.0-RC][root@coslat.localdomain]/root: echo "-1 -1 6 0 x63x6fx73x6cx61x74x2ex63x6fx6d" >> /dev/echo
[5.24.0-RC][root@coslat.localdomain]/root: echo "interface vmx0" >> /dev/echo
[5.24.0-RC][root@coslat.localdomain]/root: echo "class 0 0 192.168.101.0/24" >> /dev/echo
[5.24.0-RC][root@coslat.localdomain]/root: echo "readyCoslat" >> /dev/echo
[5.24.0-RC][root@coslat.localdomain]/root:
```

Şekil 4.7. Çekirdek modülüne kuralların yüklenmesi

Kurallarla ilgili yüklenme durumları ve kurallar sonucu tespit edilip engellene ip'ler sistem loglarına yazılmaktadır.

Ubuntu işletim sistemine sahip sanal cihazda 2 arabirim bulunmaktadır. Eth0 arabirimi anahtar üzerindeki sanal istemcilerin internete çıkışı için kullanılmaktadır. Aynı zamanda bu arabirimdeki trafik uygulama filtreleme modülünün bulunduğu sistemdeki vmx0 arabirimine gönderilmektedir. Eth1 arabirimi ise yönetim için kullanılmaktadır.

YTA kontrolcü olarak Floodlight kullanılmıştır. Floodlight kontrolcü ağ aygıtları ile 6653 portundan openflow protokolü ile haberleşmektedir.

Anahtarlar ve istemcileri simule etmek için mininet uygulaması kullanılmıştır. Mininet uygulamasındaki anahtarlar openflow protokolü ile 6653 portundan floodlight YTA kontrolcü ile haberleşmektedir. Şekil 4.8'de görüldüğü gibi örnek uygulamada mininet uygulamasında 4 anahtar ve bunların arkasında 4 istemci oluşturulmuştur.

```
floodlight@floodlight:~/floodlight$ sudo mn --nat --topo linear,4 --controller=remote,ip
=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13
[sudo] password for floodlight:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Şekil 4.8. Ağ yapılandırması

Uygulama filtreleme modülünün kurulu olduğu sistemden floodlight kontrolcülerin kurulu olduğu sisteme 8080 portundan rest api aracılığıyla erişilebilmektedir. Rest api ile YTA kontrolcü üzerine bağlı olan mininet ağındaki anahtarların listesi şekil 4.9'da gösterilmiştir.

```
[5.24.0-RC][root@coslat.localdomain]/root: curl http://192.168.101.169:8080/wm/core/controller/switches/json
[{"inetAddress":"/127.0.0.1:41414","connectedSince":1667831749377,"switchDPID":"00:00:00:00:00:00:00:02"},{"inetAddress":"/127.0.0.1:41416","connectedSince":
1667831750658,"switchDPID":"00:00:00:00:00:00:00:03"},{"inetAddress":"/127.0.0.1:41412","connectedSince":1667831749369,"switchDPID":"00:00:00:00:00:00:00:01"
},{"inetAddress":"/127.0.0.1:41411","connectedSince":1667831749377,"switchDPID":"00:00:00:00:00:00:00:04"}][5.24.0-RC][root@coslat.localdomain]/root:
```

Şekil 4.9. Aktif anahtarların listesi

Başlangıç durumunda YTA kontrolcü üzerindeki erişim listesi Rest api aracılığı ile listelendiğinde boş olduğu şekil 4.10'da görülmektedir.

```
[5.24.0-RC][root@coslat.localdomain]/root: curl http://192.168.101.169:8080/wm/acl/rules/json
[][5.24.0-RC][root@coslat.localdomain]/root:
```

Şekil 4.10. Filtreleme öncesi acl listesi

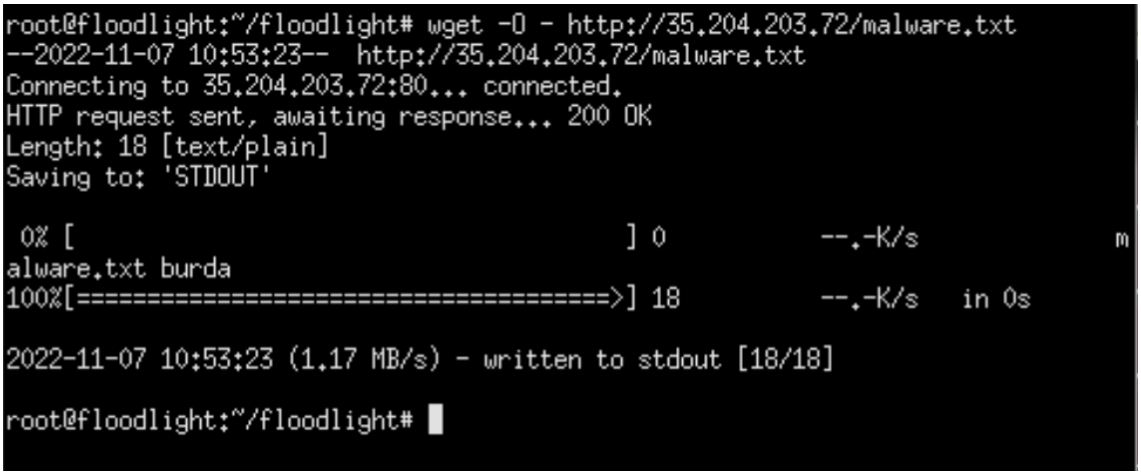
Şekil 4.11'de 1.anahtar üzerindeki flow tablosuna bakıldığında herhangi bir engelleme olmadığı görülmektedir.

```
mininet> dpctl dump-flows -O Openflow13
*** s1 ------------------------------------------------------------------------
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=28.945s, table=0, n_packets=83, n_bytes=13234, priority=0 actions=
CONTROLLER:65535
 cookie=0x0, duration=28.945s, table=1, n_packets=0, n_bytes=0, priority=0 actions=CONTR
OLLER:65535
 cookie=0x0, duration=28.945s, table=2, n_packets=0, n_bytes=0, priority=0 actions=CONTR
OLLER:65535
 cookie=0x0, duration=28.945s, table=3, n_packets=0, n_bytes=0, priority=0 actions=CONTR
OLLER:65535
 cookie=0x0, duration=28.945s, table=4, n_packets=0, n_bytes=0, priority=0 actions=CONTR
OLLER:65535
```

Şekil 4.11. Engelleme öncesi 1. anahtar üzerindeki flow tablosu

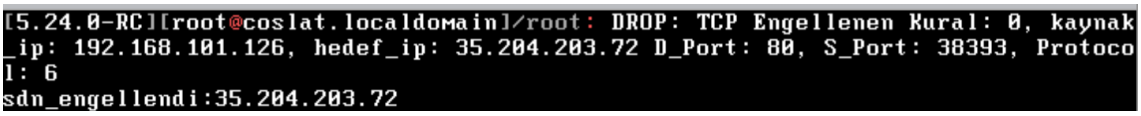
1.Anahtardaki ağda bulunan h1 istemcisinden web üzerindeki bir adrese istek atıldığında başarılı bir cevap dönmektedir. Bu istek uygulama filtreleme tarafındaki kurallarda tanımlı olmadığından bir engellemeye sebep olmamaktadır.

Şekil 4.12'de görüldüğü gibi 1. anahtardaki ağda bulunan h1 istemcisinden web üzerindeki bir adreste bulunan malware dosyasına istekte bulunulduğunda cevap dönmektedir. Çünkü filtreleme ağ geçidi modunda yapılmadığından anlık paket engellemesi yapmak mümkün değildir.



Şekil 4.12. H1 istemcisinden malware dosyasına başarılı istek

Fakat şekil 4.13'te görüldüğü gibi mirror porttan alınan trafikte malware dosyasına yapılan istek uygulama engelleme modülü tarafından yakalanmıştır.



Şekil 4.13. Uygulama engelleme tarafından tespit edilen malware isteği

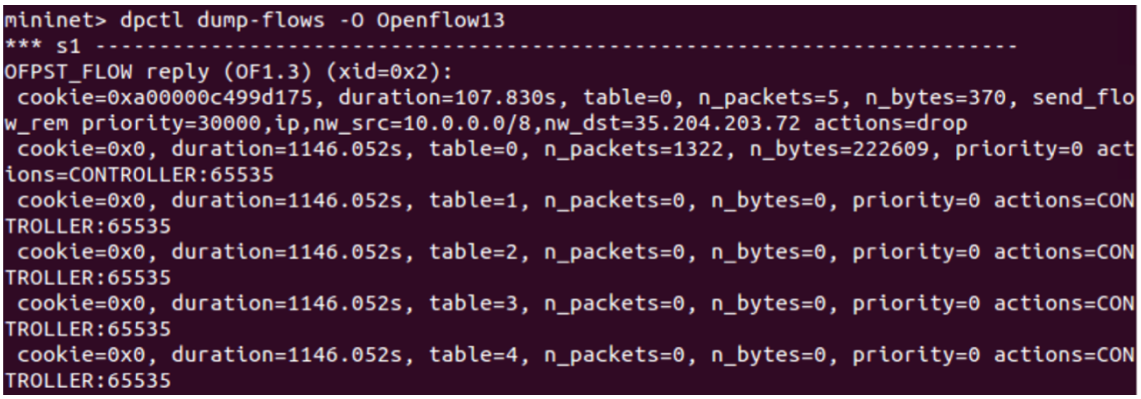
Yakalanan bu paketteki hedef ip adresinin Rest api aracılığı ile YTA kontrolcünün erişim listesine engelleme olarak otomatik eklendiği şekil 4.14'te gözükmektedir.

Şekil 4.14. Rest api ile girilen engel kuralı

Şekil 4.15'te aynı istemcinin bu dosyaya tekrar erişmek istediğinde engellendiği görülmektedir.

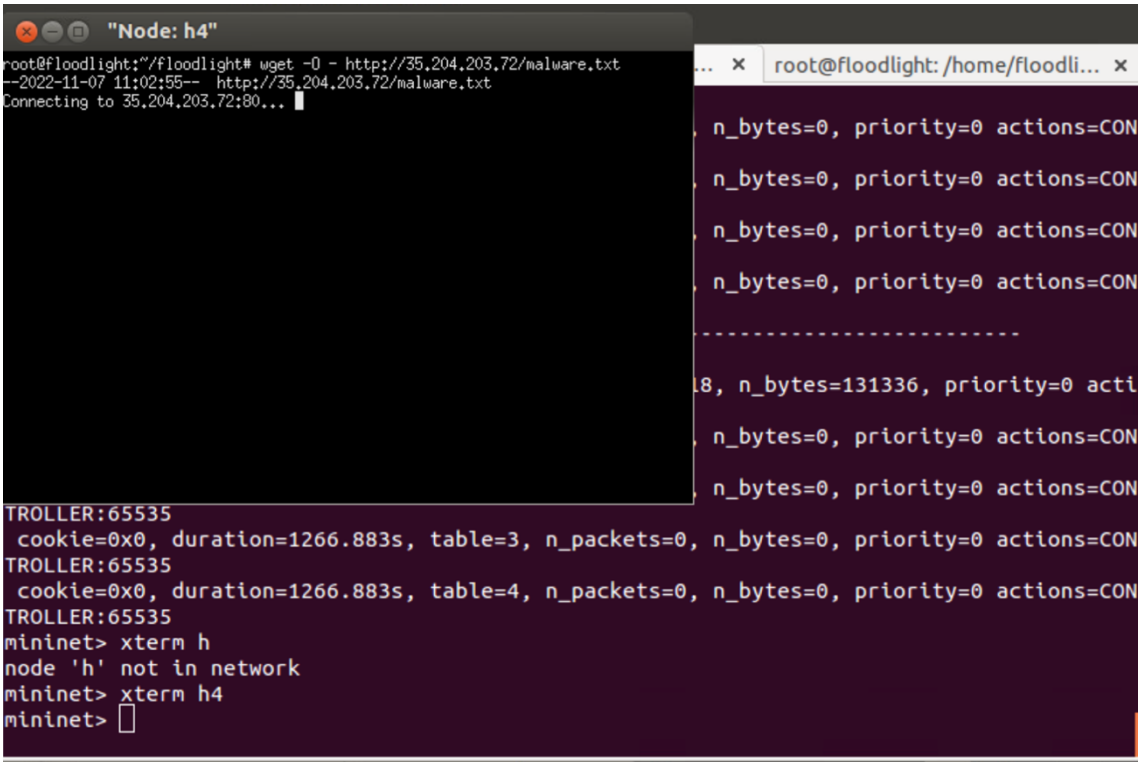
Şekil 4.15. Engellenen malware isteği

Ayrıca şekil 4.16'da gözüktüğü gibi 1. Anahtar üzerindeki flow tablosuna bakıldığında malware bulunan ipye olan isteklerin YTA kontrolcü tarafından engellendiği görülmektedir.



Şekil 4.16. Engelleme sonrası 1. anahtar üzerindeki flow tablosu

Ayrıca şekil 4.17'de diğer anahtarların arkasındaki istemcilerinde bu malware dosyasına erişilemediği görülmektedir.



Şekil 4.17. H4 istemcisinden malware dosyasına engellenen istek

Bu senaryoda aynı YTA kontrolcü tarafından yönetilen farklı ağlardaki herhangi bir istemcinin yaptığı malware isteği sonucunda, uygulama engelleme modülü tarafından mirror porttan alınan trafik içerisinde önceden tanımlanmış kurallar ile arama sonucu REST api yardımıyla tüm ağlardaki istemciler için engelleme yapıldığı kanıtlanmaktadır.

Bu modül ile YTA mimarisinde bulunmayan uygulama katmanında engelleme özelliği YTA mimarisine kazandırılmıştır. Bu modül gerçek dünyada mirror port özelliği bulunan tüm openflow destekleyen anahtarlar(switch) ile uyumlu olarak çalışmaktadır.

YTA senaryosu ile yapılan çalışmanın literatür ile karşılaştırılması ve elde edilen sonuçların tutarlılığı analiz edildiğinde; Chin ve diğerleri (2018), çekirdek modülünü anahtar üzerine konumlandırmıştır. Her anahtar üzerine çekirdek modülü kurulumu gerekiyor. Bu tez çalışmasındaki çekirdek modülü uygulamasıda anahtarla aynı sistem üzerinde çalıştırılabilir. Bir freebsd işletim sistemine openvswitch ve geliştirilen çekirdek modülü kurulduğunda anahtar seviyesinde uygulama filtreleme yapmak mümkündür. Fakat kurumlardaki tüm anahtar donanımlarına ve yazılımlarına bunun kurulması gerçek

dünyada çok mümkün değildir. YTA için üretilen çözümün daha uygulanabilir olması gerekmektedir. Bu tez çalışmasında bahsedilen mirroring ile trafik kopyası alma çözümü ile gerçek dünyada YTA yapısına uygulama filtreleme eklemek daha gerçekçi ve kolaydır.

Nam ve Kim (2018), trafiğin bir kopyası anahtar üzerinden mirroring ile alınmış, açık kaynak kodlu bir yazılım olan suricata ile incelenmiş, engellenmek istenen trafik ise anahtar gibi network aygıtlarına paket gönderilerek engellenmiştir. Bu tez çalışması diğer çalışmadan 2 yönden daha öndedir. Suricata kullanıcı uzayında çalıştığı için yüksek trafikli ağlarda çok sayıda kural ile çalıştırıldığında ağda yavaşlama olacaktır. Bu çalışma çekirdek seviyesinde çalıştığından daha yüksek bant genişliği ve kural sayısında filtreleme yapabilmektedir. (Bu çalışma ile suricata arasında karşılaştırma önceki bölümlerde yapılmıştır). Ayrıca diğer çalışmada engellenmek istenilen kural için network aygıtlarına tek tek engel kuralı gönderilmektedir. Büyük mimarideki bir yapıda ağ aygıtına erişim sorunları, kural senkronizasyonu vb. sorunlar çıkabilir. Ayrıca ağdaki aygıta doğrudan ayar göndermek YTA mimarisine uygun değildir. Bu tez çalışmasında engelleme için REST api aracılığı ile kontrolcüye kural girilmektedir. Kontrolcü bu kuralları network aygıtlarına göndermektedir.

5. SONUÇ ve TARTIŞMA

Aho-Corasick ve Boyer-Moore algoritmalarının kural sayısı arttırılarak yapılan arama testlerinde Aho-Corasick algoritmasının performansının Boyer-Moore algoritmasından daha iyi olduğu tespit edilmiştir. Kural sayısı arttıkça Aho-Corasick algoritmasının çalışma süresi daha yavaş artmıştır. Boyer-Moore algoritmasının çalışma süresininde IDS/IPS sistemler için kabul edilebilir düzeyde olduğu tespit edilmiştir. Fakat kural sayısı fazla ve bant genişliği yüksek sistemlerde Aho- Corasick algoritmasının kullanılması daha doğrudur.

Yapılan çalışma açık kaynak kodlu Suricata yazılımı ile karşılaştırılmıştır. Karşılaştırma sırasında iki yazılımda da arama algoritması olarak Aho-Corasick kullanılmıştır. Aranan tüm kural sayılarında bu tez kapsamında geliştirilen uygulamanın aynı kural sayısında suricata yazılımından yaklaşık 2 kat daha fazla bant genişliğinde filtreleme yapabildiği ve aynı kural sayısı ile aynı bant genişliğinde yaklaşık %20 daha az işlemci gücü tükettiği tespit edilmiştir. Buda filtrelemenin çekirdek seviyesinde yapıldığında daha hızlı yapılabildiğini ve çekirdek seviyesindeki filtrelemelerin daha az işlemci kullandığını kanıtlamaktadır.

Bu tez çalışmasında 100 kuraldan sonra bant genişliği miktarında suricata yazılımına göre daha hızlı bir azalma eğilimi gözükmektedir. Buradan aho-corasick algoritmasının uygulanmasının suricatada daha başarılı yapıldığını sonucuna ulaşmaktayız.

Önerilen çalışma YTA mimarisine başarılı bir şekilde uygulanmıştır. Anahtar cihazlarından trafik kopyası port mirroring ile alınarak paketler çekirdek modülü ile incelenmiştir. Tespit edilen zararlı trafik YTA kontrolcü üzerinden tüm ağda engellenmiştir.

Bu çalışmayı daha ileri taşımak için paket içerisinde desen aramak için çalışmada kullanılan algoritma optimize edilebilir ya da desen arama işlemini hızlandıracak farklı algoritmalar kullanılabilir. Ayrıca geliştirilen uygulama çalışmada sadece uygulama

filtreleme ve YTA mimarisinde kullanılmıştır. Ağ paketlerinde inceleme ve engellemeye ihtiyaç duyulan WAF, antivirüs ve antispam gibi diğer ağ uygulamalarınada uyarlanabilir.

KAYNAKLAR

Anagnostakis, K. G., Antonatos, S., Markatos, E. P., & Polychronakis, M. (2003). E 2 xB: A Domain-Specific String Matching Algorithm for Intrusion Detection. In *Security and Privacy in the Age of Uncertainty* (pp. 217–228). Springer US. https://doi.org/10.1007/978-0-387-35691-4_19

Anithakumari, S., & Chithraprasad, D. (2009). An Efficient Pattern Matching Algorithm for Intrusion Detection Systems. *2009 IEEE International Advance Computing Conference*, 223–227. https://doi.org/10.1109/IADCC.2009.4809011

Antonatos, S., Anagnostakis, K. G., Markatos, E. P., & Polychronakis, M. (2003). Performance analysis of content matching intrusion detection systems. *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, 208–215. https://doi.org/10.1109/SAINT.2004.1266118

Balaban, M. (2005). *Hooking Filters for Fun and Profit: PFIL_HOOKS*. http://www.enderunix.org/Docs/En/Pfil_hook.Html.

Bellevue Linux Users Group. (2005). *Kernel Space Definition*. http://www.linfo.org/kernel_space.html.

Chin, T., Xiong, K., & Rahouti, M. (2018). Kernel-Space Intrusion Detection Using Software-Defined Networking. *ICST Transactions on Security and Safety*, *5*(15), 155168. https://doi.org/10.4108/eai.13-7-2018.155168

Coit, J., Staniford, S., & Mcalemey, J. (2001). *Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort*.

Dell Technologies. (2021). *How to test available network bandwidth using "iperf."* https://www.dell.Com/Support/Kbdoc/Tr-Tr/000139427/How-to-Test-Available-Network-Bandwidth-Using-Iperf?Lang=en.

Dragos, R., & Avram, S.-M. (2010). *Implementation of a layer 7 BSD firewall*.

Filozof. (2008). *Çekirdek (Kernel) Nedir?* https://www.getgnu.org/Gnulinux/Gnulinux-Ipuclari/Cekirdek-Kernel-Nedir.Html.

Fisk, M., & Varghese, G. (2001). *Applying Fast String Matching to Intrusion Detection*.

Freebsd Foundation. (2022). *Character Devices*. https://docs.freebsd.org/en/books/arch-handbook/driverbasics/.

Freebsd Foundation. (2019). *pfil*. https://man.freebsd.org/cgi/man.cgi?query=pfil&sektion=9.

GeeksforGeeks. (2022). *Pattern Searching*. https://www.geeksforgeeks.org/Algorithms-%20gq/Pattern-Searching/.

Gharaee, H., Seifi, S., & Monsefan, N. (2014). A survey of pattern matching algorithm in intrusion detection system. *7'th International Symposium on Telecommunications (IST'2014)*, 946–953. https://doi.org/10.1109/ISTEL.2014.7000839

Gouda, M. G., & Liu, A. X. (2007). Structured firewall design. *Computer Networks*, *51*(4). https://doi.org/10.1016/j.comnet.2006.06.015

Govil Ayush. (2022). *Aho-Corasick Algorithm for Pattern Searching*. https://www.geeksforgeeks.org/Aho-Corasick-Algorithm-Pattern-Searching/.

Hasan, A. A., Hasan, A. A., Aini, N., & Rashid, A. (2012). *Hash-Boyer-Moore-Horspool String Matching Algorithm for Intrusion Detection System Childhood Obesity Prediction Using Data Mining View project Information and Network Security Threats View project Hash-Boyer-Moore-Horspool String Matching Algorithm for Intrusion Detection System*. https://www.researchgate.net/publication/323971262

Hume, A., & Sunday, D. (1991). Fast string searching. *Software: Practice and Experience*, *21*(11), 1221–1248. https://doi.org/10.1002/spe.4380211105

İnan Aydın. (2010). *Aho Corasick algoritmasının TCP/IP Paket içeriklerinin taranmasında kullanıldığı gerçek zamanlı donanım uygulaması*. Hacettepe Üniversitesi.

IEEE Standard for Ethernet. (2018). *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, 1–5600. https://doi.org/10.1109/IEEESTD.2018.8457469

Le Dang, N., Nhuong Le, D., & Trong Le, V. (2016). A New Multiple-Pattern Matching Algorithm for the Network Intrusion Detection System. *International Journal of Engineering and Technology*, *8*(2), 94–100. https://doi.org/10.7763/ijet.2016.v8.865

Minghao, K., Chyang, K. Y., & Karuppiah, E. K. (2007). Performance Analysis and Optimization of User Space versus Kernel Space Network Application. *2007 5th Student Conference on Research and Development*, 1–6. https://doi.org/10.1109/SCORED.2007.4451372

Nam, K., & Kim, K. (2018). A Study on SDN security enhancement using open source IDS/IPS Suricata. *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 1124–1126. https://doi.org/10.1109/ICTC.2018.8539455

Norton, M. (2004). *Optimizing Pattern Matching for Intrusion Detection*. www.idsresearch.org

Patel, K. C., & Priyanka Sharma, D. (2017). *A Review paper on pfsense-an Open source firewall introducing with different capabilities & customization* (Vol. 3). https://www.untangle.com/par

Reinholz Dan, & Reinholz Kevin. (2022). *3 Reasons to use FreeBSD*. https://www.ocf.berkeley.edu/~reinholz/freebsd/3reasons.html.

Stewart, L., & Healy, J. (2010). *An introduction to FreeBSD 6 kernel hacking*.

Suhendra, K. (2016). *Application of Boyer-Moore and Aho-Corasick Algorithm in Network Intrusion Detection System*. https://koding4fun.files.wordpress.com/2010/05/complete_example.jpg

Tahir Bilal. (2011). Content based packet filtering in Linux kernel using deterministic finite automata. Orta Doğu Teknik Üniversitesi.

Vidanagamachchi, S. M., Dewasurendra, S. D., Ragel, R. G., & Niranjan, M. (2012). COMMENTZ-WALTER: ANY BETTER THAN AHO-CORASICK FOR PEPTIDE IDENTIFICATION? *International Journal of Research in Computer Science EISSN 2249*, *8265*(6), 33–37. https://doi.org/10.7815/ijorcs

Yang Dong hong, Xu Ke, & Cui Yong. (2006). An Improved Wu-Manber Multiple Patterns Matching Algorithm. *2006 IEEE International Performance Computing and Communications Conference*, 675–680. https://doi.org/10.1109/.2006.1629469