T.C.
BİTLİS EREN ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI
YÜKSEK LİSANS TEZİ

# KAMPUS AĞLARINDA İSTENMEYEN DHCP SUNUCULARININ YTA İLE TESPİT EDİLMESİ

Cengiz ALMAZ

HAZİRAN 2020

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI
YÜKSEK LİSANS TEZİ

# KAMPUS AĞLARINDA İSTENMEYEN DHCP SUNUCULARININ YTA İLE TESPİT EDİLMESİ

Hazırlayan
Cengiz ALMAZ

Danışman
Dr. Öğr. Üyesi Musa ÇIBUK

Jüri Üyeleri
Prof. Dr. Sabir RÜSTEMLİ
Dr. Öğr. Üyesi Musa ÇIBUK
…… (varsa)

HAZİRAN 2020

**ONAY**

Cengiz ALMAZ tarafından hazırlanan "**Kampus Ağlarında İstenmeyen DHCP Sunucularının YTA ile Tespit Edilmesi**" adlı tez çalışması …../…../2020 tarihinde yapılan sınavla aşağıdaki jüri tarafından oybirliği/oyçokluğu ile Bitlis Eren Üniversitesi Lisansüstü Eğitim Enstitüsü Elektrik Elektronik Mühendisliği Anabilim Dalı'nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

| **Jüri Üyeleri** | **İmza** |
|---|---|
| Ünvanı, Adı ve Soyadı<br>(Başkan) | _____ |
| Ünvanı, Adı ve Soyadı<br>(Danışman) | _____ |
| Ünvanı, Adı ve Soyadı<br>(Üye) | _____ |
| Ünvanı, Adı ve Soyadı<br>(Üye) (varsa) | _____ |
| Ünvanı, Adı ve Soyadı<br>(Üye) (varsa) | _____ |

Bu tezin kabulü, Lisansüstü Eğitim Enstitüsü Yönetim Kurulu'nun …../…../2020 gün ve ..…/….. sayılı kararı ile onaylanmıştır.

Prof. Dr. Zeki ARGUNHAN

Enstitü Müdürü

**BİTLİS EREN ÜNİVERSİTESİ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ**
**YÜKSEKLİSANS TEZ ÇALIŞMASI**
**ETİK BEYANI**

Bitlis Eren Üniversitesi Lisansüstü Eğitim Enstitüsü tez yazım kılavuzuna göre hazırlamış olduğum "**Kampus Ağlarında İstenmeyen DHCP Sunucularının YTA ile Tespit Edilmesi**" adlı tezimin özgün bir çalışma olduğunu, tez hazırlanırken tüm aşamalarda bilimsel etik ilkelerine uygun davrandığımı, tez kapsamında sunulan tüm verileri bilimsel etik ilkelerine uygun elde ettiğimi, tezde faydalandığım tüm eserlere atıf yaptığımı ve kaynaklar kısmında bu eserleri gösterdiğimi beyan ederim. ……/……/2020

**Cengiz ALMAZ**

# ÖZET

KAMPUS AĞLARINDA İSTENMEYEN DHCP SUNUCULARININ YTA İLE TESPİT EDİLMESİ

Cengiz ALMAZ

Yüksek Lisans Tezi

Bitlis Eren Üniversitesi Lisansüstü Eğitim Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı
Danışman: Dr. Öğr. Üyesi Musa ÇIBUK
Haziran 2020, 68 sayfa

Son yıllarda teknolojinin farklı alanlarda bütünleşik olarak kullanılması ile beraber bilgisayar ağları üzerindeki yük de artmıştır. Geleneksel bilgisayar ağlarının sürekli olarak ekleme mantığıyla büyümesi, bu ağların yönetilebilirliğini zorlaştırmıştır. Ağ cihazlarındaki yeniliklerin, cihazlarda kullanılan teknolojilere bağlı olması, yeni teknolojileri yeni cihazlara bağımlı hale getirmektedir. Bu durum kullanıcı tarafında büyük maliyet ve zaman kaybına neden olmaktadır. Bu tür sorunlardan bir nebzede olsa kurtulmak için ağlarda yazılım kullanılması yaklaşımı benimsenmiştir. Yazılım Tanımlı Ağ (YTA) paradigması, geleneksel ağları dinamik bir yapıya kavuşturmak, ağ yönetimini kolaylaştırmak ve ağ cihazlarına yazılım yoluyla işlevsellik kazandırarak kullanıcıları cihaz bağımlılığından kurtarmak amacıyla ortaya atılmıştır.

Bu tez çalışmasında bilgisayar ağlarında cihazların internete çıkması için gerekli yapılandırmaları otomatik olarak sağlayan Dinamik Bilgisayar Yapılandırma Protokolü (DHCP) sunucularının güvenliği için gerekli olan bazı önlemlerden bahsedilmiştir. Bu kapsamda sahte DHCP saldırıları üzerinde çalışılmıştır. Ağda bulunan sahte bir DHCP sunucusunun faaliyetlerini önlemek amacıyla YTA yaklaşımı kullanılmıştır. Böylece ağda bulunan sahte DHCP sunucuları tespit edildikten sonra IP dağıtımlarını engelleyecek önlemler alınmıştır. Çalışmalar Mininet benzetim ortamında gerçekleştirilmiştir.

**Anahtar kelimeler**: Yazılım Tanımlı Ağlar, YTA, Mininet, Sahte DHCP, Ağ Güvenliği

## ABSTRACT

DETERMINING UNWANTED DHCP SERVER ON THE CAMPUS NETWORKS WITH THE SDN

Cengiz ALMAZ

Master Thesis

Bitlis Eren University Graduate Education Institute
Department of Electrical and Electronics Engineering
Supervisor: Asst. Prof. Dr. Musa ÇIBUK
June 2020, 68 pages

In recent years, the load on computer networks has increased with the integrated use of technology in different areas. The continuous growth of traditional computer networks by the addition logic has made the manageability of these networks difficult. The fact that the innovations in network devices depend on the technologies used in the devices makes new technologies dependent on new devices. This situation causes great cost and time loss for the user. In order to get rid of such problems, to some extent, the approach of using software in networks has been adopted. The Software Defined Network (SDN) paradigm has been developed in order to make the traditional networks a dynamic structure, to facilitate network management and to provide functionality to the network devices through software, to save users from device dependency.

In this thesis, some precautions necessary for the security of the Dynamic Host Configuration Protocol (DHCP) servers, which automatically provide the necessary configurations for the internet access of the devices in computer networks, are mentioned. In this context, rogue DHCP attacks were studied. SDN approach was used to prevent the activities of a rogue DHCP server on the network. Thus, after detecting rogue DHCP servers on the network, measures to prevent IP distributions were taken. The studies were carried out in Mininet simulation environment.



**Keywords**: Software Defined Network, SDN, Mininet, Rogue DHCP, Network Security

## TEŞEKKÜR

Bu tez çalışması sırasında, tez konusunun belirlenmesinden başlayarak tezin son aşamasına kadar benden yardımlarını esirgemeyen ve bana her konuda yol gösteren danışman hocam sayın Dr. Öğr. Üyesi Musa ÇIBUK'a şükranlarımı sunarım. Ayrıca çalışmam süresince gösterdiği iyi niyet ve yardımlarından dolayı Elektrik-Elektronik Mühendisliği Ana Bilim Dalı Başkanı sayın Prof. Dr. Sabir RÜSTEMLİ'ye de teşekkürü bir borç bilirim.

Bu günlere gelmemde büyük emekleri olan annem ve babama, yoğun çalışma temposunda maddi ve manevi desteğini benden esirgemeyen eşime, varlıkları ile bana motivasyon kaynağı olan mesai arkadaşlarıma ve kardeşim Mehmet Şirin ALMAZ'a teşekkür ederim.

# ÖNSÖZ

Son zamanlarda bilgi teknolojilerindeki kullanım alanlarının farklılaşması ile birlikte bilgisayar ağları üzerindeki yük gün geçtikçe artmaktadır. Bunlar ile birlikte her yerde ve her an erişilebilirlik, yüksek bant genişliği ihtiyacı, dinamik yönetim araçları gibi yeni gereksinimleri de beraberinde getirmektedir. Geleneksel ağlarda kullanılan ağ cihazlarından oluşan eklemeli yapı sistemi hantal ve hataya eğilimli bir hale getirmektedir.

YTA (Yazılım Tanımlı Ağlar) bu anlamda geleneksel ağlarda meydana gelen sorunları çözüme odaklı olarak geliştirilmektedir. YTA geleneksel ağlarda statik yapıda olan cihazları yazılım yardımıyla daha dinamik bir yapıya kavuşturmak istemektedir. Bunun bir sonucu olarak, YTA yenilikçi ağ tasarımlarına olan ihtiyacı karşılaması, daha etkili yapılandırmaya sahip, daha iyi performans ve daha çok esneklik sağlayan ağ yapısı yaklaşımıyla ağ mimarisini yenilemektedir.

Bu çalışmada YTA kullanılarak istenmeyen DHCP sunucularının tespiti ve engellenmesi konusu irdelenmiştir. Çalışmalar mininet benzetim ortamında gerçekleştirilerek istenmeyen sahte DHCP sunucularının tespiti ve engellenmesi gerçekleştirilmiştir. YTA kontrolcü olarak POX kontrolcüsü kullanılmıştır.

# İÇİNDEKİLER DİZİNİ

ÖZET ..... i
ABSTRACT ..... ii
TEŞEKKÜR ..... iii
ÖNSÖZ ..... iv
İÇİNDEKİLER DİZİNİ ..... v
ÇİZELGELER DİZİNİ ..... vii
ŞEKİLLER DİZİNİ ..... viii
KISALTMALAR DİZİNİ ..... ix
1. GİRİŞ ..... 1
1.1. Tez Organizasyonu ..... 4
1.1.1. DHCP Açlık (Starvation) Saldırıları ..... 5
1.1.2. Sahte (Rogue) DHCP Saldırıları ..... 6
2. MATERYAL VE YÖNTEM ..... 9
2.1. Yazılım Tanımlı Ağlar (YTA) ..... 9
2.1.1. Kontrol Katmanı ve Veri Katmanının Birbirinden Ayrılması ..... 10
2.1.2. OpenFlow ..... 11
2.2. YTA Mimarisi ..... 13
2.2.1. Veri Katmanı ..... 15
2.2.1.1. Ağ Altyapısı ..... 15
2.2.1.2. Güney Arayüzü ..... 16
2.2.2. Kontrol Katmanı ..... 17
2.2.2.1. Ağ Hypervisor ..... 17
2.2.2.2. Ağ İşletim Sistemi ..... 18
2.2.2.3. Kuzey Arayüzü ..... 19
2.2.3. Yönetim Katmanı ..... 20
2.2.3.1. Dil Tabanlı Sanallaştırma ..... 20

2.2.3.2. Programlama Dilleri ..... 21
2.2.3.3. Ağ Uygulamaları ..... 21
2.3. Mininet ..... 22
2.4. POX Kontrolcüsü ..... 22
2.5. Wireshark ..... 22
2.6. VirtualBox ..... 23
2.7. Python ..... 24
2.8. Linux ..... 25
2.9. Dynamics Host Configuration Protocol (DHCP) ..... 25
2.9.1. DHCPv4 Paket Yapısı ..... 26
2.9.2. Ağ Cihazlarına Otomatik IP Verilmesi ..... 28
2.9.3. Ağ Bağlantısı için Gerekli Asgari Bilgiler ..... 30
2.9.4. DHCP Relay ..... 31
2.9.5. DHCP Güvenliği ..... 31
2.9.6. Sahte (Rogue) DCHP Saldırıları ..... 32
2.10. Domain Name Systems (DNS) ..... 34
3. BULGULAR VE TARTIŞMA ..... 35
3.1. Benzetim Çalışması ..... 35
3.1.1. Ağ Senaryosu ..... 35
4. SONUÇ ..... 47
5. KAYNAKLAR ..... 48
3. EKLER ..... 52
ÖZGEÇMİŞ ..... 68

# ÇİZELGELER DİZİNİ

**ÇİZELGE** **Sayfa**

1.1. YTA ile geleneksel ağların karşılaştırılması [6] ..... 4
2.1. Bazı OpenFlow kontrolcüleri [3] ..... 13
3.1. Benzetim ortamında kullanılan yazılım ve donanım bilgileri ..... 35

# ŞEKİLLER DİZİNİ

**ŞEKİL** **Sayfa**

1.1. IoT uygulama alanları....1
1.2. DHCP saldırı koruma modeli çalışma mantığı....7
2.1. Geleneksel (a) ve YTA (b) mimarilerinde kontrol ve veri katmanının ayrılması....11
2.2. Yazılım tanımlı ağlarda OpenFlow protokolü ve OpenFlow anahtarı....12
2.3. Kuzey - güney ara yüz köprüleri ve çalışma mantıkları....13
2.4. (a) YTA katman yapısı, (b) YTA katman üniteleri, (c) Sistem tasarım mimarisi....14
2.5. OpenFlow akış tablolarında paket yönlendirme işlemleri....16
2.6. Popüler Programlama Dilleri....24
2.7. DHCPv4 Paket Formatı....26
2.8. Otomatik IP alma-verme işlemi....28
2.9. Sahte DHCP sunucusu....33
3.1. Kullanılan ağ senaryosu....35
3.2. POX kontrolcüsü pasif modda....36
3.3. POX kontrolcüsü aktif modda....37
3.4. Yasal DHCP sunucusu başlatılması....37
3.5. Yasal DNS sunucusu başlatılması....38
3.6. Sahte DHCP sunucusu başlatılması....38
3.7. Sahte DNS sunucusu başlatılması....39
3.8. Programın başlatılması....39
3.9. POX kontrolcüsü pasif modda başlatılması....40
3.10. POX kontrolcüsü pasif modda çalıştırıldığında Wireshark görüntüsü....40
3.11. h1 bilgisayarının durumu....41
3.12. h1 bilgisayarından ping atılması....42
3.13. h2 bilgisayarının durumu....42
3.14. h2 bilgisayarından ping gönderilmesi....43
3.15. POX kontrolcüsünün aktif modda başlatılması....43
3.16. POX kontrolcüsü aktif modda çalıştırıldığında Wireshark görüntüsü....44
3.17. h1 bilgisayarının durumu....44
3.18. h2 bilgisayarının durumu....45
3.19. POX kontrolcüsünün IP eşleştirmesi....45
3.20. Pasif mod denemesi....46
3.21. Bilgisayar h1 IP denemesi....46

## KISALTMALAR DİZİNİ

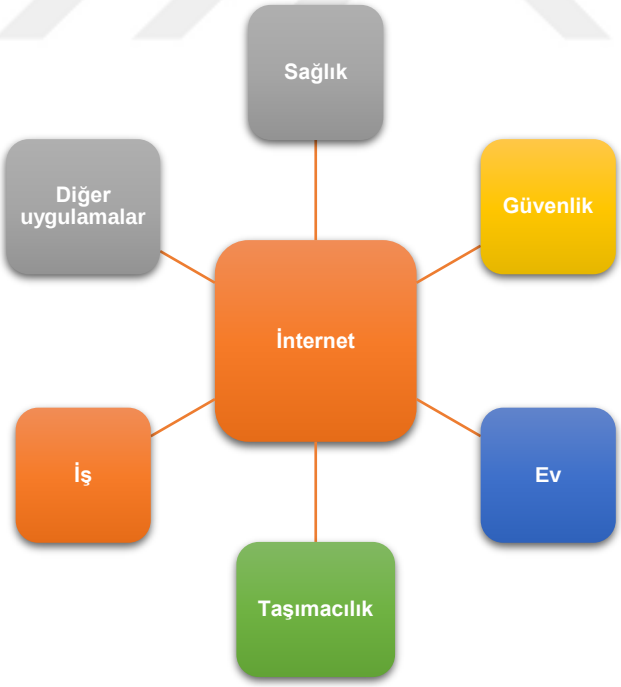
| | | |
|---|---|---|
| YTA | Yazılım Tanımlı Ağlar | Software Defined Network (SDN) |
| IoT | Nesnelerin İnterneti | Internet of Things |
| DHCP | Dinamik İstemci Ayarlama Protokolü | Dynamic Host Configuration Protocol |
| DoS | Hizmet Engelleme | Denial of Service |
| IP | İnternet Protokol | Internet Protocol |
| DNS | Alan Adı Servisi | Domain Name Service |
| MAC | Ortam Erişim Kontrolü | Media Access Protocol |
| OSI | Açık System rabağlantısı | Open System Interconnection |
| TCP | Geçiş Kontrol Protokolü | Transmission Control Protocol |
| UDP | Kullanıcı Verikatarı Protokolü | User Datagram Protocol |
| ONF | Açık Ağ Kuruluşu | Open Networking Foundation |
| SSL | Güvenli Giriş Katmanı | Secure Sockets Layer |
| API | Uygulama Programlama Arayüzü | Application Programming Interface |
| CPU | İşlemci | Central Processing Unit |
| NOS | Ağ İşletim Sistemi | Network Operation System |
| RFC | Açıklama İsteği | Request For Comments |
| HTTP | Hiper Metin Transferi Protokolü | Hyper-Text Transfer Protocol |
| SSH | Güvenli Kabuk | Secure Shell |
| VoIP | İnternet Üzerinden Sesli İletişim | Voice over IP |
| BOOTP | Önyükleme Protokolü | Bootstrap Protocol |
| IETF | İnternet Mühendisliği Görev Gücü | Internet Engineering Task Force |
| VLAN | Sanal Yerel Alan Ağı | Virtual Local Area Network |
| HType | Donanım Tipi | Hardware Type |
| HLen | Donanım Adres Uzunluğu | Hardware Adress Length |
| XID | İşlem Tanımlayıcı | Transaction Identifier |
| CIAddr | İstemci IP Adresi | Client IP Address |
| YIAddr | Kullanılan IP Adresi | Your IP Address |
| SIAddr | Sunucu IP Adresi | Server IP Address |
| GIAddr | Geçit Kapısı IP Adresi | Gateway IP Address |
| CHAddr | İstemci Donanım Adresi | Client Hardware Address |
| Sname | Sunucu ismi | Server Name |
| BT | Bilişim Teknolojileri | Information Tecnology (IT) |

| | | |
|---|---|---|
| URL | Standart Kaynak Bulucu | Uniform Resource Locator (URL) |
| TTL | Yaşam Süresi | Time To Live |
| IPv4 | İnternet Protokol sürüm 4 | Internet Protocol version 4 (IPv4) |
| SNMP | Basit Ağ Yönetimi Protokolü | Simple Network Management Protocol |
| FQDN | Tam Nitelikli Domain Adı | Fully Qualified Domain Name |

# 1. GİRİŞ

Son yıllarda teknolojinin hızla gelişmesi ve beraberinde imkânlarında artmasıyla birlikte dijital dünyaya olan bağlantı sınırsız bir şekilde büyümektedir. Günümüzdeki internet kullanıcı sayısı dünya nüfusunun yarısını geçmiştir. İnsanoğlunun haberleşme, banka işlemleri, ulaşım, sosyal faaliyet gibi hizmetlerde teknoloji altyapısını kullanması bilgisayar ağlarının önemini ve yükünü kat kat artırmıştır. İnternet kullanımına 12 yaş civarında başlandığını ve dünyada interneti kullanan kişi sayısının dört milyardan fazla olduğu günümüzde yapılan araştırmalarla ortaya çıkmıştır [1].

Bu sonuç bizlere dijital dünyadaki kullanıcıların hızlı bir şekilde artacağını göstermektedir. Bu hızlı değişim karşısında bilgisayar ağları yeni bir yapı yerine eski yapıya eklemeler yapılarak büyümektedir. Fakat günümüzde ağ cihazları çok farklı amaçlara hizmet etmektedir. Cep telefonları ile gün içinde alış verişten bankacılık işlemlerine haberleşmeden elektronik posta işlemlerine farklı uygulama alanlarına bağlı olarak internet hayatımızın önemli bir parçası haline gelmiştir. Özelde Internet of Things (IoT) olarak adlandırılan ve günlük hayatta farklı alanlardaki faaliyetlerin bütünleşik kontrolünü sağlayan sistem Şekil 1.1'de gösterilmiştir.




**Şekil 1.1.** IoT uygulama alanları [2]

Bilgi ve iletişim dünyamızdaki hızlı değişim beraberinde bant genişliği, erişebilirlik, bütünlük, güvenlik, dinamik yönetim gibi talepleri de beraberinde getirmiştir. Bu durum geleneksel ağ yapılarında karmaşıklığa ve yönetim sorunlarına neden olmaktadır. Geleneksel ağlarda meydana gelen alt yapı sorunlarının çoğu yönetimsel sorunlardır [3]. Bilgisayar ağlarında özellikle bu konuda yetişmiş eleman eksikliği göze çarpmaktadır. Bu nedenle geleneksel ağlarda meydana gelen sorunların birçoğunun çözümünde ağ yönetiminde insan kaynaklarının oldukça sınırlı olması nedeniyle farklı arayışlara yönelinmesi gerektiği sonucuna varılmaktadır.

Bilgisayar ağlarındaki sorunların devam etmesinin bir diğer nedeni ise cihaz üreticilerinin kullanıcıları kendilerine bağımlı tutma istekleridir. Bu şekilde piyasadaki ilerlemeyi kontrol edebilir ve teknolojiyi istedikleri gibi yönlendirebilirler. Cihazların işlevselliğinin dinamik olmaması, teknolojik ilerlemelerin fiziksel yapıya bağlı olarak sınırlandırılması kullanıcıyı maddi yönden oldukça zarara sokmaktadır. Ağ cihazlarındaki yenilikler fiziksel yapıya veya modele bağlı olarak sınırlı kalmakta ve kullanıcıların firmalara olan bağımlılığını artırmaktadır. Firmalar modellerdeki teknolojilere bağlı olarak maliyetlerinin çok üstünde ürünlerini pazarlayabilmektedirler. Bu sorunların hepsi göz önüne alındığında bilgisayar ağlarındaki bu kısır döngünün önüne geçebilecek tek yapının yazılım olduğu ortaya çıkmaktadır.

Yazılım Tanımlı Ağlar (YTA) geleneksel ağlardaki sorunların çözümlerine bağlı olarak ortaya çıkmıştır. YTA adından da anlaşılacağı gibi bilgisayar ağlarında yazılım desteğini sunmakla beraber birçok soruna da çözüm sunmaktadır. Esnek, programlanabilir, dinamik yapısı sayesinde sürdürülebilir bir yapıyı benimsemektedir.

YTA gün geçtikçe kullanım alanlarını da genişletmektedir. YTA mimarisi, çok uzun yıllardan beridir sadece fiziksel yapıda değişiklik göstermiş olan geleneksel ağ mimarisinin yerine kullanılması için geliştirilmektedir. Bundan dolayı sahada kullanıma başlanması kolay değildir. Bunun temel nedenlerinden biri şuan kullanılmakta olan cihazların çok büyük bir kısmının YTA yapısını desteklememesidir. Yani böyle bir yapıya uygun olarak üretilmemesidir.

YTA'nın en önemli avantajlarında biri kullanıcıyı marka ve model bağımlılığından kurtarmasıdır. Bunu yaparken cihazlar üzerinde yazılım kullanılması amaçlanmıştır. Böylece ağ yöneticisi yeni bir uygulamaya ihtiyacı olduğunda cihaz üzerinde yazılım yapılmasına imkân sağlayan arabirim sayesinde uygulamayı donanıma aktarabilecek ve elinde var olan donanımı farklı bir amaç için kullanabilecektir. Aynı zamanda maliyet ve zaman olarak da avantajlı duruma geçebilecektir. Bundan dolayı var olan yapıyı hemen elden çıkarmak yerine onu değerlendirmek kullanıcı tarafında mali açıdan büyük avantajlar sağlayacaktır.

YTA'yı sahada uygulayamamanın nedenlerinden biri de onu destekleyen protokoldeki cihazların maliyetlerinin şimdilik fazla olmasıdır. Bundan dolayı bu yapının farkında olan ve onu

kullanmak isteyen kurumlar ellerindeki cihazların kullanım ömrünü tamamlamalarını beklemektedirler.

Geleneksel ağlarda kullanılan cihazların birçoğunun konfigüre edilmesi pek fazla bilgi gerektirmemektedir. Hatta birçoğunun tak-çalıştır şeklinde kurulumu mümkündür. YTA'da ise cihazın ağa eklenmesi basit fakat yapılandırılması ve yönetilmesi denetleyici olarak adlandırılan uzak yönetim cihazı ile yapılmaktadır. Bu durum ağ konusunda uzmanlaşmış kişilerce yapılabilmekte bu da geleneksel ağ yapısından YTA'ya geçişi yavaşlatmaktadır [4].

YTA bilgisayar ağlarının yapısına yeni bir anlayış kazandırmış, geleneksel ağlardan farklı olarak bilgisayar ağlarını kontrol katmanı ve veri katmanı olmak üzere ikiye bölmüştür. Kontrol katmanında bulunan denetleyici ağın büyüklüğüne ve ihtiyaca göre bir veya birden fazla cihazdan meydana gelebilir. Denetleyici mantıksal işlemlerin yapıldığı birim olup, ağ yönetimi, paket yönlendirme, yük dengeleme gibi bilgisayar ağları için hayati önem taşıyan işlemler buradan idare edilmektedir. Geleneksel ağdaki paket yönlendirme işlemleri ile ilgili kararlar anahtarlama cihazları tarafından yapılırken YTA'da bu kararlar denetleyiciler (kontrolcüler) tarafından anahtarlama cihazları üzerinde oluşturulan akış tablolarına işlenerek oluşturulur. YTA'da bir ağın gereksinimi olan paket trafik yönetimi, ağ erişim kontrolü ve bant genişliği gibi önemli kararlar YTA'nın kontrol biriminde toplanır.

Geleneksel ağlarda kullanılan cihazların birçoğu farklı üreticiler tarafından üretildiğinden dolayı ağın tek noktadan yönetimi ve ölçeklendirilmesi oldukça zordur. Geleneksel ağlarda gelişmiş ağ yönetim kurallarını uygulamak cihazların birçoğunun OpenFlow destekli olmamasından dolayı neredeyse imkânsızdır. Geleneksel ağların kullanımının gün geçtikçe YTA'ya göre daha hızlı büyümesi ağ yönetimini daha da kısır bir döngüye sokmaktadır. [5]

YTA dinamik yapıları, programlanabilir özellikleri sayesinde bilgisayar ağlarının gelişimine, yönetimine bundan dolayı da büyümelerine imkân sağlar. Bu yüzden kullanım alanları istenilen düzeyde olmamakla beraber gün geçtikçe artmaktadır. Kullanım alanları olarak, veri merkezleri, bulut bilişim, kurumsal ağlar, kablolu ve kablosuz ağlar sıralanabilir.[3]

Daha genel bir perspektif oluşturmak için YTA ile geleneksel ağların kıyaslanması Çizelge 1.1 de verilmiştir.

**Çizelge 1.1.** YTA ile geleneksel ağların karşılaştırılması [6]

| Özellik | YTA | Geleneksel Ağlar |
|---|---|---|
| Ağ yönetimi | Kontrol bir noktadan sağlandığından kolaydır | Değişiklikler her cihaza ayrı ayrı uygulandığından zordur |
| Global ağ görünümü | Kontrolcü üzerinden kolay | Zordur |
| Bakım maliyeti | Düşük | Yüksek |
| Güncelleme/hata işleme için gereken süre | Merkezi kontrolcü sayesinden oldukça kolay | Ağın büyüklüğüne göre bazen aylar sürebilir |
| Kontrolcü kullanımı | Uygun | Uygun değil |
| Kontrolcünün, tutarlılığı uyumluluğu, güvenirliği | Önemli | Önemli değil |
| Yönlendirmenin bütünlüğü ve tutarlığı | Önemli | Önemli |
| Kullanılabilirlik kapsamı | Yüksek | Düşük |

Son zamanlarda, ağı çeşitli güvenlik saldırılarına karşı korumak, araştırma ve geliştirmesinde çok popüler bir odak alanı haline gelmiştir [7]. Farklı bir ağ paradigması olan YTA mimarisinde yapılan saldırılar daha heyecan verici bir hale gelmiştir.

YTA, geleneksel ağlarda kullanılan ağ donanımlarının kullanımı ile ilgili statik yapıyı dinamik, ihtiyaca göre programlanabilen esnek bir yapıya kavuşturmuştur. Ağ trafiğindeki paket anahtarlamalı gönderim mantığı aynı kaldığından mevcut saldırıların birçoğu yazılım tanımlı ağlarda da devam etmektedir. Gelişen mimarisi itibariyle yapılan saldırılara karşı ağ yöneticilerinin elini güçlendirmiştir.

### 1.1. Tez Organizasyonu

Bölüm 1'de tez çalışması ile ilgili giriş bilgilerine yer verilmiş olup tez organizasyonundan bahsedilmiştir. Ayrıca bu bölümde tez çalışmasının konusu ile ilgili önceden yapılan çalışmalardan bahsedilmiştir. YTA'da daha önce yapılmış olan saldırı türleri ile ilgili çalışmalar bu bölümde incelenmiştir.

Bölüm 2'de tezde kullanılan materyal ve yöntemden genel olarak bahsedilmiştir. YTA'nın genel mimarisinden bahsedilmiş olup geleneksel ağlarda kullanılan mimari ile karşılaştırılması yapılmıştır. Aynı zamanda YTA'da kullanılan ağ teknolojilerinin avantajlarına bu bölümde değinilmiştir. Ayrıca bu bölümde tez konusu olan DHCP sunucularının ağ ortamındaki işlevinden bahsedilmiş olup çalışma mantığı detaylı şekilde anlatılmıştır.

Bu çalışmada benzetim ortamını hazırlamak için kullanılan Mininet benzetim uygulaması ile ilgili bilgiye bu bölümde yer verilmiştir. Tez çalışmasında kullanılan tüm donanım ve yazılım

bilgilerine yer verilmiştir. Bunlara ek olarak DHCP güvenliği ilgili yapılan çalışmalara ayrıntılı yer verilmiştir.

Bölüm 3'te benzetim ortamında kullanılan ağ senaryosu hakkında görsel bilgi verilmiştir. Ağ senaryosunda kullanılan sahte DHCP, yasal DHCP ve kontrolcü ile ilgili genel bilgiler verilmiştir.Ağda olabilecek sahte DHCP sunucularının, ağdaki DHCP istemcilere IP vermelerini engellemek için yapılan çalışma anlatılmış olup senaryolar üzerinde anlatım gerçekleştirilmiştir.

### 1.1.1. DHCP Açlık (Starvation) Saldırıları

DHCP açlık saldırıları aynı zamanda DoS (Denial of Service) saldırı türlerinden biridir [8]. Açlık saldırıları gerek geleneksel ağlarda gerekse YTA'da en popüler saldırı çeşitlerinin başında gelmektedir. DHCP protokolünde tanımlanan kurallar çerçevesinde DHCP istemcisi ile DHCP sunucusu arasında kimlik doğrulaması gibi bir durum söz konusu olmadığından saldırganlar bu alanda kolayca istediklerini yapabilmektedir [7]. Alt ağlarda kullanılan IP sayısı ağ yöneticisi tarafından o ağın büyüklüğü göz önüne alınarak belirlenir ve DHCP havuzunda kullanılabilir olarak tanımlanır.

Bu saldırı türünde saldırganlar bu havuzda bulunan IP adreslerini ele geçirdiği istemci bilgisayarlar ile sürekli IP isteğinde bulunarak havuzda bulunan tüm adresleri kiralamaya çalışır. Böylece DHCP sunucusunun dağıtacak başka IP adresi kalmadığından devre dışı kalmış olur.

Cansu Toprak ve arkadaşları [7] Mininet sanal ortamında gerçekleştirdikleri çalışmalarında DHCP açlık saldırısı çeşidini işlemektedirler. YTA Denetleyicisi üzerinde tanımladıkları DHCP sunucularına DHCP istemcilerden gelen mesajları izleyebilme avantajını değerlendirmişlerdir. DHCP istemcilerinden gelen DHCP_DISCOVER mesajlarının sayısında belirtilen süre içerisinde anormal bir artış ile eşik değeri (MAX_DHCP_PKT) aşması durumunda bir saldırı olabileceğine karar veriliyor. Bu durumda ilgili anahtar cihazından gelen IP istek mesajları saldırı devam ettikçe denetleyiciye yönlendirilmiyor. (MAX_DHCP_PKT) sayısı normale döndüğü zaman ise denetleyicide tanımlanan DHCP sunucusu işlemlerine devam edebiliyor.

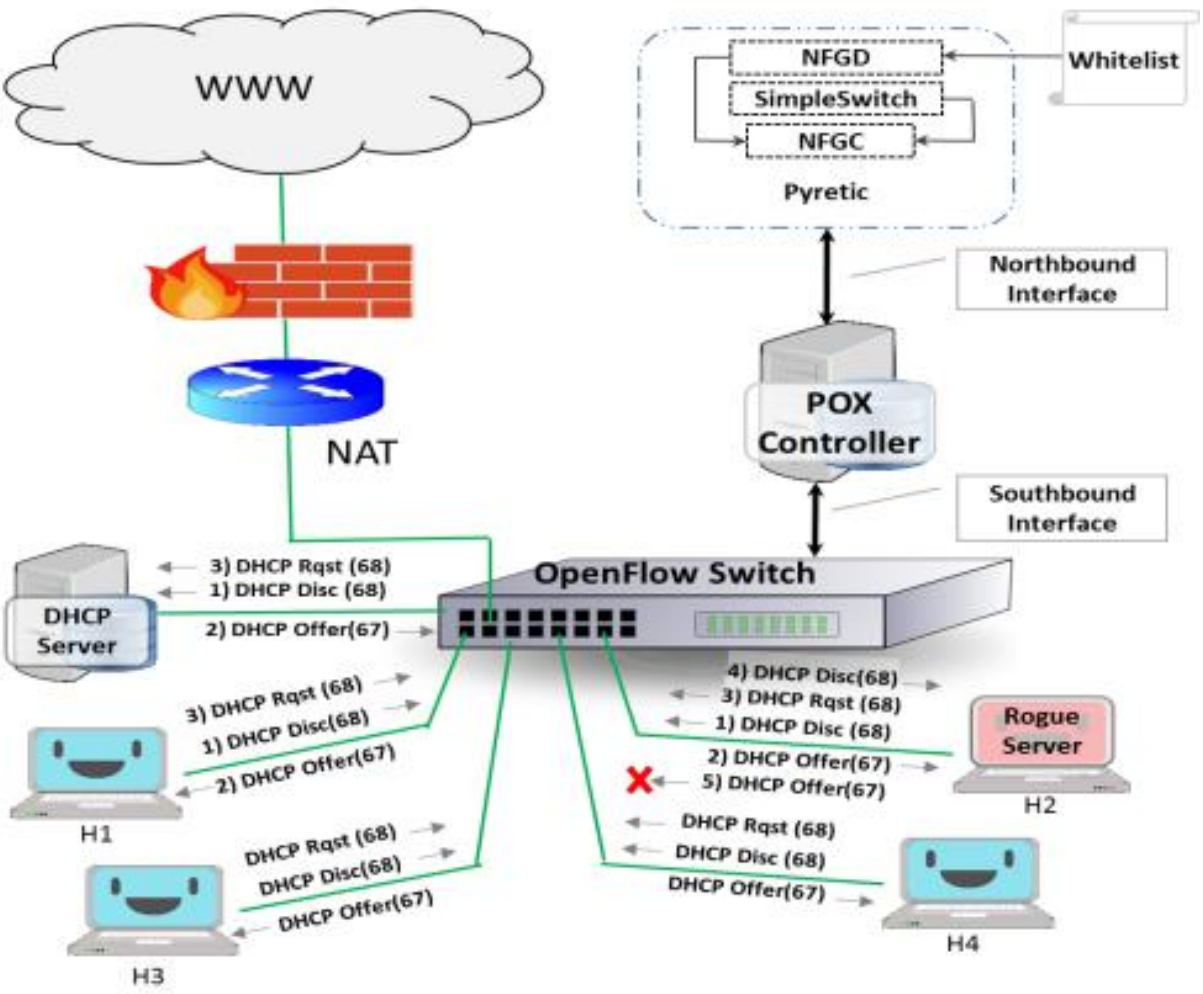
Samuel Jero ve arkadaşları [9] yaptıkları çalışmalarında YTA'daki açlık saldırısını iki adımda tanımlamaktadırlar. Birincisi; saldırgan kurban seçtiği bilgisayarın IP adresi ile MAC adresi arasındaki bağlantıyı kırıyor. Daha sonra bu bilgisayarın IP adresini DHCP_RELEASE mesajı ile DHCP havuzuna geri bıraktırmaya çalışıyor. Bu durumda Hostname ile DNS arasındaki bağlantı DNS kayıtlarının güncellenebilmesi için kırılmıyor. Saldırgan DHCP sunucusundan bıraktırdığı IP adresini almak için sahte MAC adresleri üzerinden sürekli IP almaya devam ediyor. DHCP çalışma mantığında IP adresleri dağıtılırken ilk önce havuzdaki kullanılmayan adresleri

bitirmeye çalışır. Saldırgan istediği IP'yi elde edince açlık saldırısını sonlandırıyor. Bu şekilde DHCP açlık saldırısını başarıyla tamamlamış oluyor. İkinci adımda ise saldırgan denetleyici cihazda bulunan akış tablosunu güncelleyerek kurban bilgisayarın MAC-PORT bilgisini kendi MAC-PORT bilgisi ile değiştiriyor.

### 1.1.2. Sahte (Rogue) DHCP Saldırıları

DHCP keşfedildiğinden beri sahte veya yetkisiz olarak adlandırılan DHCP saldırıları ağ yöneticileri için her zaman bir sorun olmuştur. Bu durumun başlıca ve en önemli nedeni ise DHCP istemcisinin IP isteğine cevap veren DHCP sunucusunun kimlik doğrulamasını yapamamasından kaynaklıdır. Ağ yöneticileri çok farklı savunma yöntemleri geliştirmiş olsalar bile planlı yapılan saldırılara karşı ilk adımda savunmasız kalmışlardır.

Dr. Jacob H. Cox Jr. ve arkadaşları [10] yaptıkları çalışmada YTA paradigması olan Network Flow Guard'ı (NFG) kullanmışlardır. NFG OpenFlow destekli anahtarlama cihazlarının başlık alanı eşleme özelliğini kullanarak yetkili olan DHCP sunucunun istemcilere IP adresleri verip vermediğini doğrulamak için, DHCP hizmetleriyle ilgili 67 portundan gelen trafiğini izler. NFG, izlediği bu trafiği ağ içerisinde bulunan sunucuların MAC adres tablolarının tutulduğu (Whitelist) beyaz listedeki MAC adresleri ile karşılaştırır. Böylece DHCP teklifinin yetkili bir sunucudan geldiğini doğrular. Aksi takdirde, OpenFlow anahtardaki akış kuralları paketi düşürmek üzere ayarlanır, böylece DHCP istemci yalnızca bilinen yetkili sunucudan DHCPOFFER mesajını alır. Şekil 1.2'de DHCP saldırı koruma modeli çalışma mantığı gösterilmiştir.




**Şekil 1.2.** DHCP saldırı koruma modeli çalışma mantığı [10]

Jhen-Li Wang ve Yen-Chung Chen yaptıkları çalışmada [11] sanal ağlarda DHCP güvenliğini YTA tabanlı savunma çözümleri geliştirerek işlemişlerdir. Sanal ağlarda çalıştırılan YTA uygulamalarının yönetim katmanı, kontrol katmanı ve altyapı katmanı için güvenlik önlemleri ele alınmıştır. DHCP güvenliği için kontrol katmanında bulunan YTA denetleyicide DHCP Tehdit Koruma Modülü oluşturulmuştur. Sanal makinalar birbirleriyle iletişim kurdukları zaman modül aradaki iletişim paketlerinde şüpheli veri olup olmadığını denetleyebilmektedir. Paket güvenli ise modül hiçbir şey yapmıyor. Paket güvenli değilse modül OpenFlow protokolü aracılığıyla sanal anahtar cihazına bir BLOCK komutu gönderiyor. Böylece altyapı katmanında bulunan sanal anahtar BLOCK komutunu aldığı zaman saldırıyı engellemek için şüpheli olan bağlantıyı engelliyor.

DHCP saldırı koruma modeli, sanal OpenFlow destekli anahtar üzerinde çalıştırılan bu model bulunduğu sanal ağda veri paketlerini akış tablolarına göre iletir. Sanal anahtar bilinmeyen bir

DHCP paketi aldığında YTA denetleyicisiyle güvenli bir kanal aracılığıyla iletişim kurar. Modül DHCP paketleri ile ilgili bilgileri Open Systems Interconnection (OSI) katmanlarından 2, 4 ve 7 katmanında derin öğrenme yöntemlerini kullanarak çıkarılan bilgileri ilgili DHCP saldırılarıyla analiz eder. Bu bilgiler, Ethernet başlığı, IP başlığı, TCP/UDP başlığı ve DHCP alanlarını içerir. Eldeki bilgiler ile planlı veya istemeyerek yapılan sahte DHCP saldırı paketlerinde bulunan ağ geçidi ve DNS bilgileri ile karşılaştırılır. Eşleşme durumunda herhangi bir tutarsızlık fark edildiğinde DHCP Tehdit Koruması Modülü BLOCK komutunu OpenFlow protokolüne göre gönderir. Sanal anahtar daha sonra akış tablosu girişini ekler ve kötü amaçlı bağlantıyı hemen engeller.

# 2. MATERYAL VE YÖNTEM

## 2.1. Yazılım Tanımlı Ağlar (YTA)

YTA'da OpenFlow protokolü desteğiyle yapılan yönetim katmanı ile veri katmanı ayırımı sayesinde bilgisayar ağları mimarisi farklı bir hal almıştır. OpenFlow destekli bir anahtarlama cihazında bir veya birden fazla akış tablosu bulunur ve bu tablolar içerisinde paket işleme kuralları bilgisi uzak bir kontrolcü tarafından ihtiyaca göre sürekli güncellenir. YTA ağ cihazlarının komple izlenmesi, ağa yeni katılan cihazların görülmesi, izin işlemleri kısaca bir ağı ayakta tutmak için gerekli olan birçok işlem kontrol katmanında bulunan kontrolcü ile mümkündür.

YTA, daha önce keşfedilmiş birçok yenilik gibi yine akademik bir çalışma olan OpenFlow araştırmalarında Stanford Üniversitesi bilgisayar laboratuvarlarında geliştirilmiş ve endüstriyel alanda hızlı bir şekilde birçok firma tarafından cihazları üretilip farklı alanlarda uygulanmaya başlanmıştır. Google, Microsoft, Yahoo, Deustch Telekom, Facebook, Verizon gibi şirketler tarafından asıl amaçları YTA'yı tanıtmak ve benimsetmek olan Open Networking Foundation (ONF), adında bir vakıf kurmuşlardır. Bu vakıf dünya üzerinde bulunan internet ve bilgisayar ağları ile ilgili alanlarda birçok hizmeti gönüllü olarak yürüten kişileri ve kurumları birleştirme özelliğine sahiptir. YTA ile ilgili akademik çalışmalar verimli şekilde sonuçlandıkça endüstriyel ve ticari alanlardaki kullanımları da genişlemektedir. Google dünya üzerindeki veri merkezleri arasındaki bağlantılarında, yaklaşık olarak 3 yıllık bir çalışmanın sonucunda YTA kullanmaya başlamıştır [12].

Böylece Google veri merkezleri arasındaki veri merkezleri arasındaki işlevsel etkinliğini artırarak finansal yatırımlarda avantajlı duruma geçmiş oldu. İnternet sağlayıcılardan ekipman üreticilerine, bulut bilişimden, finansal alanda hizmet veren birçok şirket ONF [13] ve OpenDaylight gibi YTA'yı destekleyen vakıflara katılım sağladılar. Böylece YTA endüstriyel alanda da hızlı bir ivme kazanmış oldu.

YTA, geleneksel bilgisayar ağlarını statik bir yapıdan dinamik bir yapıya kavuşturmak için OpenFlow protokolü çalışmaları kapsamında 2006 yılından itibaren geliştirilmektedir. Dinamik bilgisayar ağları üzerindeki ilk çalışmalar 1995 yılına kadar gitmekte ve ilk çalışmaları "Active Networks" başlığı altında toplandığı görülmektedir [14]. Geleneksel ağların yapısının çok karışık olması ve yönetilmesinin de zor olması nedeni ile şartlar, ağ yöneticilerini bilgisayar ağlarını yönetilebilir, gerektiğinde programlanabilir, sürekliliği olan bir ağ yapısı üzerinde düşündürmeye başlamıştır. Bu nedenle ilk çalışmaları Stanford Üniversitesi bilgisayar laboratuvarlarında başlamış olan OpenFlow protokolü kapsamında " Software Defined Network " Yazılım Tanımlı

Ağların geliştirilmesine başlanmış ve halen çalışmaları devam etmektedir. Geleneksel ağlarda kullanılan anahtarlama cihazları (switch), yönlendirme cihazları (router), yük dengeleyiciler (load balancing), arabirim cihazları, güvenlik duvarları (firewall) gibi önemli işlevlere sahip cihazların hepsi bir arada kullanılır. Bu durum bilgisayar ağlarındaki karmaşıklığı artırdığı gibi ağdaki yönetimi de oldukça zor duruma sokmaktadır.

YTA bu karmaşık durumu ortadan kaldırmak için bilgisayar ağlarına iki yeni karakteristik özellik kazandırmıştır. Birincisi, bilgisayar ağlarını kontrol katmanı ve veri katmanı olarak ikiye bölerek yeni bir topoloji kazandırmıştır. İkincisi, ağda bulunan birden fazla cihazın yönetimini tek bir yazılım ile kontrol katmanında toplamıştır. Kontrol katmanında, veri katmanı üzerindeki trafik akışı düzenlenirken, veri katmanında ise kontrol katmanından gelen komutlara göre akış tabloları ve diğer ağ ayarları güncellemeleri yapılır. Bu iki katman arasındaki iletişim OpenFlow Protokolü ile güvenli bir şekilde sağlanır. Veri katmanında bulunan ve OpenFlow desteği olan anahtarlama cihazları kontrol katmanındaki kontrolcü cihazı üzerindeki yazılım ara birimi sayesinde paket yönlendirmelerini akış tablolarına göre dropping (düşürme), forwarding (yönlendirme) veya flooding (taşma) işlemlerine göre sonuçlandırırlar. YTA kontrolcü tarafından ayarlaması yapılan bir anahtarlama cihazı, aynı zamanda yönlendirici, güvenlik duvarı, ağ adresi dönüştürücüsü, yük dengeleyici ve benzeri işlemlerde kullanılabilir.

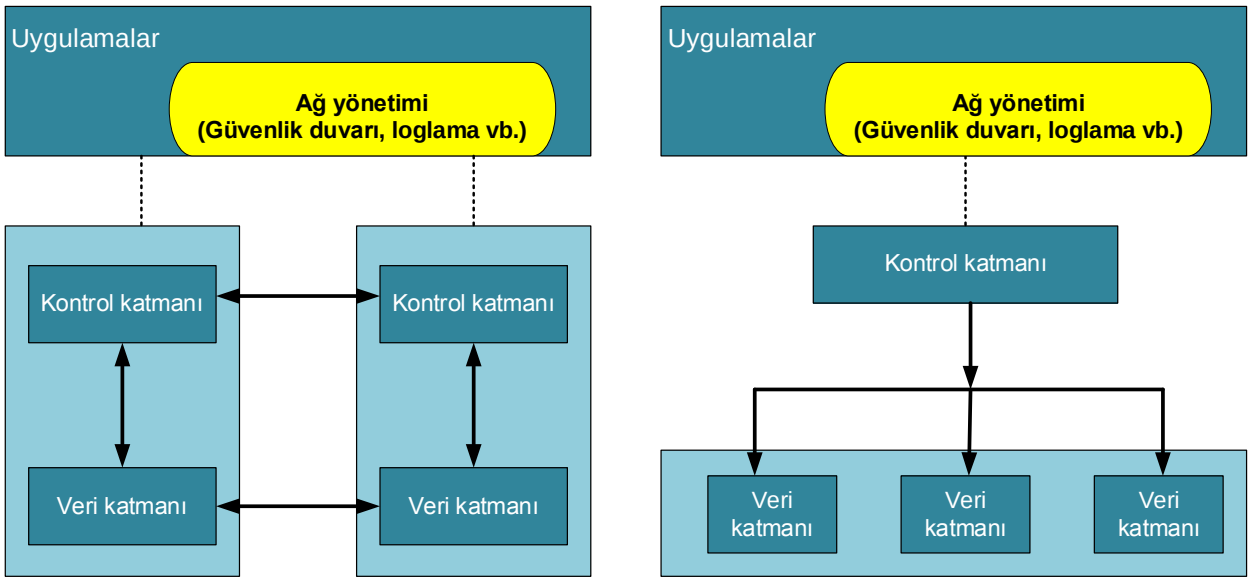
Son birkaç yılda YTA, bilgisayar ağları endüstrisinde hızlı bir ivme kazanmıştır. Ağ donanım cihazlarının önde gelen üreticilerinden HP, NEC, Pronto gibi firmalar ağ cihazlarında OpenFlow destekli üretime başlamışlardır [14]. Geliştirilen farklı kontrolcü platformları sayesinde programcılar bu platformları kullanarak dinamik erişim kontrolü, sunucu yük dengeleme, ağ sanallaştırma, enerji tasarruflu ağlar, sanal makine taşıması gibi imkânları bilgisayar ağları dünyasına kazandırmışlardır.

### 2.1.1. Kontrol Katmanı ve Veri Katmanının Birbirinden Ayrılması

Geleneksel ağ cihazlarında kontrol ve veri katmanının tek cihazda toplanması, bu ağların daha katı, değiştirilmesi zor ve statik bir yapıda kalmasına neden olmaktadır.

YTA bu zorluğun üstesinden gelmek ve ağ yönetimini kolaylaştırmak amacıyla geleneksel ağlarda tek cihazda bulunan kontrol ve veri katmanını birbirinden ayırmıştır [15].

Şekil 2.1 (a)'da geleneksel ağlarda kontrol katmanı ile veri katmanı tek bir cihazda bir arada olduğu görülürken Şekil 2.1 (b)'de ise YTA mimarisinde veri katmanı ile kontrol katmanının farklı katmanlarda olduğu görülmektedir.

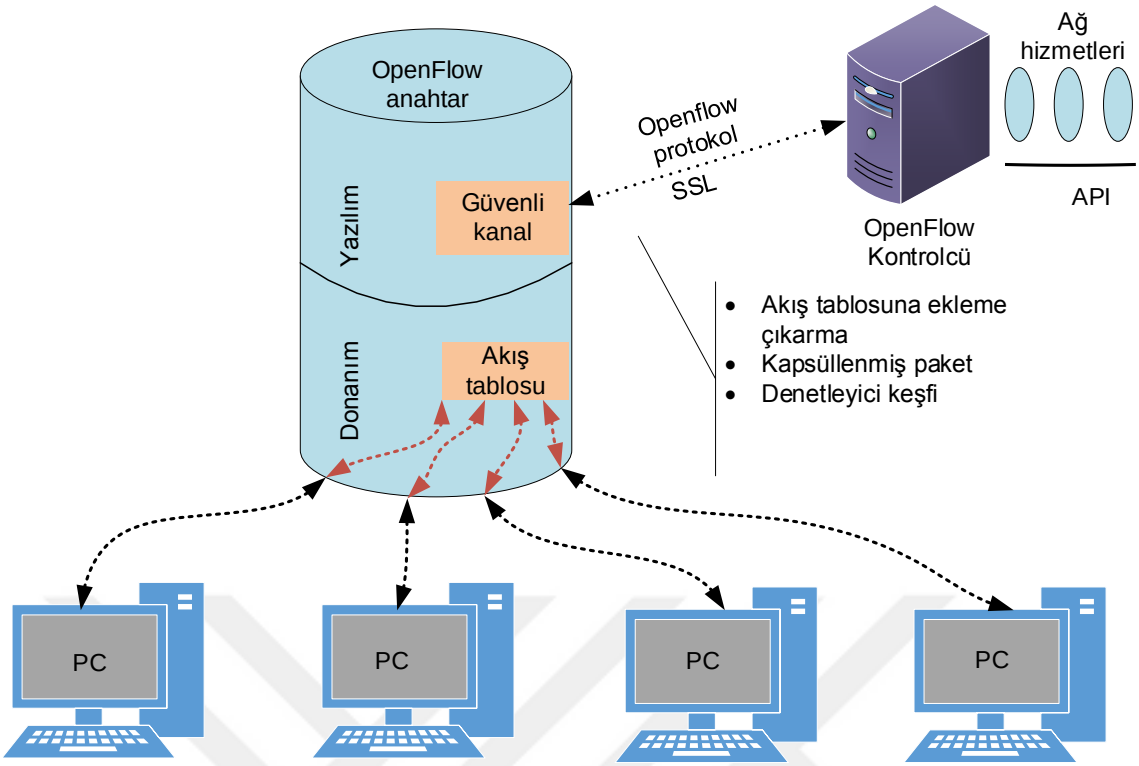



**Şekil 2.1.** Geleneksel (a) ve YTA (b) mimarilerinde kontrol ve veri katmanının ayrılması [15]

### 2.1.2. OpenFlow

OpenFlow [14,15] Open Networking Foundation [18] (ONF) tarafından YTA mimarisindeki kontrol katmanı ile veri katmanı arasındaki iletişimde ilk standart arabirim olarak tanımlanır [19]. Ağ yöneticilerinin arada herhangi bir yazılım kullanmasını gerektirmeden veri katmanındaki cihazlara erişimi ve bu cihazların kontrol edilmesini sağlar. Başka bir ifadeyle, OpenFlow, anahtarlar ve yönlendiriciler gibi ağ aygıtlarının hem fiziksel hem de sanal veri katmanına doğrudan erişilmesini ve güncellenebilmesini sağlar. Veri katmanında bulunan akış tablolarına erişim sağlayarak ağ trafiğinin nasıl yönlendirileceğini bildirir. Böylece, ağ yöneticileri akış tablolarını kısa sürede güncelleyebilir.

OpenFlow tabanlı anahtar modelleri OpenFlow ve OpenFlow-hybrid olmak üzere ikiye ayrılır. İlki yalnızca OpenFlow destekli cihazların işlemlerini yapabilirken, OpenFlow-hybrid ise hem OpenFlow destekli cihazların işlemlerini hem de OpenFlow olmayan yani normal ethernet anahtarlama işlemlerini yapabilir [20]. Şekil 2.2'de OpenFlow anahtar ile OpenFlow protokolünün çalışma mantığı gösterilmiştir.

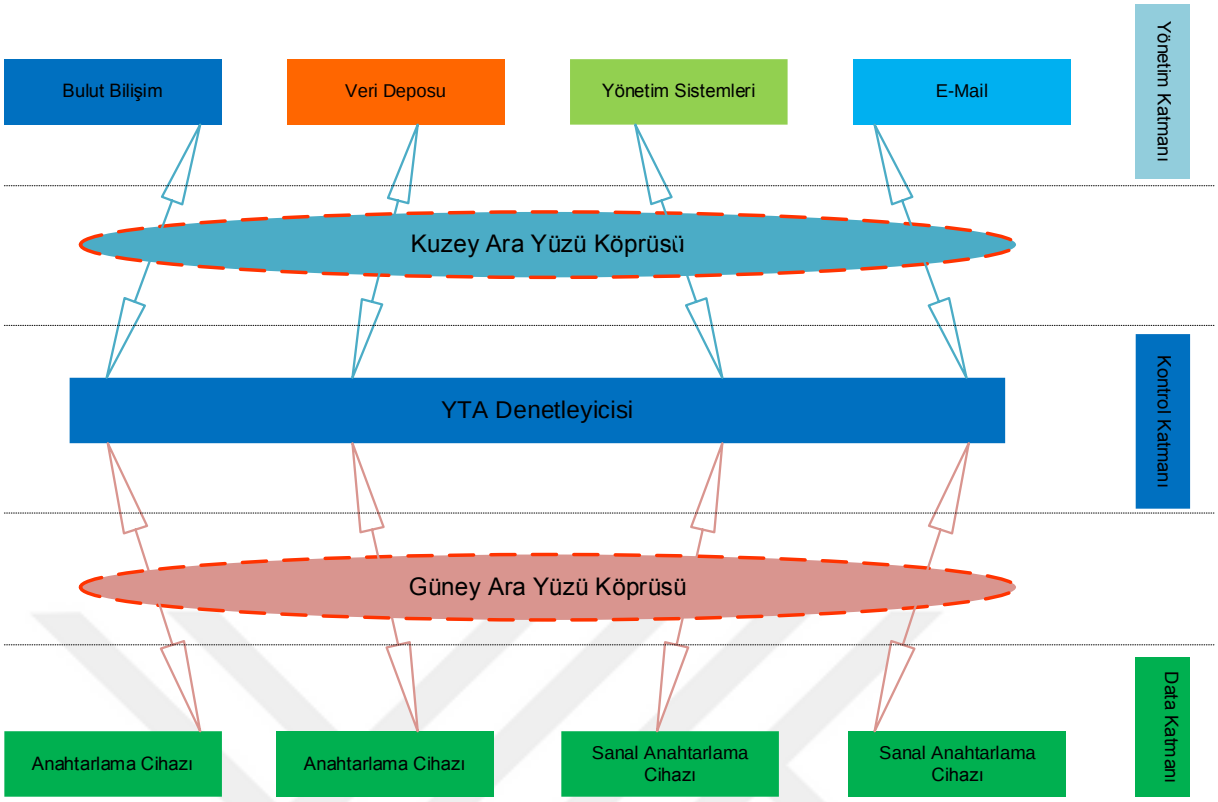



**Şekil 2.2.** Yazılım tanımlı ağlarda OpenFlow protokolü ve OpenFlow anahtarı [21]

OpenFlow protokolü alt yapı katmanı ile kontrol katmanı arasındaki standartlaştırılmış açık kaynak kodlu, en çok tercih edilen protokoldür [7].

OpenFlow ile birlikte Open vSwitch Database (OVSDB) [22], OpenState [23], OpFlex [24], POF [25], ForCES [26] gibi uygulamalarda YTA için güney ara yüz bağlantısı protokolleri arasında yer alır [27]. YTA'ın uygulama katmanındaki ağ uygulamalarında oluşturulan ağ politikaları kuzey ara yüzü köprüsü aracılığıyla denetleyiciye veya denetleyicilere (denetleyiciler yedekli olabilir veya ihtiyaca göre denetleyici sayısı birden fazla olabilir) gönderilir. Daha sonra bu politikalar veri katmanında bulunan yönlendirme cihazlarına güney ara yüzü köprüsü protokolleri ile iletilerek ağ trafiğinin istenildiği gibi yönetilmesi sağlanır.

Güney ara yüzü köprüsü protokollerinden biri olan OpenFlow protokolü ile ağ yönetimi ve güvenliği için ağın anlık durumu, topolojideki değişimler sonucu oluşabilecek güncellemeler, yük trafiğinin izlenmesi gibi ağ uygulamalarında kritik önem arz eden birçok özelliği gözlemlemek mümkündür. Kuzey ve güney arayüz köprüleri ve çalışma mantıkları Şekil 2.3'te gösterilmiştir.




**Şekil 2.3.** Kuzey - güney ara yüz köprüleri ve çalışma mantıkları [28]

OpenFlow kontrolcüsü, OpenFlow anahtarlar ve yönlendiriciler ile OpenFlow protokolü üzerinden, SSL (Secure Socket Layer) güvenli kanal aracılığıyla iletişim kurar. Günümüzde kullanılan bazı OpenFlow kontrolcüleri Çizelge 2.1'de gösterilmiştir.

**Çizelge 2.1.** Bazı OpenFlow kontrolcüleri [3]

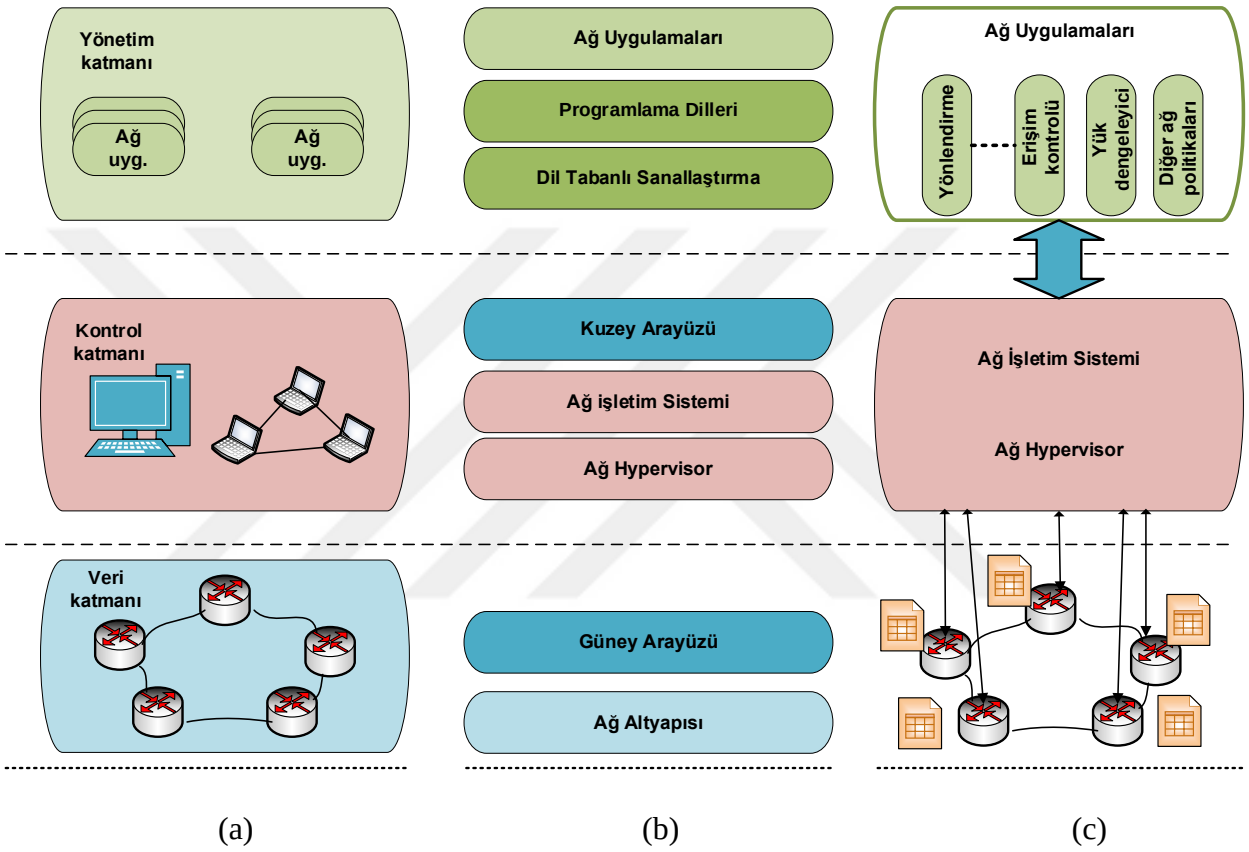
| **Kontrolcü** | **Programlama Dili** |
|---|---|
| POX, Ryu | Python |
| Beacon, Maestro, Floodlight | Java |
| NOX, SNAC, RouteFlow | C++ |
| Node.Flow | Java Script |
| Trema | Ruby ve C |

## 2.2. YTA Mimarisi

YTA ile ilgili ilk denemeler, Stanford Üniversitesi bilgisayar laboratuvarlarında OpenFlow protokolü çalışmaları kapsamında ortaya çıkmıştır [29]. Daha öncede belirtildiği gibi YTA,

geleneksel ağlarda fiziksel katmanda tek bir cihazda ve bir arada bulunan veri ünitesi ile kontrol ünitesini birbirinden ayırmış, her birine katman ismi vermiştir. Bu şekilde ağ mimarisine yeni bir paradigma kazandırmıştır.

YTA mimarisi esas anlamda 3 ana katmandan oluşur. Bu katmanlar; yönetim katmanı, kontrol katmanı ve veri katmanı olarak Şekil 2.4 (a)'da gösterilerek YTA mimarisinin genel bir şekli çizilmiştir.




**Şekil 2.4.** (a) YTA katman yapısı, (b) YTA katman üniteleri, (c) Sistem tasarım mimarisi [30]

### 2.2.1. Veri Katmanı

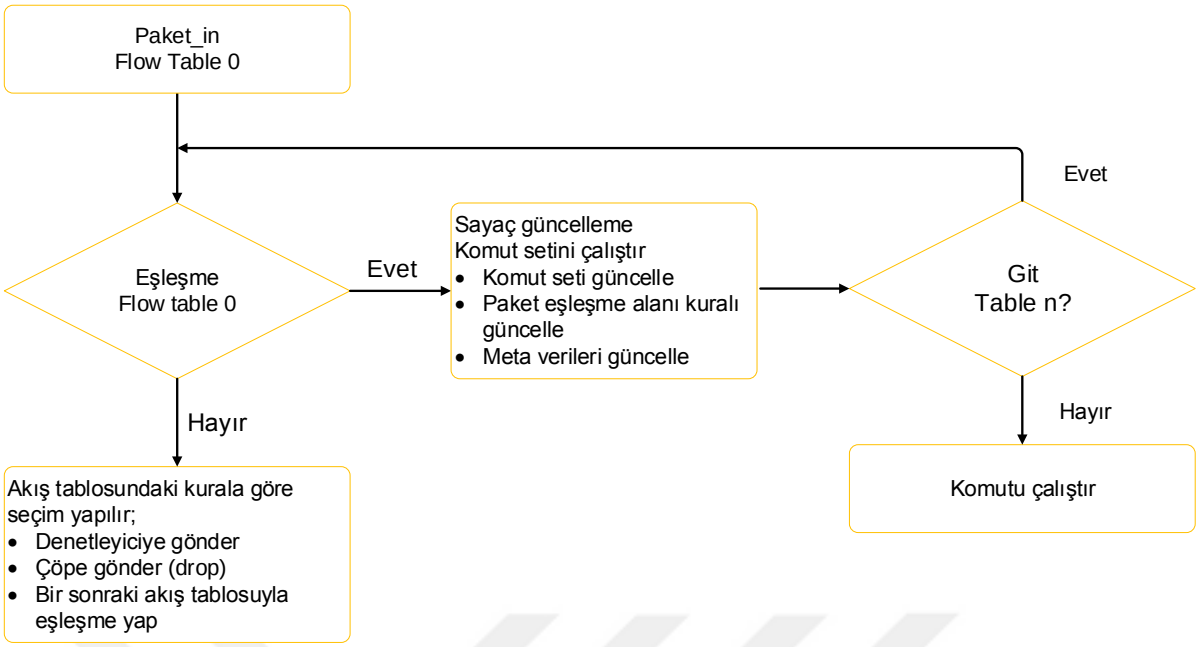
YTA'daki veri katmanı geleneksel ağlardaki gibi fiziksel ağ cihazlarından oluşur. İki yapı arasındaki temel fark geleneksel ağlarda her ağ cihazında gömülü halde yani bütünleşik ve her bir cihazda ayrı ayrı bulunan işletim sisteminin yazılım tanımlı ağ cihazlarında olmaması bunun yerine kontrol ünitesi biriminde olmasıdır**.**

Geleneksel ağlarda paket yönlendirme işlemi IP yönlendirme algoritmalarına dayanırken, YTA ağlarında ise kontrolcü denetiminde bulunan yönlendiricilerin üzerindeki akış tablolarına göre belirlenir. Yönlendirme tabloları "match-plus-action" davranma yeteneğine sahiptirler [31]. Match, çoklu başlık alanlarına sahip paketlerde ağ, bağlantı ve aktarım katmanlarına karşılık gelebilir. Action, başlık alanlarını yeniden yazarak, datagramı iletmek, bırakmak, kopyalamak eylemlerini gerçekleştirir.

#### 2.2.1.1. Ağ Altyapısı

YTA altyapısı, geleneksel bir ağda olduğu gibi bir grup ağ ekipmanından (anahtarlama cihazları, yönlendiriciler ve yük dengeleyiciler ) oluşur. Buradaki temel fark, geleneksel ağlardaki fiziksel cihazlardan farklı olarak artık otonom kararlar almak için gömülü kontrol veya yazılım olmadan basit yönlendirme öğelerinin olmasıdır. Ağın beyni, veri düzlemi cihazlarından, farklı bir yapıda mantıksal olarak merkezi bir kontrol sistemine, yani ağ işletim sistemine sahiptir. Daha da önemlisi, bu yeni ağlar birbirinden ayrı olan veri katmanı ve kontrol katmanı cihazları arasında konfigürasyon ve iletişim uyumluluğunu sağlamak ve birlikte çalışabilirliği sürdürmek için önemli bir yaklaşım olan, OpenFlow üzerine inşa edilmiştir. Başka bir deyişle, OpenFlow kontrolcü yönlendirici olarak adlandırdığımız anahtarlayıcı ve yönlendirici gibi (çok çeşitli, tescilli ve kapalı arabirimlere sahip geleneksel ağlarda zor olan ve dağınık yapıdaki) iletme aygıtlarını dinamik olarak programlamasına olanak tanır.

OpenFlow; kendisini destekleyen yönlendirme cihazlarının içinde, akış tabloları dizisi boyunca, paketlerin nasıl ele alınacağını tanımlar. Veri katmanındaki yönlendirme cihazına yeni bir paket geldiğinde, paket başlığında bulunan bilgiler akış tablolarındaki bilgiler ile karşılaştırılır. Bu tablolardan birinde bir adresle eşleşirse paket o bilgi doğrultusunda yönlendirme yapılır. Bu paket için hiçbir kural bulunmadığında ise paket ya denetleyici cihazına yönlendirilir veya varsayılan bir kural yoksa paket çöpe atılır. Şekil 2.5'te OpenFlow cihazların akış tablolarına göre paket yönlendirme işlemleri gösterilmiştir.




**Şekil 2.5.** OpenFlow akış tablolarında paket yönlendirme işlemleri [32]

Bununla birlikte, YTA'da genel durum, anahtarlama cihazına, gelen paketin (packet_in olarak adlandırılır) denetleyiciye mi yoksa anahtarın normal OpenFlow olmayan portuna mı göndermesini belirten aynı zamanda varsayılan bir kural devreye girer. Kuralların işlem sırası, tabloların sıra numarasını ve akış tablosundaki satır sıra numarasını izler.

Gelen yeni bir paketin işleme sırası;

- Paketi giden bağlantı noktalarına yönlendirmek (Gideceği port belli ise)
- Paketi kapsüllemek ve kontrolöre iletmek (Gideceği port belli değilse)
- Düşürmek (Çöp kutusuna göndermek)
- Normal işleme portuna (OpenFlow olmayan) göndermek
- Bir sonraki akış tablosuna veya en son OpenFlow protokolünde tanıtılan grup veya ölçüm tabloları gibi özel tablolara göndermek.

#### 2.2.1.2. Güney Arayüzü

Güney-bağlı ara yüzler veya güney-bağlı API'ler kontrolör ve yönlendirme elemanları arasındaki bağlantı köprüleri olup, kontrolör ve veri düzlemi arasındaki işlevselliği artırmak ve bu iki bölüm arasındaki iletişimi sağlamak için çok önemli bir araçtır. Bu yapı sayesinde, Uygulama kısmındaki API'ler hala temeldeki fiziksel veya sanal altyapının yönlendirme öğelerine sıkı sıkıya bağlıdır.

OpenFlow, YTA için en yaygın kabul gören ve bu anlamda kontrolör ve yönlendirme elemanları arasında konuşlandırılan açık kaynak kodlu güney sınırı standardıdır. OpenFlow destekli yönlendirme cihazlarının uygulanması ve veri katmanı (yönlendirme katmanı) ile kontrol katmanı cihazları arasındaki iletişim kanalı için ortak bir protokol sağlar.

OpenFlow protokolü, ağ işletim sistemleri için üç adet bilgi kaynağı sağlar. Birincisi, bir bağlantı veya bağlantı noktası değişikliği bilgisi güncellendiğinde bu durum olay tabanlı mesajlar olarak yeni aygıtların bilgisi şeklinde denetleyiciye (kontrolöre) gönderilir. Böylece ağ fiziksel yapısı her zaman güncel kalmış olur. İkincisi belirli zaman aralıklarında ( bu zaman aralıkları ağ yöneticisi tarafından değiştirilebilir ) akış istatistikleri, yönlendirme cihazları tarafından oluşturulur ve bu istatistikler kontrolör cihazına gönderilir. Böylece ağda oluşabilecek anormal durumlar gözlemlenebilir. Üçüncü ve son olarak, yönlendirme cihazına gelen bazı paketlerde yönlendirme bilgisi olmadığında bu paketlere ne yapılacağı ile ilgili kontrolörden gelen bilgilere göre akış tabloları güncellenir. Böylece ağda yığılma söz konusu olmaz.

OpenFlow, YTA için güney bağlantıda (veri düzlemi ile denetleyici arasında) en çok tercih edilen bağlantı protokolü olsa bile bu işi yapan tek ara yüz değildir. Buna benzer olarak ForCES, OVSDB (Open vSwitch Database), POF (Protocol Oblivious Forwarding), OpFlex, OpenState, ROFL (Routing On Flat Labels) gibi uygulamalarda mevcuttur.

### 2.2.2. Kontrol Katmanı

Kontrol katmanında bulunan aynı zamanda ağın beyni olarak adlandırılan denetleyici (kontrolcü), YTA mimarisinde önemli bir yere sahiptir. Günümüzde birçok YTA denetleyicisi mevcuttur. Bunlar, geliştirildikleri veya geliştirilecekleri yazılım diline, elde ettikleri performansına, programcının denetleyiciye uygulama yazılımı geliştirebilmek için öğrenilmesi gereken zamana, desteklediği güney ara yüz tiplerine, hangi amaca yönelik olduğuna, dağıtılmış kontrol desteğine gibi birçok faktöre göre değişim göstermektedir [27]. Kontrol katmanında üç temel başlık üzerinde durulacaktır. Bunlar, Ağ Sanallaştırma, Ağ İşletim Sistemi ve Kuzey-bağlantı Köprüsüdür.

#### 2.2.2.1. Ağ Hypervisor

Hypervisorler, farklı sanal makinelerin aynı donanım kaynaklarını paylaşmasını sağlar. Günümüzde sanallaştırma teknolojilerinin çok önemli özelliklerinden biri, sanal makinelerin bir fiziksel sunucudan diğerine kolayca geçirilebilmesi ve isteğe bağlı olarak yeniden

oluşturulabilmesi veya silinebilmesi, esnek ve kolay yönetim ile hizmetlerin sağlıklı bir şekilde yürütülebilmesidir. Bilgi işlem ve depolama öğelerinin sanallaştırılmasında büyük ilerlemeler kaydedilmiştir. YTA mimarisinde bant genişliği, ağ topolojisi, ağ trafiği, CPU ve yönlendirme tabloları gibi çok önemli mantıksal işlevler başka ağ ortamlarına da uygulanabilmektedir [30].

YTA geleceğin internet ağlarında yeni hizmetler ve protokoller oluşturarak gelecek vaat eden bir ağ mimarisi amaçlamaktadır. Ağı "programlayabilme", uygulamaların ağın içinde yenilikçi yeni hizmetler oluşturmasını sağlar. Bununla birlikte, mevcut YTA programlanabilirliği, uygulanması ve konuşlandırılmasını engelleyebilecek olumsuzluklarla karşı karşıya gelmektedir. YTA mimarisinde, tam kontrol sağlamak için düşük seviyeli API'ler sunulmaktadır. Bu durumda, gelecekteki ağ uygulamalarının ihtiyaç duyduğu yüksek düzeyli karmaşık hizmetleri uygulamak zor bir görev haline gelmektedir. Diğer bir önemli sorun ise, geliştirilen YTA teknolojilerinin birbiriyle çok az ortak noktası bulunmakta ve bu durum çok sayıda YTA teknolojisi arasındaki bütünlüğü bozmaktadır. Ağ hypervisorü, karmaşık olan YTA teknolojilerini büyük ölçüde kolaylaştıran yüksek düzey soyutlamalar ve API'ler sunar. Ayrıca, ağ hypervisorü, çeşitli YTA geliştiricilerini tek bir arayüzle çalıştırabilir. Böylece uygulamalar YTA geliştiricileri arasındaki farkları görmeye ve uçtan uca bir bütünlük oluşturmaya yarar [33].

#### 2.2.2.2. Ağ İşletim Sistemi

Geleneksel ağlardaki cihazlarda bulunan işletim sistemleri, veri katmanındaki aygıtlarla bütünleşik şekildedir ve programlanabilir yetenekleri oldukça sınırlıdır. Bir ağda bulunan işletim sistemi veya ağın beyni olarak da adlandırılan NOS (Network Operating System) üzerinde yapılabilecek işlevsellikler ve programlanabilirlik kapasitesi, üretkenliği artırmak için önemli olanaklardır. Sistem ve uygulama geliştiricilerinin işlerini kolaylaştırır.

Geleneksel ağlarda, çoğunlukla kapalı (tescilli) ağ işletim sistemleri kullanılmıştır. Bu durum ağ cihazlarını, üretici firmalara özgü karakteristikleri destekleyen ve kullanıcıyı üreticiye bağımlı hale getiren bir sistemin oluşmasına neden olmuştur. Şeffaf bir şekilde ortak işlevler sağlayan işletim sistemleri fikri hala geleneksel ağlarda mevcut değildir. Yönlendirme protokolü tasarımcılarının, ağ sorunlarını çözerken, karmaşık dağıtılmış algoritmalarla uğraşmaları gerekmektedir. Bu nedenle ağ uzmanları aynı sorunlarla yeniden uğraşmak zorunda kalıyorlar.

Ağ işletim sistemleri, ağ mimarisi geliştiricilerine temel hizmetler ve ortak uygulama arabirimlerini sağlamaktır. Bu arabirimler içerisinde, ağ durumu analizi, ağ topolojisi bilgileri, cihaz keşfi ve ağ yapılandırmasının dağıtımı gibi genel işlevsellik uygulamalarına benzer hizmetler başlıca olarak sıralanabilir.

Geleneksel ağların tersine YTA mimarisi ağ işletim sistemini (NOS) merkezi bir denetleyici olarak tanımlamış ve ağın tüm yönlendirme cihazlarını yöneten bir beyin olarak nitelendirmiştir. Çok sayıda veri katmanı elemanına sahip bir ağ için tek bir denetleyici yeterli olmayabilir. İhtiyaca göre ağ içerisinde birden fazla denetleyici de konuşlandırılabilir. YTA mimarisinde kullanılan denetleyiciler birçok karmaşık işlemi kısa sürede yapabilecek kapasiteye göre geliştirilmektedirler. YTA denetleyicilerinden biri olan Beacon, Amazon gibi bulut sağlayıcılarının büyük boyutlu bilgi işlem düğümlerini kullanarak saniyede 12 milyondan fazla akışla başa çıkabilir [34]. Denetleyiciler daha çok veri merkezleri, uluslararası banka ağları, büyük kampüs ağlarında, bulut altyapılarında ve operatör sınıfı ağları gibi büyük yük altındaki ve belirli ortamdaki ağları kotarmak için geliştirilirler.

Bir ağda en az iki tane denetleyicinin olması demek hata toleransı oranın artabileceği anlamına gelir. Denetleyicilerden birisi başarısız olduğunda, başka bir komşu denetleyici başarısız düğümün görevlerini ve cihazlarını devralabilir.

Hatta ikiden fazla kontrolörün kullanıldığı ağlarda, kontrol katmanı mevcudiyeti kritikse, başarımı artırmak için daha yüksek bir mevcudiyet derecesi elde etmek veya daha fazla cihazı desteklemek için bir kontrol birimi kullanılabilir. Sonuç itibariyle, dağıtılmış bir kontrolör ünitesi, kontrol katmanının esnekliğini, ölçeklenebilirliğini geliştirebilir. Böylece ağ topolojisinden yapısından kaynaklı sorunların etkisini azaltabilir. Ağ işletim sistemleri aynı zamanda paketleri en kısa yola iletme, güvenlik mekanizmalarını, istatistik, veri ile kontrol katmanı arasındaki bildirimleri yönetimine yardımcı olurlar. Örnek vermek gerekirse, bildirim yöneticisi olay tabanlı mesajları olarak bilinen alarm bildirimleri, güvenlik alarmları ve topoloji durum güncellemelerini alabilmeli, işleyebilmeli ve iletebilmelidir [34].

Tezimizin temel konusu olan güvenlik mekanizmaları konusu geleneksel ağlarda da olduğu gibi çok önemlidir, çünkü hizmetler ve uygulamalar arasında temel izolasyon ve güvenlik uygulaması sağlamak için kritik bileşenlere ihtiyaç vardır. Tez çalışmasında üzerinde çalıştığımız güvelik konusu kontrolör güvenliğinden ziyade yönlendirme cihazların düzlemindeki güvenlik politikaları ile ilgilidir.

#### 2.2.2.3. Kuzey Arayüzü

Kuzey arayüzü YTA mimarisinin en üstte kalan uygulama katmanı ile kontrol katmanı arasındaki iletişimi sağlayan uygulama odaklı soyut bir kavramdır. RFC7426, farklı katmanlardaki varlıklar arasında kuzey/güney iletişimi gibi bağlantıları sağlayan protokol kurallarını düzenler. Kontrol katmanı üstünde bulunan yönetim düzlemindeki uygulamalarda belirlenen ağ

politikalarını kontrol katmanındaki cihazların anlayabileceği formata çevirerek denetleyicilere iletir. Bu arayüz aynı zamanda güney arayüzündeki donanımsal cihazlardan farklı olarak yazılımsal yani soyutsal bir uygulamadır [30].

Şu anda birbirinden farklı kuzeye bağlı API'leri olan yirmiden fazla YTA denetleyicisi bulunmaktadır. Bu denetleyicilerin her biri kendilerine özgü kuzey API'leri sunmaktadırlar. Bunlar aynı zamanda üç farklı kategoride sınıflandırılabilir. RESTful API'ler, Ad-Hoc API'ler ve kuzeydeki arayüzler için kullanılan programlama dilleri. [35].

### 2.2.3. Yönetim Katmanı

Yönetim katmanı, ağ trafiğini yönetmek için ağ yöneticisi tarafından belirlenen ağ politikalarının yazıldığı kısımdır. Ağ cihazlarının izlenmesi, yapılandırılması, bakımı ve güncel durumlarıyla ilgili kararların alındığı katmandır. Bu katmanda, yönlendirme katmanı cihazlarını konfigüre etmek için kontrol katmanında kullanılan uygulamalardan farklı olarak üst düzey uygulama programları kullanılır. Bu işlemler kontrol katmanında yapılamayan ancak üst düzey bir programlama altyapısı gerektiğinde ve kontrol katmanından daha ileri seviyede bir yaklaşımla yapılır. Belirlenen ağ politikaları buradan denetleyicilere gönderilir. Web ara yüzü kullanılarak yapılan işlemlerde Hyper Text Transfer Protokol (HTTP), Secure Shell (SSH), Simple Network Management Protokol (SNMP) kullanılır. Diego Kreutz ve arkadaşları [34] yaptıkları çalışmalarında yönetim katmanını üç başlık altında toplamışlardır.

#### 2.2.3.1. Dil Tabanlı Sanallaştırma

Sanallaştırma teknolojilerinin önemli amaçlarından biri, modüler bir yapıyı desteklemesi. Bir diğer önemli amaçlarından biri ise güvenlik. Örneğin, sanallaştırma teknikleri tek bir fiziksel altyapının farklı görünümlerine izin verebilir. Yani, sanal olan kapasiteli bir yönlendirme cihazı altındaki birkaç yönlendirme cihazının bir birleşimini temsil edebilir. Bu durumda, yönlendirme kurallarının yüklenmesi gereken anahtarların sırasını düşünmek zorunda olmadıkları için uygulama geliştiricilerin görevi basit bir şekilde kolaylaşır. Böylece geliştiriciler ağı basit bir büyük anahtar olarak görür. Bu tür bir soyutlama, gelişmiş güvenlikle ilgili hizmetler gibi karmaşık ağ uygulamalarının geliştirilmesini ve dağıtımını önemli ölçüde basitleştirir.

Piretik, bu tür üst düzey ağ topolojisi soyutlaması sunan bir programlama diline önemli bir örnektir. Ağ nesnelerini tanıtarak bu soyutlama kavramını birleştirir. Bu nesneler soyut bir ağ

topolojisi ve bu ağa uygulanan politika kümelerinden oluşur. Ağ nesneleri aynı anda bilgileri gizler ve gerekli hizmetleri sunar.

#### 2.2.3.2. Programlama Dilleri

Programlama dilleri gün geçtikçe çoğalmakta ve farklı alanlarda kullanılabilmekte aynı zamanda hızlı bir şekilde geliştirmektedirler. Bu diller zamanla, x86 mimarileri gibi düşük seviyeli donanıma özgü makine dillerinden Java ve Python gibi üst düzey ve güçlü programlama dillerine dönüşmüştür. Dahası taşınabilir ve yeniden kullanılabilir kodlara yönelik gelişmeler, bilgisayar teknolojilerinde önemli gelişmelere ön ayak olmuştur. Sadece OpenFlow gibi sınırlı programlanabilirlik hizmeti veren, esasen, yönlendirme cihazlarının davranışını taklit ederek, geliştiricileri problem çözmektense düşük seviyeli detaylarda çok fazla zaman harcamaya zorlar. Üzerine herhangi bir ekleme yapılmamış veya geliştirilmemiş ham OpenFlow programları, birbiriyle çakışan kurallar, kuralların öncelik sıralaması ve akış kurallarından kaynaklanan veri düzlemi tutarsızlıkları gibi donanım davranışı tutarsızlıklarıyla boğuşmak zorunda kalırlar. Bu düşük seviyeli dillerin kullanılması, yazılımı yeniden kullanmayı, modüler ve kapsamlı kod oluşturmayı zorlaştırmakta ve hataya daha yatkın bir geliştirme sürecine yol açmaktadır.

Üst düzey programlama dilleri ile veri katmanı cihazlarının programlanması ve yönetimi, ağ topolojisinin yönetimi, yazılım geliştiriciler için üretken ve sorun odaklı ortamlar sağlayarak geliştirme ve yeniliğin hızlandırılması, kontrol düzleminde yazılımın modüler ve yeniden kullanılabilirliğini teşvik etmek ve en önemlisi YTA ağ sanallaştırmasını geliştirilmesi amaçlanmaktadır.

Programlama dili soyutlamalarının sağladığı bir diğer önemli destek ise sanal ağ topolojileri için program oluşturma ve yazma yeteneğidir. Bu kavram, ağların uygulama geliştiricileri için hem verileri hem de belirli işlevleri soyutladığı, veri yapıları ve bunların yönetimi hakkında endişelenmeden belirli bir sorunu çözmeye odaklanmayı sağlayabilir. Bu tez çalışmasında Python dili ile yazılmış açık kaynak kodlu, Mininet ağ benzetim programı kullanılmıştır. Mininet ile tam bir YTA mimarisi oluşturabilir, burada yazılan bütün kodlar gerçek ortamdaki cihazlar üzerine aktarılabilir.

#### 2.2.3.3. Ağ Uygulamaları

Ağ uygulamaları YTA mimarisinde en üst katmanda bulunur ve asıl kontrol katmanındaki denetleyici burada bulunan uygulamalar ile programlanır. Ağ uygulamaları “ağ beyinleri” olarak

adlandırılabilir [34]. Ağ yöneticisi uygulamalar sayesinde veri düzlemindeki cihazlarını kontrol edebilir, bu cihazlar üzerindeki akış tablolarını güncelleyebilir, ağ topolojisini görüntüleyebilir kısacası ağ yöneticisinin yeteneğine göre ağda görmek istediği tüm hizmetleri bu ağ uygulamaları sayesinde yapabilir.

Ağ uygulamaları ile ağda bulunan tüm cihazların güvenlik uygulamaları, yük dengelemeleri (load balancing), ağ sanallaştırma işlemleri, uçtan uca QoS uygulamaları ve benzeri birçok işlem yapılabilir [30]. Bunların yanı sıra ağ topolojisi keşfi, ağ provizyonu, yol rezervasyonu vb. uygulamalarda mümkündür. Ağ uygulamalarındaki gelişim devam ettikçe YTA kullanımı da aynı oranda artacaktır.

### 2.3. Mininet

YTA mimarisi bilgisayar ağları içerisinde henüz istenen seviyede yer edinemediğinden dolayı YTA destekleyen yönlendiriciler ve anahtarlar gibi pek çok cihaz yeterince bulunmamaktadır. Yenilerini elde etmek ise çok pahalıya mal olmaktadır. Ağ uzmanları Mininet [36] gibi sanal uygulama programları ile YTA alanındaki çalışmalarını sürdürmekte ve yaptıkları çalışmaları düşük bir maliyetle pratik olarak test edebilmek için kullanmaktadırlar. Mininet ortamında yazılan bütün kodlar gerçek ortamdaki kodlar ile aynıdır.

Mininet, araştırmacıların veya ağ programcılarının basit bir şekilde YTA prototipini oluşturmalarını sağlayan özelliklere sahiptir. Aynı zamanda etkileşimli çalışabilen, ağı özelleştirebilme ve paylaşma özellikleri ile birlikte açık kaynak kodlu olup tamamen ücretsizdir [37]. Mininet ile yapılan ağ uygulamalarını gerçek ortamda uygulamak çok fazla zaman kaybına ve yüksek maliyetlere neden olmaktadır. Linux tabanlı olan program sanal ortam uygulamalarında da çalıştırılabilmekte ve aynı zamanda Python API'lerini desteklemektedir [38].

### 2.4. POX Kontrolcüsü

POX [39], Python programlama dilinde yazılmış, açık kaynak kodlu YTA kontrolcülerinden biridir. POX varsayılan paketinde bulunan bileşenlerinden dolayı YTA'da hemen uygulanabilir. Bu şekilde kullanım kolaylığı sağlar. Aynı zamanda ağın beyni olarak da nitelendirilir.

Kontrolcü YTA'da güney arayüzü protokolünü kullanarak altyapı katmanı ile kuzey arayüzü protokolünü kullanarak yönetim katmanı ile iletişim sağlar.

### 2.5. Wireshark

Bilgisayar ağlarında veri paketlerinin aktarımı TCP/IP protokolleriyle gerçekleşmektedir. En büyük ağ olarak kabul edilen internet TCP/IP protokollerini kullanarak veri aktarımını sağlar. Wireshark TCP/IP protokollerinin kullanıldığı ağlarda trafik analizi sağlar [40]. Ücretsiz ve açık kaynak kodlu paket analiz programıdır.

Birçok ağ geliştiricisi, ağ uzmanı Wireshark uygulamasını destekleyip gönüllü olarak geliştirilmesine yardımcı olmaktadır. Wireshark 1998 yılında Gerald Combs tarafından başlatılan bir projenin devamı niteliğindedir [41]. Wireshark ağ sorunları analizi, protokol geliştirme ve eğitim gibi farklı birçok amaç için kullanılabilir. Windows, Linux, Unix gibi farklı platformlarda çalıştırılabilir.

Wireshark komut satırında veya kullanıcı arayüz ile kullanılabilir. Veri paketlerini yakalamak için pcap filtresini kullanır.

Wireshark ile yapabileceğiniz bazı işlemler şunlardır;

- Kablolu veya kablosuz canlı ağ trafiğini, Ethernet 802.11 ve Point-to-Point Protokol (PPP) dâhil yakalayabilir veya daha önceden yakalanmış ağ trafiğini analiz edebilirsiniz.
- İstediğiniz herhangi bir filtre dâhilinde (IP, MAC, DNS vb) filtreleme yapabilir veya sonuçları analiz edebilirsiniz.
- Ağ üzerinde yapılan Voice over IP (VoIP) aramalarını filtreleyebilir veya analiz edebilirsiniz.
- TCP/IP ağlarının nasıl çalıştığını gözlemleyebilir ve ağ sorunlarını çözebilirsiniz. [42]

## 2.6. VirtualBox

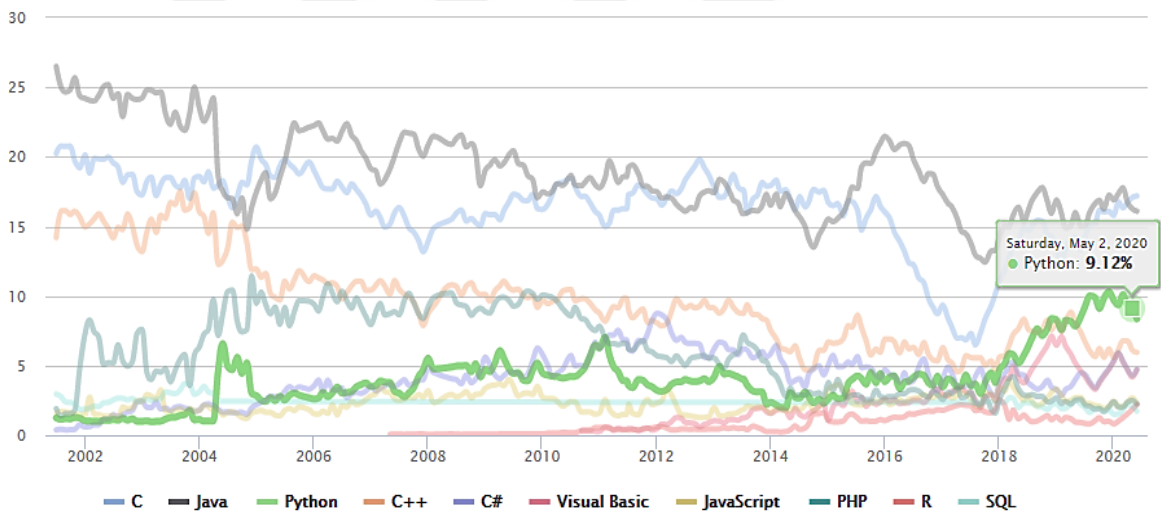
VirtualBox [43], bir bilgisayardaki donanım cihazlarını (hard disk, ethernet kartı, ekran kartı, işlemci vb.) taklit ederek işletim sistemlerini gerçek işletim sistemi üzerinde sanal olarak kullanmaya imkân tanıyan açık kaynak kodlu sanallaştırma yazılımıdır [44]. Bilgisayar içinde bilgisayar kullanımı olarak ifade edilebilir. VirtualBox, Linux, Windows, Macintosh ve Solaris işletim sistemlerinde çalışır. Sanal olarak çalıştırılan bilgisayara konuk bilgisayar, konuk bilgisyar üzerindeki işletim sistemine ise konuk işletim sistemi adı verilir. Sanal bilgisayarın üzerinde çalıştığı gerçek bilgisayara host, işletim sistemine ise host işletim sistemi denir. VirtualBox çok sayıda (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10 ,Windows / DOS, Linux , Solaris, OpenSolaris, OS / 2 ve OpenBSD) konuk işletim sistemini destekler [45].

Popüler olarak bilinen sanallaştırma yazılımları VirtualBox ve VMware Workstation [28]'dır. VirtualBox ücretsiz olduğundan dolayı daha çok tercih edilmektedir.

Konuk bilgisayar ile yapabileceğiniz tüm işlemler tamamen başka bir bilgisayarda yapılıyormuş gibi ev sahibi bilgisayarları etkilemezler. Bundan dolayıdır ki malware analizleri sanal makinalarda yapılır. Sanal makinalar ne kadar da ayrı fiziksel makinalar olarak görünsede üzerlerinde bulunan hostlarla ihtiyaç halinde dosya paylaşımı yapabilir ve ağ kartlarını kullanarak internete erişim sağlayabilirler.

## 2.7. Python

Python [46] genel bir programlama dili olarak kabul edilebilir. Web uygulamaları, veri madenciliği, yapay zekâ, yapay sinir ağlarında, bilimsel hesaplamalar, nesnelerin interneti (IoT) gibi geniş kavramlı birçok alanda kullanılabilir. 2002 yılından itibaren en popüler 10 program dili arasına giren Python, Şekil 2.6'te görüldüğü gibi son yıllarda artan bir popülarite grafiğini izlemektedir.




**Şekil 2.6.** Popüler Programlama Dilleri [47]

Geniş bir kitle tarafından kullanılması, kolay öğrenilmesi, bilimsel hesaplamalar, yapay zekâ, derin öğrenme, veri madenciliği gibi çok güncel konularda araştırmalara katkı sağlaması, popüler olmasının altındaki nedenlerden birkaçıdır. Dünyada birçok üniversitede programlamaya giriş dersi olarak verilmektedir [48]. Python, C programlama dili baz alınarak geliştirilmiş daha sonra birçok amaçla kullanılmaya başlanmıştır.

## 2.8. Linux

Linux [49], tamamen ücretsiz, kendi çekirdeği üzerinde geliştirilmiş çok sayıda işletim sistemine sahip, birden çok platformda, çok kullanıcılı olarak çalıştırılabilen UNIX işletim sisteminin bir türevi olan temelde bir çekirdek yapısıdır.

Açık kaynak kodlu olması nedeniyle daha güvenlidir ve bir çok ortamda, çeşitli amaçlar için yoğun bir şekilde kullanılmaktadır [50]. Linux Mint, Centos, Ubuntu, Kali Linux, Fedora, Xubuntu, Red Hat ve Debian en çok kullanılan işletim sistemlerindendir.

## 2.9. Dynamics Host Configuration Protocol (DHCP)

Dinamik istemci ayarlama protokolü olarak da adlandırılabilir. İnternette bağlı olan tüm cihazlar birbirleriyle haberleştikleri zaman kendi adresleri veya kimlikleri olarak IP (Internet Protocol) adreslerini kullanırlar. Ağa bağlanan her cihaz statik veya otomatik olarak IP adresi almalıdır. Bu adres o bilgisayar veya cihaz için adres belirleyici kimlik numarası olarak da adlandırılabilir. En büyük ağ olan internet ise TCP/IP mimarisini kullanır.

DHCP, cihazların ağa katılımları için TCP/IP yapılandırma parametlerini belirli kurallar ve protokoller çerçevesinde otomatik olarak yapar. DHCP, client-server (istemci-sunucu) mantığıyla çalışır ve DHCP sunucusu (server) iletişimde UDP 67 portunu kullanırken, DHCP istemcisi (client) UDP 68'i kullanır.

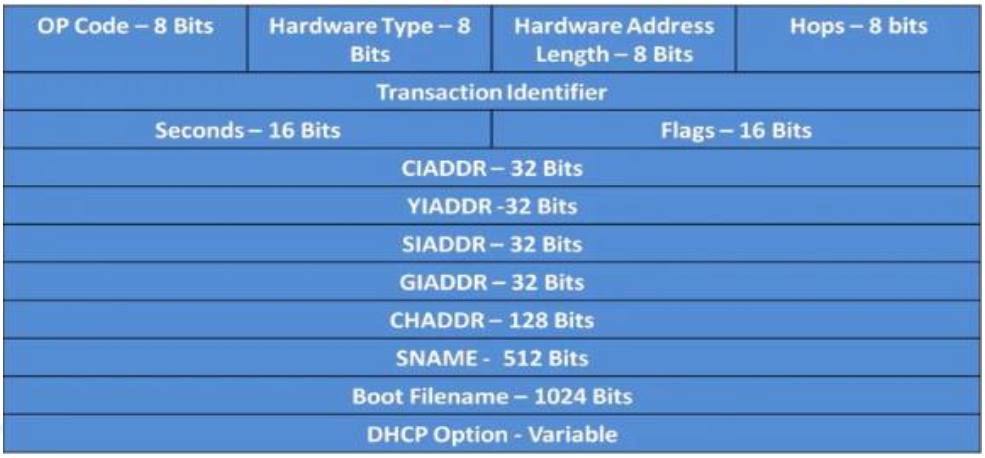
DHCP, RFC (Request for Comment) standartlarından RFC 1531, RFC 1534, RFC 1541, RFC 2131 ve RFC 2132 göre standartlaştırılmıştır. DHCP, BOOTP (Bootstrap Protocol) protokolünü temel alan bir IETF (Internet Engineering Task Force) standardıdır [51]. Bir cihazın internete bağlı olması için IP adresi, alt ağ maskesi, varsayılan ağ geçidi ve DNS adresi gibi bilgilere sahip olması gereklidir. DHCP sayesinde internete bağlanması gereken cihazların IP adresleri ve gerekli olan diğer bilgilerin statik olarak tanımlanması ihtiyacı ortadan kalkar.

DHCP sunucusunun kullanılmadığı büyük ağlarda, IP dağıtımının statik olarak yani elle yapılması sistem yöneticilerine zor görevler yükleyebilir. Özellikle büyük ölçekli ağlarda kullanılmayan IP adreslerinin bilgilerinin tutulması, aynı alt ağı kullanan kullanıcıların belirlenip aynı alt ağ maskesinin tanımlanması gibi dikkat gerektirecek işleri yoluna koymak oldukça zordur. IP çakışması durumu, kullanımda olan bir IP adresinin başka bir cihaza verilmesi durumunda gerçekleşir ve bu karmaşıklık sistem yöneticisi için istenmeyen durumların başında gelmektedir. Bunun gibi birçok sorun Statik IP dağıtımında karşımıza çıkmaktadır. Kampüs ağları gibi büyük sayılabilecek bilgisayar ağlarında, sistem yönetimi ve güvenliği için sanal ağlar (Virtual LAN,

vLAN) oluşturulmakta bu ağlara özel IP gurupları tanımlanmaktadır. Statik IP dağıtımı sanal ağlarda (vLAN) kullanılan IP guruplarını kullanmaları durumunda güvenlik açıkları ortaya çıkabilmektedir [52].

### 2.9.1. DHCPv4 Paket Yapısı

DHCP, TCP/IP ağ katmanlarından uygulama katmanı içerisinde yer alır ve iletişimde UDP protokolünü kullanır [53]. DHCPv4 paket formatı Şekil 2.7'de gösterilen başlık alanlarından oluşmaktadır.




**Şekil 2.7.** DHCPv4 Paket Formatı

Operation Code (OpCode): Option 53 olarak adlandırılır. DHCP mesajının türünü belirtir 1 byte değerindedir.

DHCPDISCOVER için değeri 1'dir.
DHCPOFFER için değeri 2'dir.
DHCPREQUEST içine değeri 3'tür.
DHCPDECLINE için değeri 4'tür.
DHCPACK için değeri 5'tir.
DHCPNACK (Negatif ACK) için değeri 6'dır.
DHCPRELEASE için değeri 7'dir.
DHCPINFORM için değeri 8'dir.

Hardware Type (HType): Bu alan ARP mesajını ileten yerel ağ için kullanılan donanım türünü belirtir. 1 byte değerindedir.

Hardware Adress Length (HLen): Bu mesajda donanım adreslerinin uzunluğu belirtilir. Ethernet veya IEEE 802 MAC adreslerini kullanan ağlar için bu değer 6'dır. 1 byte değerindedir.

Hops: Bu alan sayaç görevi görür, bir istek iletilmeden önce istemci tarafından 0 olarak ayarlanır, iletilerin iletilmesini sayar ve kontrol eder. 1 byte değerindedir.

Transaction Identifier (XID): İstemci tarafından oluşturulan 32 bit değerindeki bu alan, DHCP sunucularından gelen mesajları istemci kendi mesajları ile karşılaştırmak için kullanır. 4 byte değerindedir.

Seconds (Secs): İstemci cihazın kullandığı IP adresinin kullanım süresini saniye cinsinden tutar. DHCP sunucuda istekler biriktiğinde sunucu gelen mesajları sıraya koymak için bu alanı kullanır. 2 byte değerindedir.

Flags: DHCP istemci tarafından DHCP sunucuya yalnızca bir biti kullanılarak broadcast yayını olacak şekilde gönderilir ve geriye kalan diğer 15 bit'lik alan daha sonra kullanılır.. Bu alana 1 yazıldığında DHCP sunucudan, DHCP istemciye cevap gönderilir. 2 byte değerindedir.

Client IP Address (CIAddr): Option 50 olarak adlandırılır. DHCP istemcisi geçerli bir IP adresine sahip olduğunda bu alanı kullanır. İstemci bu alanı, IP adresi isteme sırasında değil, gerçek ve kullanılabilir bir IP adresine sahip olduğunda kiralama yenilemesi sırasında kullanır. Aksi takdirde değeri 0'dır. 4 byte değerindedir.

Your IP Address (YIAddr): DHCP server tarafından DHCP istemciye atanması teklif edilen IP adresidir. 4 byte değerindedir.

Server IP Address (SIAddr): DHCP sunucusunun IP adresidir. DHCP istemci tarafından IP kiralama veya IP yapılandırma işlemi sırasında kullanılır.

Gateway IP Address (GIAddr): Option 3 olarak adlandırılır. DHCP istemcisi ile DHCP sunucusunun farklı alt ağlarda bulunduğu durumlarda istemci ile sunucu arasındaki iletişimi kolaylaştırmak DHCP mesajlarını yönlendirmek için kullanılır. 4 byte değerindedir.

Client Hardware Address (CHAddr): DHCP istemcisini tanımlamak ve iletişimde kullanılmak üzere 2.katmanda kullanılan MAC adres bilgisidir. 16 byte değerindedir.

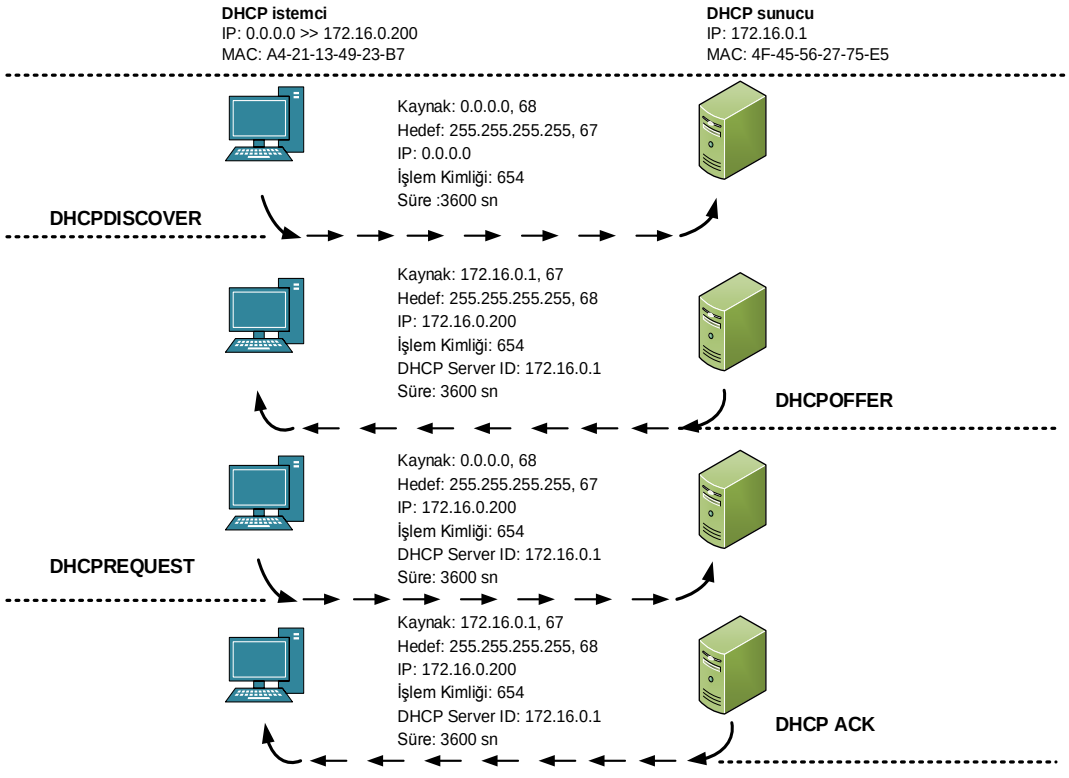
Server Name (Sname): Option 6 olarak adlandırılır. DHCP istemci cihazına DHCPOFFER ve DHCPACK mesajları ile cevap veren DHCP sunucusu isteğe bağlı olarak adını (organization .org) bu alana koyabilir. DNS alan adı kullanılabileceği gibi takma bir ad da kullanılabilir.[54]. 64 byte değerindedir.

Boot Filename (File): DHCP İstemcisi isteğe bağlı olarak bir DHCPDISCOVER iletisinde belirli bir önyükleme dosyası türü istemek için bir önyükleme dosyası adı kullanır. DHCP sunucusu da bir önyükleme dosyası dizinini belirtmek için bu alanı DHCPOFFER'da kullanır. 128 byte değerindedir.

DHCP Options: Bu alan isteğe bağlı olarak DHCP sunucu veya DHCP istemci tarafından kulanılır, temel DHCP istekleri için birkaç parametre içerir. Uzunluğu sabit değildir.

### 2.9.2. Ağ Cihazlarına Otomatik IP Verilmesi

Günümüzde birçok elektronik cihaz internete bağlanabilir ve ağda bulunan diğer cihazlarla etkileşim halinde kullanılabilir şekilde üretilmektedir. Teknolojideki bu gelişme nesnelerin interneti olarak adlandırılır. Bilgisayar, tablet, cep telefonları gibi cihazlar bir ağa yeni dâhil olduklarında ağdaki trafiğe hemen katılamazlar fakat bulundukları yerel ağda broadcast trafiği gerçekleştirebilirler. TCP/IP yapılandırılmasından sonra ağ trafiğine katılabilirler. Bundan dolayı ağda otomatik IP dağıtan bir cihazdan ağa bağlanmak için gerekli olan bağlantı bilgilerini isterler. Bu durumda DHCP istemcisi olarak adlandırılırlar. DHCP sunucusu ile DHCP istemcisi arasında nasıl bir iletişim olduğunu anlamak ve IP adresinin DHCP sunucudan bilgisayara nasıl verildiğini bilmek, istenmeyen DHCP ataklarına önlem alabilmek açısından çok önemlidir [52]. DHCP sunucusu ile ilk defa ağa bağlanan bir DHCP istemci arasındaki diyalog dört mesajda tamamlanır. Şekil 2.8'de DHCP sunucu ile DHCP istemci arasındaki otomatik IP alma sırasındaki bilgiler verilmiştir.




**Şekil 2.8.** Otomatik IP alma-verme işlemi

DHCPDISCOVER: Bu mesaj MAC (Media Access Control) adresine sahip, ağa yeni bağlanan veya ağda yeni açılan istemci cihaz tarafından, ağda bulunan DHCP sunucusunu bulmak için broadcast yayını şeklinde gönderilir. İstemci belirli bir süre sonra cevap alamazsa aynı mesajı tekrar gönderir. DHCP istemcisi, DHCP sunucusunu bulmak için hedef IP adresini 255.255.255.255 olarak kabul eder. Bu IP adresi özel bir adrestir ve literatürde sınırlı yayın adresi olarak adlandırılır. Yayın paketi ağda bulunan tüm cihazlara gönderilir ve yalnızca o ağdaki DHCP sunucusu veya sunucuları tarafından cevaplanır. Ağda bulunan diğer bilgisayarlar da bu mesajı alır, kendileriyle ilgili olmadığını anladıklarında mesajı çöpe atarlar. DHCP istemci ve sunucusu UDP (User Datagram Protocol) aktarma protokolünü kullanır. İstemci tarafı UDP port 68'i sunucu tarafı ise UDP port 67'yi kullanır. DHCP sunucusu birden fazla olan ağlarda DHCPDISCOVER mesajına ilk cevap veren sunucu, DHCP istemcisine IP verme hakkını elde etmiş olur. DHCP istemci henüz bir IP adresine sahip olmadığından dolayı ağdaki iletişimini IP adresi 0.0.0.0 olacak şekilde MAC adresi ile beraber yürütmektedir. Buradaki dikkat edilecek önemli husus istemci cihaza daha önce statik IP adresinin verilmemiş olması gerek. Statik IP adresi verilmiş ise cihaz otomatik IP adresi yapılandırılmasına kapalıdır demektir. Bu durumda otomatik olarak yeni IP adresi alamaz.. DHCPDISCOVER mesajına cevap veren DHCP sunucusu kendisine ayrılmış IP havuzundan bir IP adresi belirler ve istemci cihaza bu IP adresini kendi bilgileri ile birlikte iletir. DHCPDISCOVER paketinin içeriği;

Hedef IP adresi: DHCP sunucusunun IP adresi bilinmediğinden broadcast olarak hedef IP adresi 255.255.255.255 şeklinde tüm ağa gönderilir.

Hedef MAC adresi: DHCP sunucusunun MAC adresi bilinmediğinden hedef MAC adresi FF.FF.FF.FF.FF.FF şeklinde gönderilir.

Kaynak IP adresi: IP talebinde bulunan istemci cihazın henüz bir IP'si olmadığından 0.0.0.0 olarak atanır.

Kaynak MAC adresi: Ağa bağlanmak istenen her cihaz mutlaka bir Ethernet kartına sahiptir. Ethernet kartlarının hepsinde o ethernet kartına özgü MAC adresi bulunur. İstemci cihazın üzerindeki ethernet kartının MAC adresi paket içeriğinde gönderilir. Örneğin A4-21-13-49-23-B7.

DHCPOFFER: DHCP sunucusu, istemci cihazla arasındaki iletişimi DHCPOFFER mesajı ile başlatır. Bu mesajda istemci cihaza verilebilecek IP adresi bilgisi, istemci cihazın MAC adresi ile birlikte DHCP sunucusuna ait IP bilgileri bulunmaktadır. İstemci cihaz kendisine gönderilen

IP adresini almak için DHCP sunucusuna tekrar bir mesaj gönderir. DHCPOFFER paketinin içeriği;

Hedef IP adresi: İstemci cihazın IP adresidir ve henüz bir adrese sahip olmadığından dolayı 0.0.0.0 olarak devam eder.

Hedef MAC adresi: İstemci cihazın MAC adresidir. A4-21-13-49-23-B7.

Kaynak IP adresi: DHCP sunucusunun IP adresidir. 172.16.0.1

Kaynak MAC adresi: DHCP sunucusunun MAC adresidir. 4F-45-56-27-75-E5.

DHCPREQUEST: Bu mesajı istemci cihaz kendisine DHCP sunucusu tarafından DHCPOFFER mesajı ile birlikte gönderilen IP adresini almak istediğini belirtmek için DHCP sunucusuna broadcast mesajı gönderir. Aynı zamanda istemci cihaza varsa ağda kendisine DHCPOFFER mesajı gönderen diğer sunuculara cevap olarak da mesaj gönderilir ve IP adresi isteğini iptal ettiği bilgisini gönderir. DHCP sunucusu DHCPREQUEST mesajını aldıktan sonra istemci cihazın IP adresini kendisinden almak istediğini anlar ve ona gerekli diğer bilgililer ile birlikte yapılandırma bilgilerini gönderir. Bu aşamadan sonra istemci cihaz IP adresini kiralama yöntemiyle DHCP sunucusundan kiralar. Kiralama süresi sistem yöneticisi tarafından daha önce sunucu ayarlarında belirtilen süredir.

DHCPACKNOWLEDGEMENT: İstemci cihazdan DHCPREQUEST mesajını alan DHCP sunucusu tarafından istemci cihaza gönderilen son mesajdır. Bu mesajda istemci cihazın internete bağlanması için gerekli asgari bilgiler olan, IP adresi, alt ağ maskesi, DNS (Domain Name System) bilgilerini içerir. Bu mesajdan sonra ağ üzerinde gerekli ayarlamalar yapılmış ise istemci cihaz internete bağlanabilir.

### 2.9.3. Ağ Bağlantısı için Gerekli Asgari Bilgiler

IP Adresi: Ağa bağlanan cep telefonu, tablet, bilgisayar gibi ağ cihazlarının birbirleriyle etkileşimde bulunabilmeleri için gerekli olan özgün kimlik bilgisidir.
Alt Ağ Maskesi: IP maskesi olarak da bilinir. Ağdaki cihazların aynı ağda olup olmadıkları kullandıkları alt ağ maskesi belirler. IP adresinin ilk 24 bitine karşılık gelen kısımdır. Ağa bağlı her cihazın mutlaka alt ağ maskesi belirlenmiş olması gerekir.

Default Gateway: Varsayılan ağ geçidi olarak da bilinir. Bir ağın internete bağlantısı var ise o ağdaki cihazların sahip olması gereken geçit-kapısı IP adres bilgisine denir.

DNS IP Adresi: Ağdaki bilgisayarlar için isim çözümleme işlemi yapacak olan sunucunun IP adresidir. İnternet bağlantısının devamlılığını sağlar. Bir ağda bir veya daha fazla sayıda olabilir.

### 2.9.4. DHCP Relay

Kampüsler, bankalar veya devlet kurumları gibi çalışan büyük bilgisayar ağların gerek güvenlik için gerekse ağ yönetimini kolaylaştırmak için veya ihtiyaç olması durumunda ağı birden fazla alt ağlara ve geniş alan ağlarına bölmek zorundadırlar. Ağda IP isteğinde bulunan bir cihaz IP işlemi tamamlanıncaya kadar DHCP sunucusu ile yaptığı tüm haberleşmeyi broadcast yayını yaparak gerçekleştirir. Bu yayını bulunduğu alt ağın sınırları içerisinde bulunan cihazlara erişim yapabilir. Güvenlik veya ağ yoğunluğunu azaltmak adına yayınlar alt ağlar dışına yayılamazlar. Kendisine yayın gönderilen DHCP sunucunun başka bir ağda olması durumunda normal koşullarda DHCP sunucusunun bu yayınını alması mümkün değildir. Bu sorunu aşmak için yönlendiricilerde DHCP mesajları için özel ayarlamalar yapılır. Bu işleme DHCP Relay adı verilir.

### 2.9.5. DHCP Güvenliği

Günümüzde insanlar ve kurumlar birçok işlemi kullandıkları cihazlar sayesinde internet ağı üzerinden gerçekleştirmektedirler. IT (Information Tecnology) dünyasındaki hızlı gelişmelerin getirdiği kolaylıklar yanında bazı sıkıntıları da barındırmaktadır. Bunların en başında güvenlik konusu gelmektedir [10].

DHCP servisi 1997 yılında RFC 2131 standardı ile tarif edildiğinden beri çok çeşitli saldırılara maruz kalmıştır. DHCP servisine saldırı yapılabilmesi için saldırganın sunucu ile aynı yayın alanında (broadcast domain) bulunması gerekir. Aynı kurum içerisinde kablo alt yapısının kurumun kendi personeli tarafından kullanıldığı düşünülerek bu uygulamanın büyük bir tehdit oluşturmadığı düşünülüyordu. Ancak günümüzde misafir kullanıcı, kurum içi bilgisayarların ele geçirilebilmesi, kurum içinde DHCP özelliği olan yetkisiz modem kurulumu gibi değişen şartlar sonucunda bu gibi ataklar ciddi tehditler oluşturmaya başlamıştır. [55]

DHCP, ağa bağlanabilme donanımına sahip cihazların TCP/IP ağ ayarlarını merkezi bir sunucudan alma işlemini gerçekleştirir. [56]. DHCP, cihazların otomatik olarak ağa

bağlanmasında ağ kullanıcılarına ve yöneticilerine sağlamış olduğu kolaylıktan dolayı en çok kullanılan protokollerin başında gelmektedir.

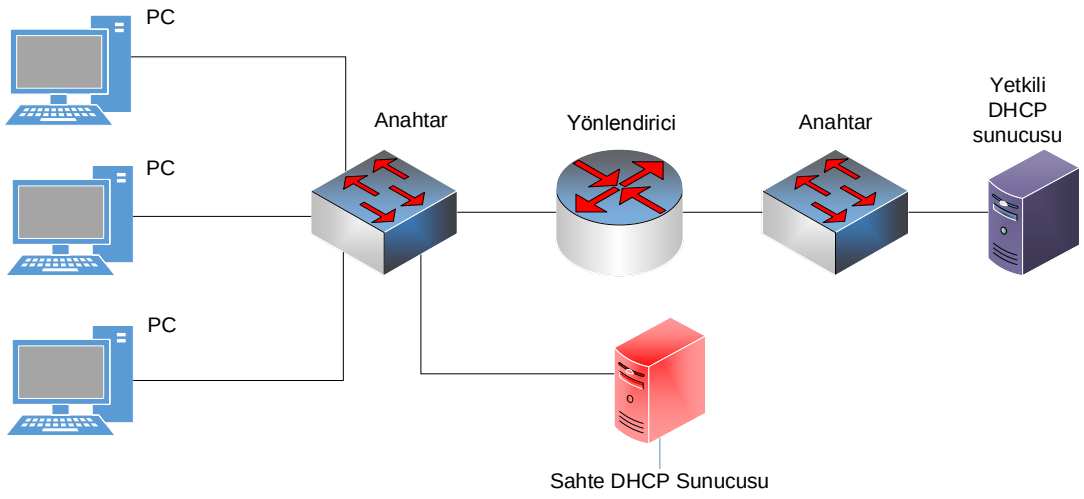
DHCP güvenliğinin riskli olmasının temel nedeni bir DHCP istemci ile DHCP sunucusu arasındaki iletişimin herhangi bir kontrol mekanizmasına bağlı olmamasından kaynaklıdır. 2001 yılında "Authentication for DHCP Messages" DHCP mesajları için kimlik denetimi ile RFC 3118 başlığı altında yetkisiz DHCP istemcilerinin ve DHCP sunucularının atak yapmalarını engellemek amacı ile bir yöntem önerilmiştir [57]. Ancak bu şekilde kimlik denetimi yapılabilmesi için sunucunun ve istemcinin bunu desteklemesi gerekmektedir. Şu anda yaygın olarak kullanılan DHCP istemcilerinde bu destek pek bulunmamaktadır. Belki aynı kurum içerisindeki bütün DHCP istemcilere ve DHCP sunuculara bu şekilde koruma yapabilen bir DHCP yazılımı kurulabilir. Bu durumda ağa sonradan katılan ve misafir diye adlandırılan kullanıcı grubuna bu şekilde bir kurulum yapılması mümkün değildir. DHCP servisi güvenliği, ağ yöneticileri tarafından çok zaman göz ardı edildiğinden saldırılara karşı savunmasız kalabilmektedir. DHCP servisi üstlendiği görev itibariyle ağ yöneticilerinin işin çok kolaylaştırmaktadır fakat bu servisin çalışması ile ilgili çıkabilecek herhangi bir sorun büyük sıkıntılara neden olabilir. DHCP sürekliliği ve güvenliği bundan dolayı çok önemlidir.

### 2.9.6. Sahte (Rogue) DCHP Saldırıları

Etkileşimli ağ cihazlarının çeşitliliğinin artması ve kullanılan IPv4 sayısındaki azalma ile birlikte DHCP sunucularına düşen yük artmaktadır. Bu durum beraberinde sahte DHCP sunucularının artışına da neden olmaktadır [10]. Sahte DHCP sunucusunu; kısaca ağda bulunan izin verilmemiş veya yetkisiz bir sunucu olarak da tanımlayabiliriz. Bazen bu tür sunucular kötü niyetli olmayan kullanıcılar tarafından da ağa açık olan portlar üzerinden bağlanabilir. Ağ kullanıcısı DHCP özelliği olan bir cihazı ağ üzerindeki bir porta takmış olabilir. Bu durumda ağda oluşacak olan IP isteklerine yetkisiz olan bu cihazdan tarafından IP verilebilir. Sonuç olarak yetkisiz cihazdan IP alan cihazlar kullandıkları IP bloğunun farklı olması nedeni ile internete çıkamayacak veya farklı sorunlar yaşayacaklardır. Aynı zamanda bu yetkisiz sunucular ağdaki bulunan diğer cihazlarında internete erişimlerini de engellemiş olabilirler. Daha sonra ağ kullanıcıları tarafından yerel ağ içindeki bağlantılarda veya internet bağlantıları ile ilgili şikâyetler gelmeye başlar. Sahte DHCP olayları her zaman iyi niyetle başlamaz. Daha da önemlisi kötü niyetli ve planlı olarak gerçekleştirilen sahte DHCP saldırıları vardır. Kasıtlı olarak gerçekleştirilen sahte DHCP saldırısında saldırgan kurban seçtiği bilgisayarın ağ erişimini kendi DNS sunucuları üzerinden gerçekleştirerek ağ trafiğini kontrol edebilir onu istediği sahte

sunuculara yönlendirerek önemli bilgilerini elde edebilir. Ağ yöneticileri için sahte veya yetkisiz DHCP sunucularını bulmak ve onları devre dışı bırakmak son derece zor bir iştir. DHCP sunucuları ağ hizmetine başladıkları tarihten beri sürekli olarak tehdit altında olmuşlardır. DHCP sunucularının güvenliği ile ilgili birçok yöntem geliştirildiyse bile çalışma yöntemleri gereğince saldırılara her zaman açık olmuşlardır. DHCP sunucularının güvenliği aynı zamanda ağ yöneticilerinin sunucuyu doğru konumlandırması ve yapılandırması ile ilgilidir. Günümüzde çoğunlukla geleneksel ağlarda kullanılan anahtarlama ve yönlendirme cihazları ağ güvenliği ile ilgi çok sınırlı imkânlar sunmaktadır. Bir ağda birbirinden farklı markalarda cihazların kullanımı güvenliği düşürmekte güvenlik ayarlarının eksik yapılmasına neden olmaktadır [58].

YTA paradigması ile birlikte DHCP sunucularına yapılan saldırıların tespiti ve engellenmesi ile ilgili imkânlar artmıştır [10]. YTA ile ağ uygulamalarında yazılan programlara bağlı olarak ağ topolojisinin güncel durumu görüntülenebilir, DHCP sunucusuna belli bir süre içerisinde gelen DHCP_DISCOVER mesaj sayısı sınırlandırılabilir, IP alan bilgisayarın yasal bir DHCP sunucusundan alıp almadığı kontrol edilebilir. Şekil 2.9'da yasal ve sahte DHCP sunucularının aynı anda hizmet verdiği bir yapılandırma gösterilmiştir. Yasal DHCP farklı bir alt ağda olduğundan maalesef tüm PC'ler gerekli tedbirler alınmaz ise sahte DHCP sunucusundan IP adresi alacaklardır.




**Şekil 2.9.** Sahte DHCP sunucusu

Sahte bir DHCP sunucusunu ağ içerisinde tespit etmek için çeşitli yöntemler kullanılması gerekir. Bu yöntemler ve yöntemlerde uygulanacak olan ağ müdahaleleri uzman kişiler tarafından gerçekleştirilmelidir. Bu yöntemlerden bazıları şunlardır;

- Son kullanıcıların sahte olan DHCP sunucusundan IP almaya devam edip etmediğini anlamak için yetkili olan DHCP sunucusunu devre dışı bırakmak.
- IP alan bilgisayarların ağ geçidini kontrol etmek.
- Kampüs ağları gibi birden fazla yerel ağa sahip bir ağda ise internete erişimde sıkıntı yaşanan yerel ağları (vLAN) kapatmak.
- IP adresleri ile MAC adreslerinin eşlemelerini elde etmek için ARP tablosunu görüntülemek.
- Sahte DHCP sunucusunun MAC adresini belirlemek.
- MAC adresi belirlenen sahte DHCP sunucusunun bağlantı portonu tanımlamak.
- Son olarak da portu kapatmak.

## 2.10. Domain Name Systems (DNS)

Internet dünyasındaki tüm cihazlar birbirleriyle IP adresleri aracılığıyla haberleşirler. Kullanıcıların bu IP adreslerini akılda tutması hem çok zor hem de olanaksızdır. Alan adı sistemi olarak da bilinen DNS hizmeti sayesinde, IP adresleri ile host adresleri arasında iki taraflı dönüşüm sağlanır. Herhangi bir adrese erişim sağlamak isteyen bir kullanıcı bu adrese ait bir IP adresini kullanması ve bu bilgiyi akılda tutması arama çubuğuna o adresi yazmasından çok daha zordur. Bu sistem sayesinde internet ortamında bulunan dağıtık durumdaki erişim adresleri bir araya toplanarak veri tabanı oluşturulur ve internetin devamlılığı sağlanmış olur. DNS sunucuları, URL (Uniform Resource Locator) ya da FQDN(Fully Qualified Domain Name) olarak adlandırılan adreslerin IP'lere çözümlenmesini sağlarlar. Böylece hatırlaması zor olan IP adreslerini kullanmak yerine, hatırlaması daha kolay olan host isimlerinin kullanımına izin verirler.

Caching (hafızada tutma) işlemi DNS sorgularını hızlandıran en önemli özelliklerden biridir. DNS sunucuları ve istemciler bir kere öğrendikleri host adreslerine karşılık gelen IP adres bilgilerini veya IP adres bilgisine karşılık gelen host adresi bilgilerini (Ters DNS Çözümlenmesi) belli bir TTL (Time To Live) değeri süresince hafızalarında (caching) tutarlar. Böylece yapılacak sonraki aynı sorgular için işlem süresi azalır [59].

Sahte DHCP saldırılarında DNS'lerde önemli bir yer tutar. Zira kötü niyetli DHCP sunucusu aynı zamanda bir DNS gibi davranıp istemcileri yanlış yönlendirebilir. Yanlış IP çözümlemesi yapabilir.

# 3. BULGULAR VE TARTIŞMA

## 3.1. Benzetim Çalışması

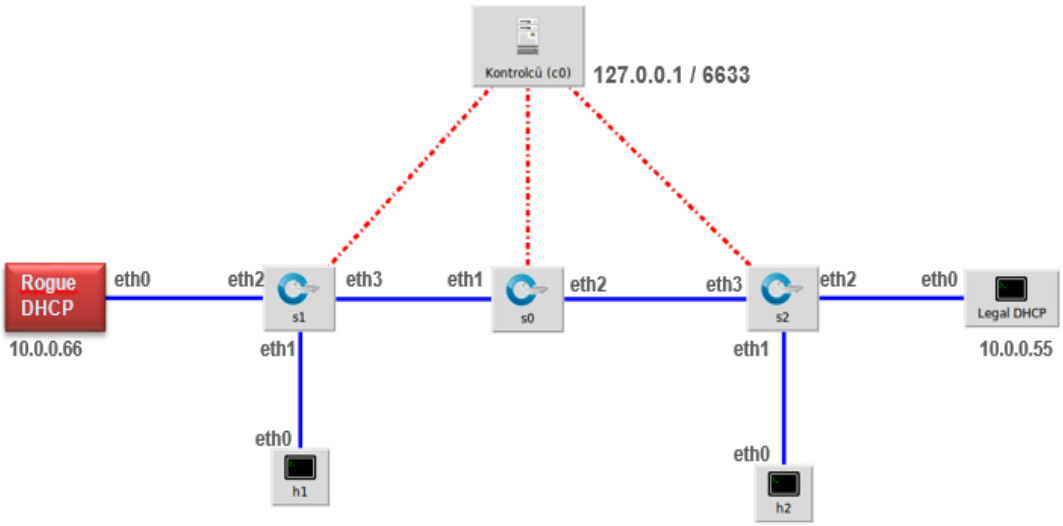
Bu tez çalışmasında yapılan teorik araştırmaların uygulamasını çalıştırmak için kullanılan donanım ve yazılımların listesi Çizelge 3.1'de gösterilmiştir.

Çizelge 3.1 Benzetim ortamında kullanılan yazılım ve donanım bilgileri

| **Kullanılan araçlar** | **Özellikleri** |
|---|---|
| Dizüstü Bilgisayar | Intel Core i5-3317U CPU @ 1.70Ghz, 8 GB RAM, Windows 10 64bit işletim sistemi |
| Sanallaştırma Yazılımı | Oracle VirtualBox 6.0.22 |
| Sanal Makina | 2GB MB RAM, 20 GB HD |
| Konak İşletim Sistemi | Ubuntu 14.04.6 LTS, Trusty Tahr, 64 bit |
| Simülasyon Yazılımı | Mininet 2.2.1 |
| Kontrolcü | POX 0.1.0 (betta) |
| Programlama Dili | Python 2.7.6 |

### 3.1.1. Ağ Senaryosu

Ağ senaryosu Mininet simülasyonunun çizim uygulaması olan Miniedit'te hazırlanmış olup Şekil 3.1'de gösterildiği gibidir.

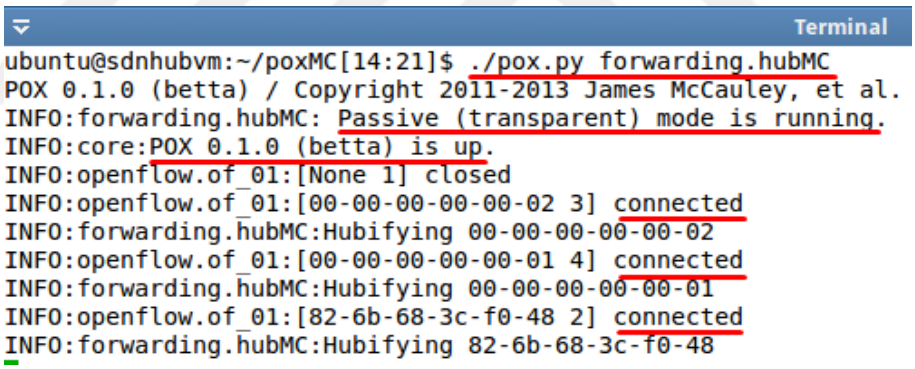



**Şekil 3.1.** Kullanılan ağ senaryosu

Şekil 3.1'deki ağ senaryosunda iki adet bilgisayar (h1, h2), üç adet anahtarlama cihazı (s0, s1, s2), bir adet legal (yasal) DHCP, bir adet Rogue (sahte) DHCP ve bir adet kontrolcü (c0) bulunmaktadır. Oluşturulan ağda bilgisayarlar ve DHCP sunucular aynı ağda bulunmaktadır. Bant genişliği h1 ve h2 bilgisayarları için varsayılan olarak değiştirilmemiş olup aynıdır. DHCP sunucularda DHCPv4 versiyonu, bilgisayarlarda ise TCP/IPv4 sürümü kullanılmaktadır.

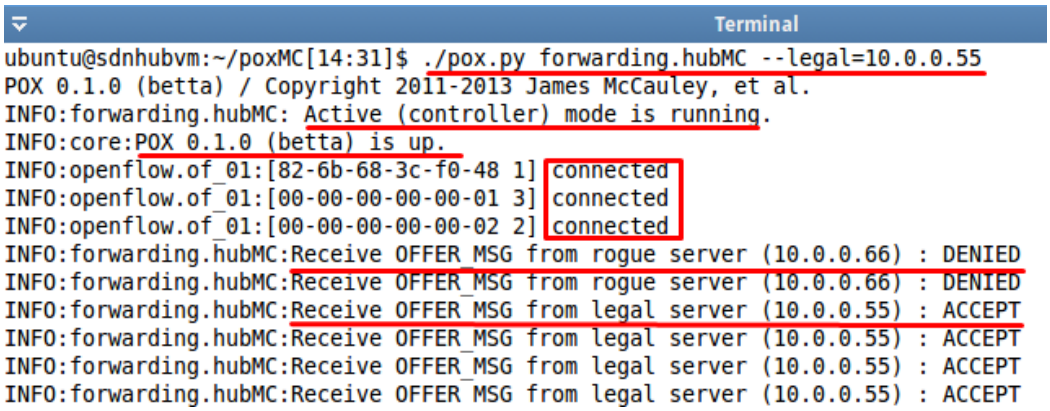
**Kontrolcü (c0):** Kontrolcü ağ senaryosunda POX 0.1.0 (betta) sürümü, Mininet uygulaması dışında ayrı bir servis olarak 6633 nolu port üzerinden hizmet verecek şekilde konumlandırılmıştır. POX yazılım dosyası olan "hubMC.py" Ek1'de gösterilmiştir. Bu dosya pox/forwarding klasöründe bulunan hub.py dosyasında gerekli düzenlemeler yapılarak elde edilmiştir.

POX kontrolcüsü iki şekilde çalışabilmektedir. Servis herhangi bir kontrol yapmadan **"./pox.py forwarding.hubMC"** komutu ile başlatılırsa Şekil 3.2'deki ekran çıktısından görüleceği üzere POX pasif (transparent) modda çalışır. Bu durumda kontrolcü cihaz ağda herhangi bir DHCP denetimi yapmaz.




**Şekil 3.2.** POX kontrolcüsü pasif modda

Servis kendisine tanımlanan **"./pox.py forwarding.hubMC.py --legal=10.0.0.55"** yasal bir DHCP IP adresi ile başlatılırsa Şekil 3.3'te görüldüğü üzere aktif modda devreye girecek ve –legal parametresi ile kendisine tanımlanan ip dışınki iplerden gelecek dhcp offer veya ack mesajlalrını engelleyecektir. Bu şekilde spoof etmek isteyen sahte dhcp sunucularının ip dağıtımı engellenmiş olacaktır.




**Şekil 3.3.** POX kontrolcüsü aktif modda

**Yasal DHCP Sunucusu:** Ağ senaryosunda 10.0.0.55 ile tanımlanan host tabanlı bir sunucudur. POX kontrolcüsü aktif modda çalıştırıldığında ağda yasal DHCP sunucu olarak DHCP istemci cihazlarına IP dağıtan tek sunucu görevini görür. Bu sunucu Mininet/dhcp_spoof klasörü içinde **"./dhcp_spoof_MC.py"** komutu ile Şekil 3.4' gösterilen kodlarla çalışmakta olan programın içinden **"udhcpd.conf"** dosyası çağrılarak çalıştırılır.

Udhcpd [60], gömülü sistemlere yönelik çok küçük boyutta bir DHCP sunucu programıdır. Aynı zamanda Udhpc adında DHCP istemcisi de mevcuttur. Program çalışır durumda ve RFC 2131 standartlarında, Busybox'ın bir parçası olarak kullanılmaktadır.

```
# Good DHCP Server
def makeDHCPconfig(filename, intf, gw, dns):
    config = (
        'interface %s' % intf,
        DHCPTemplate,
        'option router %s' % gw,
        'option dns %s' % dns,
        '')
    with open(filename, 'w') as f:
        f.write('\n'.join(config))

def cleanDHCPconfig(host, filename):
    host.cmd('rm ', filename)

def startGoodDHCPserver(host, gw, dns):
    info('* Starting good DHCP server on', host, 'at', host.IP(), '\n')
    dhcpConfig = '/tmp/%s-udhcpd.conf' % host
    makeDHCPconfig(dhcpConfig, host.defaultIntf(), gw, dns)
    #host.cmd('busybox udhcpd -f', dhcpConfig, '1>/tmp/%s-dhcp.log 2>&1  &' % host)
    host.cmd('udhcpd -f', dhcpConfig, '1>/tmp/%s-dhcp.log 2>&1  &' % host)

def stopGoodDHCPserver(host):
    info('* Stopping good DHCP server on', host, 'at', host.IP(), '\n')
    host.cmd('kill %udhcpd')
    dhcpConfig = '/tmp/%s-udhcpd.conf' % host
    cleanDHCPconfig(host, dhcpConfig)
```

**Şekil 3.4.** Yasal DHCP sunucusu başlatılması

Yasal DHCP sunucusundan IP alan bilgisayardan www.google.com adresine ping atıldığında Google'a ait bir IP adresine ping attıkları gözlenmektedir. Bu DHCP sunucusuna ait olarak kullanılan DNS sunucusu dnsmasq'dır ve program başladıktan sonra çalışması Şekil 3.5'te gösterildiği gibidir.

DNSmasq [61] DHCP sunucusu ile entegre olabilen, DHCP tahsisli adresleri olan yerel makinaların DNS'te görünmesine izin veren, küçük bilgisayar ağları için tasarlanmış, DNS önyükleme özelliklerini sağlayan, ücretsiz bir DNS yazılımıdır.

```
# Good DNS Server
def startGoodDNSserver(host):
    info('* Starting good DNS server', host, 'at', host.IP(), '\n')
    host.cmd('dnsmasq -k -x /tmp/dnsmasq.pid -C - 1>/tmp/dns-good.log 2>&1 </dev/null &')

def stopGoodDNSserver(host):
    info('* Stopping good DNS server', host, 'at', host.IP(), '\n')
    host.cmd('kill $(cat /tmp/dnsmasq.pid)')
    host.cmd('rm /tmp/dns-good.log')
    host.cmd('rm /tmp/dnsmasq.pid')
```

**Şekil 3.5.** Yasal DNS sunucusu başlatılması

**Sahte DHCP Sunucusu:** Ağ senaryosunda 10.0.0.66 IP adresi ile tanımlanan host tabanlı bir sunucu üzerinde çalışan Ettercap 0.8.2 sürümü olan sahte DHCP saldırı yazılımıdır. DHCP istemci tarafından başlatılan IP isteğinde yasal DHCP sunucusundan gelen DHCP ACK mesajları burada dinlenip taklit edilerek istemciye sahte IP adresi veriliyor. Ettercap [62] birden fazla özelliği olan trafik dinleme ve trafik injection yapabilen bir ağ analiz programıdır.

POX kontrolcüsü pasif modda çalıştırıldığında ağda IP talebinde bulunan DHCP istemcilere sahte DHCP sunucusu üzerinden IP adresi vererek bu bilgisayarları kendi DNS sunucusuna yönlendirir. Böylece ele geçirdiği bilgisayarları ağ erişimlerini kontrol edebilir istediği bilgileri elde edebilir. Mininet/dhcp_spoof klasörü içinde **"./dhcp_spoof_MC.py"** komutu ile başlatılan program dosyası içinde **"ettercap.conf"** dosyası çağrılarak çalıştırılır.

```
# Bad DHCP Server
def startBadDHCPserver(host, gw, dns):
    info('* Starting bad DHCP server on', host, 'at', host.IP(), '\n')
    host.cmd('ettercap -T -M dhcp:10.0.0.10-90/255.255.255.0/%s -a etter.conf &>/tmp/ettercap.log &' % host.IP())

def stopBadDHCPserver(host):
    info('* Stopping bad DHCP server on', host, 'at', host.IP(), '\n')
    host.cmd('kill %ettercap')
    host.cmd('rm /tmp/ettercap.log')
```

**Şekil 3.6.** Sahte DHCP sunucusu başlatılması

Sahte DHCP sunucusundan IP alan bilgisayardan www.google.com adresine ping atıldığında bu bilgiyarları kendi DNS sunucusu olan DNSChef uygulaması üzerinden dinlemeye başlar. DNSChef [63], sızma testleri ve kötü amaçlı yazılım analizleri için son derece gelişmiş özelliklere sahip ve yapılandırılabilir bir DNS proxy aracıdır.

DNS proxy veya diğer adıyla "Sahte DNS", örneğin bir DNS proxy adresi "www.google.com" u Internet üzerinde bir yerde gerçek bir sunucu yerine sonlandırma veya müdahale amacıyla yerel bir makineye işaret etmesini sağlamak amacıyla sahte isteklerde kullanılabilir. Bu DHCP sunucusuna ait sahte DNS'in başlatılması Şekil 3.7 de gösterildiği gibidir.

```
# Bad DNS Server
def startBadDNSserver(host):
    info('* Starting bad DNS server', host, 'at', host.IP(), '\n')
    host.cmd('python dnschef.py --file=dnschef.ini -i 10.0.0.66 1>/tmp/dns-bad.log 2>&1 &')

def stopBadDNSserver(host):
    info('* Stopping bad DNS server', host, 'at', host.IP(), '\n')
    host.cmd('kill %dnschef')
    host.cmd('rm /tmp/dns-bad.log')
```

**Şekil 3.7.** Sahte DNS sunucusu başlatılması

Bilgisayarlar IP isteklerinde bulunduklarında veya IP kiralama sürelerini yenilemek istediklerinde bu isteklerine DHCP sunucularından gelen cevap öncelikle kontrolcü cihazına gider.

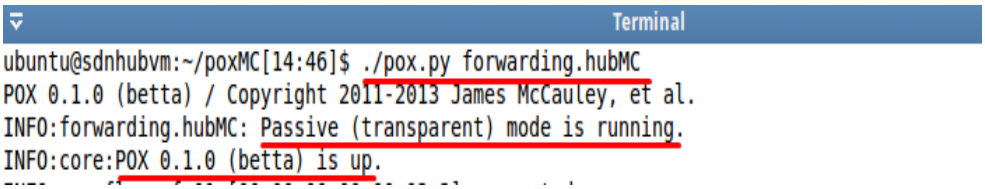
**Programın Başlatılması:** Benzetim senaryosu mininet/dhcp_spoof klasörü altında **"sudo ./dhcp_spoof_MC.py eth0 0 remote"** komutu ile program POX kontrolcüsüne bağlanarak Şekil 3.8'de gösterildiği gibi Mininet uygulamasında başlatılmıştır.

Buradaki "eth0" parametresi hangi ethernet biriminin benzetiminde internet çıkışı olarak kullanılacağını, "0" atak koruma tipini, "remote" parametresi ise YTA kontrolcünün dışardan olacağını mininete söyler

```
Terminal
ubuntu@sdnhubvm:~/mininet/dhcp-spoof[16:16] (master)$ sudo ./dhcp_spoof_MC.py eth0 0 remote
*** Creating network
*** Adding controller
*** Adding hosts:
dhcp evil h1 h2
*** Adding switches:
s0 s1 s2
```

**Şekil 3.8.** Programın başlatılması

**Senaryo 1:** İlk senaryoda kontrolcü cihazına DHCP sunuculardan gelen mesajlar, herhangi bir denetime tabi tutulmadan pasif modda geçirilir. Ağda bulunan bilgisayarlar sahte DHCP sunucusundan IP almış olabilirler. Bilgisayarların IP alma durumları, DHCP sunucularının daha hızlı olmalarına bağlıdır.

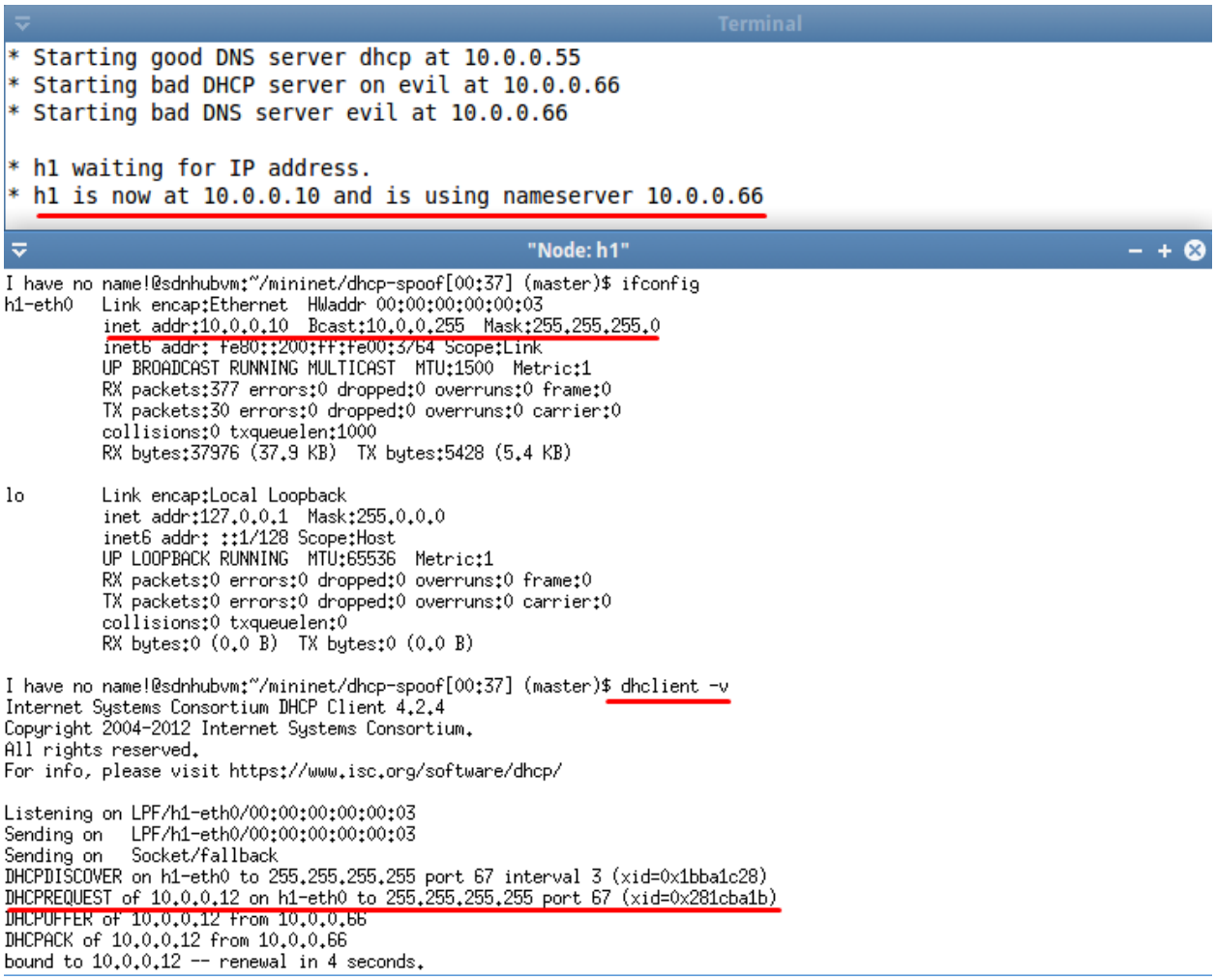



**Şekil 3.9.** POX kontrolcüsü pasif modda başlatılması

**"./pox.py forwarding.hubMC"** komutu Şekil 3.9'da gösterildiği gibi başlatılan POX kontrolcüsü, herhangi bir DHCP sunucusu denetimi yapmamaktadır. Program çalıştırıldığında ağdaki DHCP sunucularının gelen IP isteklerine yanıt verdikleri Şekil 3.10 Wireshark ağ analiz uygulamasında görülmektedir.

| | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 780 | 8.836898000 | 0.0.0.0 | 255.255.255.255 | DHCP | 342 | DHCP Discover - Transaction ID 0x45896a1a |
| 781 | 8.839372000 | 00:00:00_00:00:01 | Broadcast | ARP | 60 | Who has 10.0.0.23? Tell 10.0.0.55 |
| 782 | 8.840592000 | 10.0.0.66 | 255.255.255.255 | DHCP | 582 | DHCP Offer - Transaction ID 0x45896a1a |
| 783 | 8.844757000 | 0.0.0.0 | 255.255.255.255 | DHCP | 342 | DHCP Request - Transaction ID 0x45896a1a |
| 784 | 8.847025000 | 10.0.0.66 | 255.255.255.255 | DHCP | 582 | DHCP ACK - Transaction ID 0x45896a1a |
| 785 | 9.340254000 | 00:00:00_00:00:01 | Broadcast | ARP | 60 | Who has 10.0.0.23? Tell 10.0.0.55 |
| 786 | 9.763864000 | 0.0.0.0 | 255.255.255.255 | DHCP | 342 | DHCP Discover - Transaction ID 0x59683d50 |
| 787 | 9.842687000 | 00:00:00_00:00:01 | Broadcast | ARP | 60 | Who has 10.0.0.23? Tell 10.0.0.55 |
| 788 | 10.264506000 | 0.0.0.0 | 255.255.255.255 | DHCP | 342 | DHCP Discover - Transaction ID 0x59683d50 |
| 789 | 10.765358000 | 0.0.0.0 | 255.255.255.255 | DHCP | 342 | DHCP Discover - Transaction ID 0x59683d50 |
| 790 | 10.766726000 | 10.0.0.66 | 255.255.255.255 | DHCP | 582 | DHCP Offer - Transaction ID 0x59683d50 |
| 791 | 10.875432000 | 10.0.0.55 | 255.255.255.255 | DHCP | 323 | DHCP Offer - Transaction ID 0x45896a1a |
| 792 | 10.887651000 | 00:00:00_00:00:01 | Broadcast | ARP | 60 | Who has 10.0.0.24? Tell 10.0.0.55 |
| 793 | 11.267002000 | 10.0.0.66 | 255.255.255.255 | DHCP | 582 | DHCP Offer - Transaction ID 0x59683d50 |
| 794 | 11.375590000 | 10.0.0.55 | 255.255.255.255 | DHCP | 323 | DHCP Offer - Transaction ID 0x45896a1a |

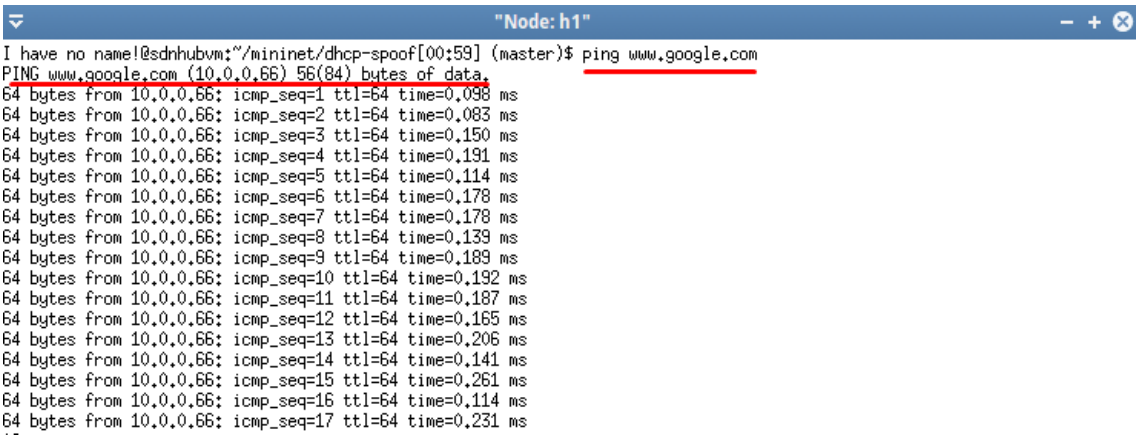
**Şekil 3.10.** POX kontrolcüsü pasif modda çalıştırıldığında Wireshark görüntüsü




**Şekil 3.11.** h1 bilgisayarının durumu

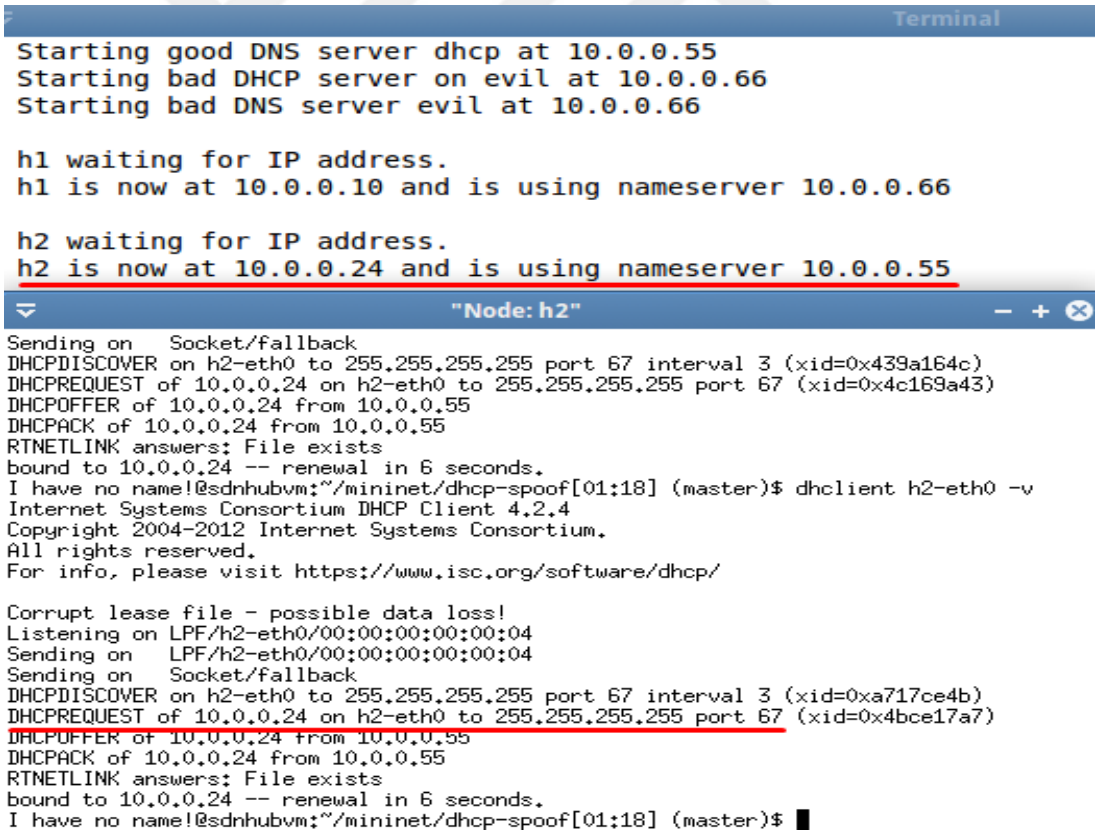
Şekil 3.11'de görüldüğü gibi senaryoda kontrolcüye herhangi bir yasal DHCP tanımlanması yapılmadığı zaman bilgisayardan h1'in ağ topolojisindeki yeri, sahte DHCP (10.0.0.66) sunucusuna yakın olduğundan bu sunucudan 10.0.0.10 IP adresini aldığı görülmektedir. Bu bilgisayarın arayüzüne Mininet'te "xterm h1" yazılıp girdikten sonra "dhclient –v" komutu ile IP yenilemesi yapıldığında bu kez aynı sunucudan 10.0.0.12 IP adresini aldığı Şekil 3.11'de görülmektedir.

Bu bilgisayardan www.google.com adresine ping gönderildiğinde ise pingi kendi dns sunucuna yönlendirildiği Şekil 3.12'de gözlemlenmektedir.




**Şekil 3.12.** h1 bilgisayarından ping atılması

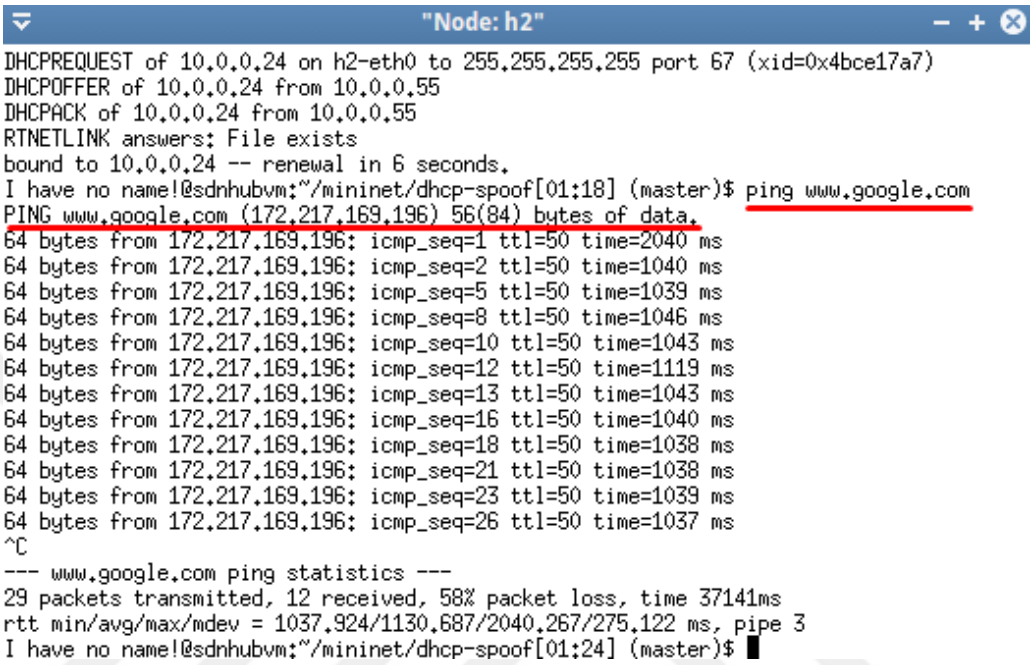
Yine bu senaryoda incelenen bir diğer bilgisayar h2'dir. Şekil 3.13'de görüldüğü gibi h2 bilgisayarı yasal DHCP (10.0.0.55) sunucusuna daha yakın olduğu için IP adresini bu sunucudan almaktadır.




**Şekil 3.13.** h2 bilgisayarının durumu

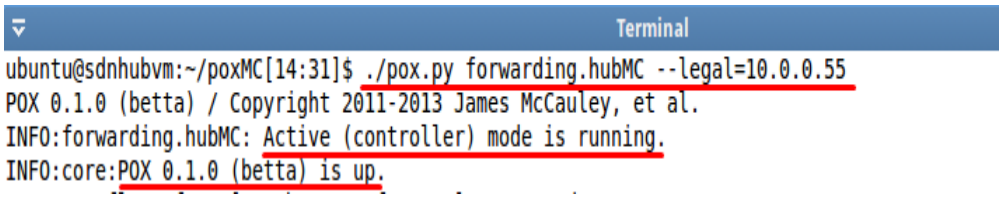
Bu bilgisayardan www.google.com adresine ping gönderildiğinde Google'a ait bir IP adresi olan adrese ping gönderdiği Şekil 3.14'te gözlemlenmektedir.

Bu durumda sahte DHCP sunucusundan IP adresi alan bilgisayarın üzerinde yapılacak olan tüm işlemler güvenirliliğini yitirmiş olacaktır.




**Şekil 3.14.** h2 bilgisayarından ping gönderilmesi

**Senaryo 2**: Bu senaryoda birinci senaryodan farklı olarak, kullanılan POX kontrolcüsüne yasal DHCP sunucusunun IP adresi tanımlanmış olup aktif modda çalıştırılmıştır. Bu IP adresi dışından başka DHCP sunucularına ait IP adreslerinden gelecek olan DHCPOFFER ve DCHPACK mesajlarının dikkate alınmaması ve bu tür IP adreslerinin engellenmesi amaçlanmıştır. Senaryo 2'de POX kontrolcüsünün başlatılması Şekil 3.15'de gösterildiği gibidir.




**Şekil 3.15.** POX kontrolcüsün aktif modda başlatılması

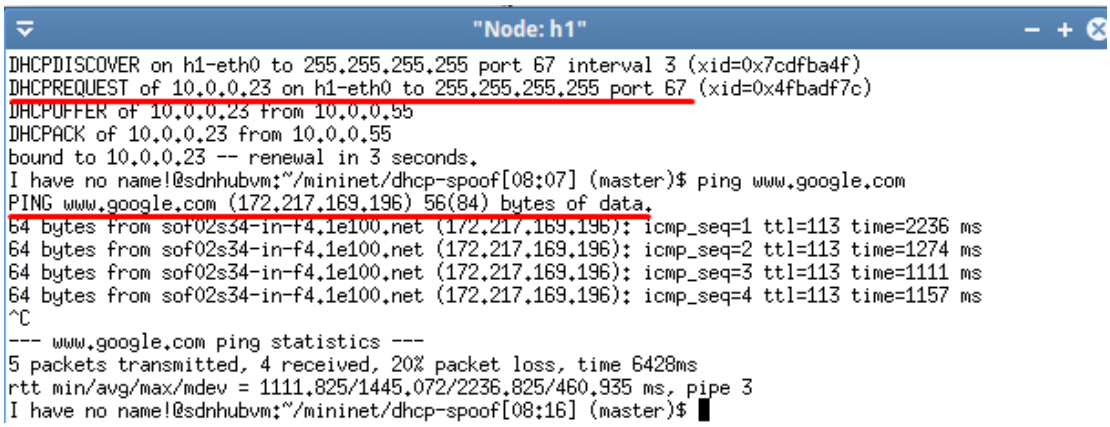
**"/pox.py forwarding.hubMC --legal=10.0.0.55"** ile başlatılan POX kontrolcüsü aktif modda çalışarak DHCP sunucularından gelen mesajları değerlendirmektedir. Burada kontrolcüye 10.0.0.55 ip adresli DHCP'nin "--legal=10.0.0.55" parametresi yasal olduğu belirtilmiştir.

Böylece kontrolcü tarafından 10.0.0.55 dışındaki tüm DHCP işlemi yapan makinalar sahte (rogue) olarak kabul edilecek ve bunların ip adresi dağıtmasına müsaade edilmeyecektir. Program çalıştırıldığında DHCP istemcisinden gelen isteklere yasal DHCP sunucusunun cevap verdiği Şekil 3.16'da görülmektedir.

| | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 2 | 0.002240000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |
| 3 | 0.519427000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 4 | 0.523014000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |
| 5 | 3.543902000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 6 | 3.561565000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |
| 7 | 4.078573000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 8 | 4.080108000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |
| 9 | 4.596890000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 10 | 4.598374000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |
| 11 | 4.818420000 | 10.0.0.23 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 12 | 5.339645000 | 10.0.0.23 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 13 | 5.454936000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 14 | 5.472513000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |
| 15 | 5.889902000 | 10.0.0.23 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 16 | 5.922557000 | 10.0.0.55 | 10.0.0.23 | DHCP | 323 | DHCP ACK |
| 17 | 5.987444000 | 10.0.0.24 | 10.0.0.55 | DHCP | 342 | DHCP Request |
| 18 | 5.988328000 | 10.0.0.55 | 10.0.0.24 | DHCP | 323 | DHCP ACK |

**Şekil 3.16.** POX kontrolcüsü aktif modda çalıştırıldığında Wireshark görüntüsü

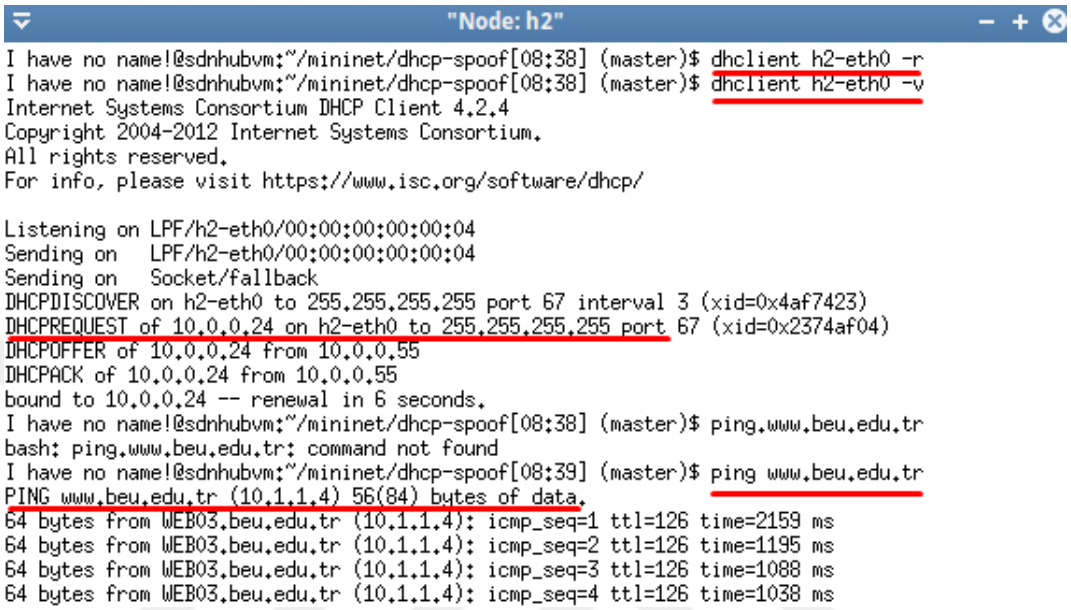
Şekil 3.17'de görüldü gibi senaryoda kontrolcüye yasal DHCP tanımlanması yapıldığı zaman aktif durumu geçen kontrolcü senaryo 1'de sahte DHCP sunucusundan IP alan h1 bilgisayarına bu kez yasal DHCP olan 10.0.0.55'ten IP aldırdığı görülmektedir.




**Şekil 3.17.** h1 bilgisayarının durumu

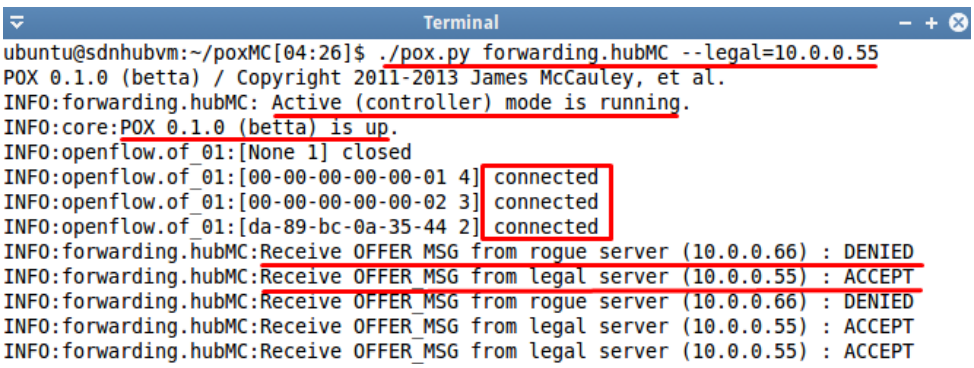
Bu bilgisayarın komut arayüzüne Mininet'te "**xterm h1**" yazılıp girildikten sonra www.google.com adresine ping atıldığında, h1 bilgisayarının Google'a ait bir IP adresine ping attığı Şekil 3.17 de görülmektedir.

Yine bu senaryoda incelenen diğer bilgisayar h2'dir. Şekil 3.18'de h2 bilgisayarının yasal DHCP sunucusunda IP aldığı görülmektedir. Bu bilgisayarın arayüzüne Mininet'te "**xterm h2**" girildikten sonra ping komutu ile Bitlis Eren Üniversitesinin web sitesi olan www.beu.edu.tr adrese ping atıldığında bu adrese ait olan bir IP adresine ping gönderildiği görülmektedir.



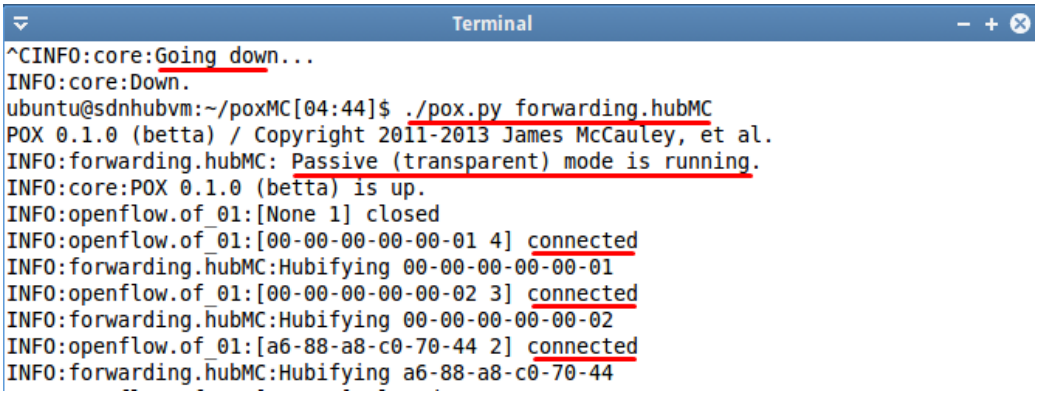

**Şekil 3.18.** h2 bilgisayarının durumu

Terminal ekranından **"sudo ./dhcp_spoof_MC.py eth0 remote"** komutu ile başlatılan program, POX kontrolcüsü ile kurduğu bağlantı Şekil 3.19'da **connected** bilgisi ile görülmektedir. Bu senaryoda kontrolcüye tanımlanan yasal DHCP IP adresi ile kontrolcü, DHCP sunucularından kendisine gelen paketleri alarak kaynak IP karşılaştırması yapmaktadır. Kaynak IP bilgisi yasal DHCP IP bilgisi ile eşleştiğinde Şekil 3.19'te görüldüğü gibi "ACCEPT" olacak şekilde işlem devam edecek, eşleşmediği zaman ise "DENIED" olacak şekilde işlem engellenecektir.



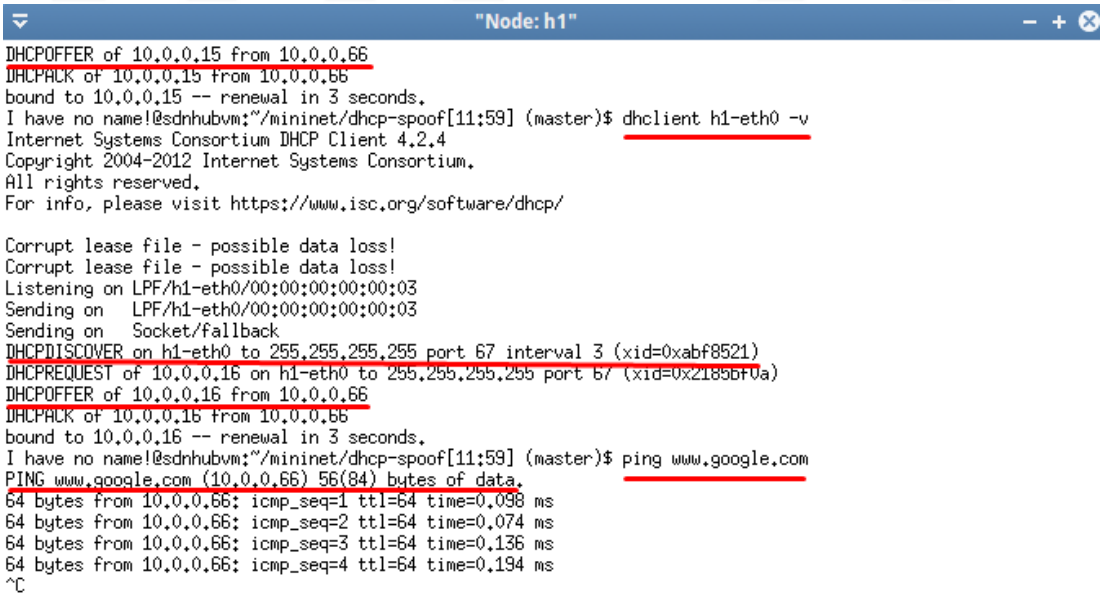

**Şekil 3.19.** POX kontrolcüsünün IP eşleştirmesi

Senaryo 1'deki durumu tekrar kontrol etmek için POX kontrolcüsü Şekil 3.20'de gösterildiği gibi durdurulmuştur.




**Şekil 3.20.** Pasif mod denemesi

Kontrolcü **".pox.py forwarding.hub.MC"** komutu ile tekrar çalıştırılmıştır. Bilgisayarlardan h1'in IP adresi **"dhclient h1-eth0 –v"** komutu ile yeni IP adresi isteği için çalıştırılmıştır. Bu durumda h1 bilgisayarı sahte DHCP sunucusundan yeni bir IP adresi aldığı, www.google.com adresine ping gönderildiğinde ise kendi DNS sunucusu olan 10.0.0.66 adresine yönlendirdiği Şekil 3.21' de gözlemlenmiştir.




**Şekil 3.21.** Bilgisayar h1 IP denemesi

# 4. SONUÇ

Son yıllarda teknolojinin farklı alanlarda bütünleşik olarak kullanılması ile birlikte ağlar üzerinde olan yük artmıştır. Geleneksel bilgisayar ağlarının sürekli olarak ekleme mantığıyla büyümesi ağın yönetilebilirliğini zorlaştırmış ve bu var olan yapının içinden çıkılmaz bir hal almasına neden olmuştur. Ağ cihazlarındaki yeniliklerin, cihazlarda kullanılan teknolojilere bağımlı olarak kalması, yeni teknolojilere olan ihtiyaçları yeni cihazlara bağımlı hale getirmektedir. Bu durum kullanıcı tarafında büyük maliyet ve zaman kayıplarına neden olmaktadır. Ağ yöneticilerini ve kullanıcılarını benzer sorunlardan bir nebzede olsa kurtarmak için ağlarda yazılım kullanılması uygun görülmüştür. Yazılım tanımlı ağ paradigması, geleneksel ağların statik yapılarını dinamik bir yapıya kavuşturmak, ağ yönetimini kolaylaştırmak ve ağ güvenliğini artırmak, ağ cihazlarına yazılım ile işlevsellik kazandırarak kullanıcıları cihaz bağımlı durumdan kurtarmak amacıyla ortaya atılmıştır.

Bu tez çalışmasında Mininet ağ benzetim ortamında oluşturulmuş olan ağ şeması üzerinde ağ cihazlarına otomatik IP dağıtan DHCP sunucusunun güvenliği için bir çalışma yapılmıştır. Bu çalışmada yazılım tanımlı ağların kontrol ünitesinde bulunan kontrolcü üzerine tanımlanan yasal DHCP adresi ile kontrolcü cihazına gelecek olan DHCP paketlerinde filtreleme –eşleştirme mantığı kullanılmıştır. Böylece IP adresi isteğinde bulunan bilgisayarlara cevap verecek olan DHCP sunucularından gelen paketler ilk önce kontrolcüden geçmektedir.

Kontrolcü aktif modda çalıştığında, ağda bulunan yasal veya sahte DHCP sunucularından DHCP istemcilere giden DHCPACK ve DHCP istemcilerden DHCP sunuculara giden DHCPREQUEST mesajlarını yakalayarak kontrolden geçirir. Böylece aktif modda kendisine tanımlanan IP adresi üzerinden mesajların yasal DHCP sunucusundan geldiğine ve ona gitmesi gerektiğini kontrol eder. Bu şekilde yapılan bir kontrol mekanizması ile sahte DHCP sunucularından gelen DHCPACK mesajlarının istemci cihazlara ulaşması engellenmiş olur.

Gelecek çalışmalarda, YTA ağlarında sahte DHCP sunucusunun komple ağ içerisinde nerede olduğu tespit edilecek ve ağ içerisinde yeri belirlenen sahte DHCP sunucusunun portu kapatılarak devre dışı kalması sağlanacaktır. Ayrıca sahte DHCP tespit edilmesi durumu ve ilgili bilgiler ağ yöneticisine iletilecek şekilde yapılandırmalar gerçekleştirilecektir.

## 5. KAYNAKLAR


[1] https://wearesocial.com/global-digital-report-2019 (Erişim Tarihi:29/06/2020).

[2] Aydın H, Görmüş S, Ulutaş G, 2018. Nesnelerin İnterneti Teknolojisi İçin Güvenlik: Var Olan Mekanizmalar, Protokoller Ve Yaşanılan Zorlukların Araştırılması. Gazi Üniversitesi Mühendislik-Mimarlık Fakültesi Derg, 2018(2018):1247–1272.

[3] Polat O, 2017. Yazılım Tanımlı Ağlarda DOS Saldırılarının Tespiti ve Engellenmesi. Yüksel Lisans Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, Ankara.

[4] Göksel N, 2019. Yazılım Tanımlı Ağlarda Hizmet Dışı Bırakma Saldırılarının Tespiti ve Engellenmesi. Yüksek Lisans Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, Ankara.

[5] Lali MI, Mustafa RU, Ahsan F, Nawaz MS, Aslam W, 2017. Performance Evaluation of Software Defined Networking vs. Traditional Networks. A quartelry Int Sci J, 54(1):16–22.

[6] Singh AK, Srivastava S, 2018. A survey and classification of controller placement problem in SDN Ashutosh. Int J Netw Management, 28(3), e2018.

[7] Toprak C, Turker C, Erman AT, 2018. Detection of DHCP Starvation Attacks in Software Defined Networks: A Case Study. 2018 3rd International Conference on Computer Science and Engineering (UBMK), 20-23 Sept. 2018, Sarajevo, Bosnia-Herzegovina, s:636-641.

[8] Hubballi N, Tripathi N, 2017. A Closer Look into DHCP Starvation Attack in Wireless Networks. Comput Secur, 65:387–404.

[9] https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-jero.pdf (Erişim tarihi: 06/05/2020).

[10] Cox JH, Clark RJ, Owen HL, 2016. Leveraging SDN to Improve the Security of DHCP. SDN-NFV Security '16: Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, March 2016 New York, s: 35–38.

[11] Wang J-L, Chen Y-C, 2017. An SDN-based Defensive Solution against DHCP Attacks in the Virtualization Environment. In: 2017 IEEE Conf. Dependable Secur. Comput. IEEE, s:529–530.

[12] Kreutz D, Ramos FMV, Verissimo P, 2013. Towards Secure and Dependable Software-Defined Networks. In: Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw. - HotSDN '13. ACM Press, New York, New York, USA, s:55-60.

[13] https://www.opennetworking.org/ (Erişim Tarihi:30/05/2020).

[14] Feamster N, Rexford J, Zegura E, 2014. The Road to SDN. ACM SIGCOMM Comput Commun Rev, 44(2):87–98.

[15] Li W, Meng W, Kwok LF, 2016. A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures. J Netw Comput Appl, 68:126–139.

[16] Lara A, Kolasani A, Ramamurthy B, 2014. Network Innovation using OpenFlow: A survey. IEEE Commun Surv Tutorials, 16(1):493–512.

[17] Shalimov A, Zimarina D, Pashkov V, 2013. Advanced Study of SDN/ OpenFlow controllers. In: ACM Int. Conf. Proceeding Ser. s:1–6.

[18] https://www.opennetworking.org/ (Erişim Tarihi:30/05/2020).

[19] Sócrates-Dantas J, Careglio D, Perelló J, Silveira RM, Ruggiero WV, Solè-Pareta J, 2014. Challenges and requirements of a control plane for elastic optical networks. Comput Networks, 72:156–171.

[20] Jammal M, Singh T, Shami A, Asal R, Li Y, 2014. Software defined networking: State of the art and research challenges. Comput Networks, 72:74–98.

[21] http://www.cables-solutions.com/whats-openflow-switch-how-it-works.html (Erişim Tarihi:01/06/2020).

[22] Huang S, Griffioen J, Calvert KL, 2014. Network Hypervisors: Enhancing SDN Infrastructure. Comput Commun, 46:87–96.

[23] https://tools.ietf.org/html/rfc5810 (Erişim Tarihi:30/05/2020).

[24] https://tools.ietf.org/html/draft-smith-opflex-00 (Erişim Tarihi:30/05/2020).

[25] Song H, 2013. Protocol-oblivious forwarding. In: Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw. - HotSDN '13. ACM Press, New York, New York, USA, s:127–132.

[26] Tarnaras G, Haleplidis E, Denazis S, 2015. SDN and ForCES based optimal network topology discovery. In: Proc. 2015 1st IEEE Conf. Netw. Softwarization. IEEE, s:1–6.

[27] Ciciloğlu M, Çalhan A, 2016. Yazılım Tanımlı Ağlarda Denetleyiciler ( Controllers In Yazılım Tanımlı Ağlarda Denetleyiciler Controllers In Software-Defined Networks. (November 2016):2–7.

[28] https://www.networkurge.com/2017/06/how-sdn-works.html (Erişim Tarihi:19/06/2020).

[29] http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/ (Erişim Tarihi:02/06/2020).

[30] Cicioğlu M, Çalhan A, 2017. Yazılım tanımlı ağlar–YTA. Karaelmas Fen ve Mühendislik Derg, 7(2):684–695.

[31] https://www.dcs.bbk.ac.uk/~PTW/teaching/IWT/network-layer/notes.html (Erişim Tarihi:10/06/2020).

[32] https://www.slideshare.net/kingstonsmiler/tutorial-on-sdn-and-openflow (Erişim

Tarihi:02/06/2020).

[33] Huang S, Griffioen J, Calvert KL, 2014. Network Hypervisors: Enhancing SDN Infrastructure. Comput Commun, 46:87–96.

[34] Kreutz D, Ramos FM V., Esteves Verissimo P, Esteve Rothenberg C, Azodolmolky S, Uhlig S, 2015. Software-Defined Networking: A Comprehensive Survey. Proc IEEE, 103(1):14–76.

[35] Vijay Tijare P, Vasudevan D, 2016. The Northbound APIs Of Software Defined Networks. © Int J Eng Sci Res Technol, 501(10):501–513.

[36] Santo de Oliveira RLS, Schweitzer CM, Shinoda AA, Ligia Rodrigues Prete, 2014. Using Mininet for emulation and prototyping Software-Defined Networks. 2014 IEEE Colomb. Conf. 4-6 June 2014, Commun. Comput. IEEE, s:1–6.

[37] Keti F, Askar S, 2015. Emulation of Software Defined Networks Using Mininet in Different Simulation Environments. In: 2015 6th Int. Conf. Intell. Syst. Model. Simul. IEEE, s:205–210.

[38] Huang T, Jeyakumar V, Lantz B, Connor BO, Winstein K, Sivaraman A, 2014. Teaching Computer Networking with Mininet. .

[39] https://openflow.stanford.edu/display/ONL/POX+Wiki.html (Erişim Tarihi:06/06/2020).

[40] Shaoqiang Wang, DongSheng Xu, ShiLiang Yan, 2010. Analysis and Application of Wireshark in TCP/IP Protocol Teaching. In: 2010 Int. Conf. E-Health Netw. Digit. Ecosyst. Technol. IEEE, s:269–272.

[41] https://www.wireshark.org/ (Erişim Tarihi:30/05/2020).

[42] https://wmaraci.com/nedir/wireshark (Erişim Tarihi:31/05/2020).

[43] Tashakkori R, 2010. Improving believability of simulated characters. .

[44] https://bilenlerkabilesi.com/virtualbox-nedir-nasil-kullanilir/ (Erişim Tarihi:31/05/2020).

[45] Virtualbox R, 2013. Oracle VM VirtualBox. :1–362.

[46] https://www.python.org/ (Erişim Tarihi:31/05/2020).

[47] https://www.tiobe.com/tiobe-index/ (Erişim Tarihi:03/06/2020).

[48] https://medium.com/yazılıma-dair/pyhton-programlama-dili-neden-giderek-popülaritesini-arttırıyor-ab14b1da39c6 (Erişim Tarihi:31/05/2020).

[49] https://www.linux.org/ (Erişim Tarihi:05/06/2020).

[50] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5997689 (Erişim Tarihi:06/06/2020).

[51] https://www.omnisecu.com/tcpip/dhcp-dynamic-host-configuration-protocol-how-dhcp-works.php (Erişim Tarihi:31/05/2020).

[52] Ertam F, Avcı HDE, 2017. Büyük Ağlarda DHCPSNOOPING Etkisinin İncelenmesi. In: 7th Int. Adv. Technol. Symp. (IATS'13), 30 October-1 Novemb. 2013, Istanbul, Turkey. s:566–569.

[53] https://informatics.buzdo.com/specific/tcp-ip-1.htm (Erişim Tarihi:10/06/2020).

[54] http://www.tcpipguide.com/free/t_DHCPMessageFormat.htm (Erişim Tarihi:11/06/2020).

[55] Akın, G. (2008). DHCP Servisine Yeni Bir Bakış. *inet-tr'08*, 41.

[56] Issac B, 2014. Secure ARP and Secure DHCP Protocols to Mitigate Security Attacks. Int J Netw Secur, 8(2):107–118.

[57] Dinu DD, Togan M, 2014. DHCP server authentication using digital certificates. In: 2014 10th Int. Conf. Commun. IEEE, s:1–6.

[58] Cox JH, Clark RJ, Owen HL, 2016. Leveraging SDN for ARP Security. In: SoutheastCon 2016. IEEE, s:1–8.

[59] Akın, G. (2008). DHCP Servisine Yeni Bir Bakış. *inet-tr'08*, 41.

[60] https://en.wikipedia.org/wiki/Udhcpc (Erişim Tarihi:15/06/2020).

[61] https://en.wikipedia.org/wiki/Dnsmasq (Erişim Tarihi:15/06/2020).

[62] https://www.ettercap-project.org/ (Erişim Tarihi:15/06/2020).

[63] https://tools.kali.org/sniffingspoofing/dnschef (Erişim Tarihi:15/06/2020).

# 6. EKLER

## EK 1. Pox SDN kontrolcüde çalıştırılan hubMC.py dosyası kaynak kodları

```
#There are actually two mode in here: a active one and a passive one.

import sys
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr

from pox.lib.addresses import IPAddr, EthAddr
from pox.lib.util import dpid_to_str, str_to_dpid
from pox.lib.revent import EventMixin, Event

import pox.lib.packet as pkt
from pox.lib.packet import *

log = core.getLogger()


def launch (legal = ""):

  def _handle_ConnectionUp (event):
    """
    Be a passive controller. Telling every connected switch to flood all packets
    """
    msg = of.ofp_flow_mod()
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    event.connection.send(msg)
    log.info("Hubifying %s", dpidToStr(event.dpid))


  def _handle_PacketIn (event):
    """
    Be a active controller. Check every incoming dhcp packet for prevent the
dhcp_spoof attack.
    """
    isLegal = True
    if prevent:
      p = event.parsed.find("dhcp")
      if p is None:
        """log.info("NO dhcp parse (p is null)")"""
      else:
        t = p.options.get(p.MSG_TYPE_OPT)
        if t is None:
          """log.info("NO p.MSG_TYPE_OPT!.")"""
        else:
          ipp = event.parsed.find('ipv4')
          if t.type == p.ACK_MSG:
            pmsg = "ACK_MSG"
          if t.type == p.OFFER_MSG:
            pmsg = "OFFER_MSG"
```

```
50          if t.type == p.OFFER_MSG and ipp.srcip != legal:
51            log.info("Receive %s from rogue server (%s) : DENIED", pmsg, ipp.srcip)
52            isLegal = False
53            event.halt = True
54          elif t.type == p.OFFER_MSG and ipp.srcip == legal:
55            log.info("Receive %s from legal server (%s) : ACCEPT", pmsg, ipp.srcip)
56          else:
57            isLegal = True
58
59    if isLegal:
60      msg = of.ofp_packet_out()
61      msg.data = event.ofp
62      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
63      event.connection.send(msg)
64
65  """
66  Main code
67  """
68  if legal:
69    prevent = True
70  else:
71    prevent = False
72
73  if prevent:
74    core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
75    log.info(" Active (controller) mode is running.")
76  else:
77    core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
78    log.info(" Passive (transparent) mode is running.")
```

## EK 2. Mininet üzerinde yürütülen topoloji dosyası (dhcp_spoof_MC.py) kaynak kodları

```
1  #!/usr/bin/python2
2  #
3  # An evil host near the client in the network tries to respond
4  # to DHCP requests faster than the real DHCP server and point
5  # hosts to its own DNS server allowing it to point people to
6  # phishing sites.
7  #
8  # Based on:
9  #   https://bitbucket.org/lantz/cs144-dhcp/raw/master/dhcp.py
10 #
11 # Dependencies:
12 #   busybox (provides udhcd)
13 #   dhclient
14 #   dnsmasq
15 #   ettercap
16 #   iptables
17 #   ebtables
18 #
19 # Software used for which tasks:
```

```
20 #    busybox udhcd - the normal DHCP server
21 #    dhclient - the DHCP client
22 #    dnsmasq - the DNS server commonly used for caching on routers
23 #    ettercap - the tool to spoof the real DHCP ACK
24 #    dnschef - Python DNS server pointed to by spoofed DHCP
25 #    iptables - filter packets at the network layer
26 #    ebtables - filter packets at the link layer
27 #
28 # Note: in all likelyhood, this won't work on another Linux distribution
29 # without some modification. For example, you'll need to change 'python2' to
30 # be just 'python'. You might have to use OVSBridge instead of LinuxBridge
31 # if `mn --switch lxbr --test pingall' drops all your pings.
32
33 # Future work
34 #   - mark in ebtables, act on mark in userspace
35 #   - block spoofing of server's IP by looking at MAC with ebtables
36 #   - see if making the evil host the gateway allows more packet sniffing
37
38 import os
39 import sys
40 import time
41 import shutil
42 from mininet.net import Mininet
43 from mininet.topo import Topo
44 from mininet.link import TCLink
45 from mininet.cli import CLI
46 from mininet.log import setLogLevel, info
47 #from mininet.nodelib import LinuxBridge
48 from mininet.node import OVSBridge
49 from mininet.node import OVSSwitch,RemoteController,DefaultController
50 from nat import connectToInternet, stopNAT
51
52 # Is network manager running?
53 def isNetworkManagerRunning():
54     return os.system("pgrep NetworkManager &>/dev/null") == 0
55
56 # Topology:
57 #   evilDHCP -- switch1 -- switch0 -- switch2 -- legalDHCP
58 #                  |                    |
59 #               client1              client2
60 class DHCPTopo(Topo):
61     def __init__(self, *args, **kwargs):
62         Topo.__init__(self, *args, **kwargs)
63
64         # Make sure that these are private mounts in only the desired process,
65         # so we don't end up hiding /etc and /var for the duration of this
66         # program. To make it private, it must be mounted, so bind it to itself
67         # and then make it private.
68         os.system('mount --bind /etc /etc')
69         os.system('mount --make-rprivate /etc')
70         os.system('mount --bind /var /var')
71         os.system('mount --make-rprivate /var')
72
73         # Private directories for client1
74         self.client_etc1 = '/tmp/etc_h1'
75         self.client_var1 = '/tmp/var_h1'
76         self.createDirs([self.client_etc1, self.client_var1])
```

```
77          private_h1 = [('/etc', self.client_etc1), ('/var', self.client_var1)]
78          # Private directories for client2
79          self.client_etc2 = '/tmp/etc_h2'
80          self.client_var2 = '/tmp/var_h2'
81          self.createDirs([self.client_etc2, self.client_var2])
82          private_h2 = [('/etc', self.client_etc2), ('/var', self.client_var2)]
83
84          # inNamespace is needed so that we get a private network and mount
85          # namespace for processes run on this host
86          client1 = self.addHost('h1',
87                                 ip='10.0.0.11/24',
88                                 privateDirs=private_h1,
89                                 inNamespace=True,
90                                 privateLogDir=True,
91                                 privateRunDir=True,
92                                 inMountNamespace=True,
93                                 inPIDNamespace=True,
94                                 inUTSNamespace=True)
95          client2 = self.addHost('h2',
96                                 ip='10.0.0.12/24',
97                                 privateDirs=private_h2,
98                                 inNamespace=True,
99                                 privateLogDir=True,
100                                privateRunDir=True,
101                                inMountNamespace=True,
102                                inPIDNamespace=True,
103                                inUTSNamespace=True)
104
105         # The rest don't need any special setup
106         switch1 = self.addSwitch('s1')
107         switch2 = self.addSwitch('s2')
108         switch0 = self.addSwitch('s0')
109         dhcp = self.addHost('dhcp', ip='10.0.0.55/24')
110         evil = self.addHost('evil', ip='10.0.0.66/24')
111         self.addLink(client1, switch1)
112         self.addLink(client2, switch2)
113         self.addLink(evil, switch1)
114         #self.addLink(evil, switch1, bw=10, delay='500ms')
115         self.addLink(dhcp, switch2)
116         #self.addLink(dhcp, switch2, bw=10, delay='500ms')
117         self.addLink(switch1, switch0)
118         #self.addLink(switch1, switch0, bw=10, delay='500ms')
119         self.addLink(switch0, switch2)
120         #self.addLink(switch0, switch2, bw=10, delay='500ms')
121
122     def __enter__(self):
123         return self
124     def __exit__(self, type, value, traceback):
125         shutil.rmtree(self.client_etc1)
126         shutil.rmtree(self.client_var1)
127         shutil.rmtree(self.client_etc2)
128         shutil.rmtree(self.client_var2)
129
130         # Try to reduce the number in /proc/mounts, but if something is running
131         # in one of these, then it won't unmount. It doesn't really matter
132         # since we just mounted it to itself.
133         os.system('umount /etc &>/dev/null')
```

```
134         os.system('umount /var &>/dev/null')
135     # Create a list of directories
136     def createDirs(self, dirs):
137         for d in dirs:
138             try:
139                 os.makedirs(d)
140             except OSError:
141                 pass
142 
143 # DHCP server functions and data
144 DHCPTemplate = """
145 start            10.0.0.10
146 end              10.0.0.90
147 option   subnet   255.255.255.0
148 option   domain   local
149 option   lease    60   # seconds
150 """
151 
152 # Output network addresses and names for assisting in Wireshark analysis
153 def outputNet(l):
154     for h in l:
155         print "Host:", h.name, "IP:", h.IP(), "MAC:", h.MAC()
156 
157 # We need a separate /etc/resolv.conf for each host so that they can
158 # resolve DNS names with the DNS server they received from DHCP. Might as
159 # well also not mess up the real udhcpd or dhclient lease files, especially
160 # when they might actually be in use on the real host.
161 def setupPrivateFS(host, etc, var):
162     host.cmd('touch ', etc + '/resolv.conf')
163     host.cmd('mkdir -p', var + '/lib/misc')
164     host.cmd('mkdir -p', var + '/lib/dhclient')
165     host.cmd('mkdir -p', var + '/lib/dhcp')
166     host.cmd('mkdir -p', var + '/run')
167     host.cmd('touch ', var + '/lib/misc/udhcpd.leases')
168     host.cmd('touch ', var + '/lib/dhclient/dhclient.leases')
169     host.cmd('touch ', var + '/lib/dhcp/dhclient.leases')
170 
171 # Output when we get an IP from the DHCP server
172 def waitForIP(host):
173     info('*', host, 'waiting for IP address')
174     while True:
175         host.defaultIntf().updateIP()
176         if host.IP():
177             break
178         info('.')
179         time.sleep(1)
180     info('\n')
181     info('*', host, 'is now at',host.IP(),'and is using', host.cmd('grep
182 nameserver /etc/resolv.conf'))
183     info('\n')
184 
185 # DHCP Client
186 def startDHCPclient(host):
187     intf = host.defaultIntf()
188     host.cmd('touch /tmp/dhclient.conf', intf)
189     host.cmd('dhclient -v -d -r', intf)
190 
```

```
191     host.cmd('dhclient -v -d -cf /tmp/dhclient.conf 1> /tmp/dhclient.log 2>&1',
192 intf, '&')
193 def stopDHCPclient(host):
194     host.cmd('kill %dhclient')
195     host.cmd('rm /tmp/dhclient.log')
196     host.cmd('rm /tmp/dhclient.conf')
197
198 # Good DHCP Server
199 def makeDHCPconfig(filename, intf, gw, dns):
200     config = (
201         'interface %s' % intf,
202         DHCPTemplate,
203         'option router %s' % gw,
204         'option dns %s' % dns,
205         '')
206     with open(filename, 'w') as f:
207         f.write('\n'.join(config))
208
209 def cleanDHCPconfig(host, filename):
210     host.cmd('rm ', filename)
211
212 def startGoodDHCPserver(host, gw, dns):
213     info('* Starting good DHCP server on', host, 'at', host.IP(), '\n')
214     dhcpConfig = '/tmp/%s-udhcpd.conf' % host
215     makeDHCPconfig(dhcpConfig, host.defaultIntf(), gw, dns)
216     #host.cmd('busybox udhcpd -f', dhcpConfig, '1>/tmp/%s-dhcp.log 2>&1  &' %
217 host)
218     host.cmd('udhcpd -f', dhcpConfig, '1>/tmp/%s-dhcp.log 2>&1  &' % host)
219
220 def stopGoodDHCPserver(host):
221     info('* Stopping good DHCP server on', host, 'at', host.IP(), '\n')
222     host.cmd('kill %udhcpd')
223     dhcpConfig = '/tmp/%s-udhcpd.conf' % host
224     cleanDHCPconfig(host, dhcpConfig)
225
226 # Bad DHCP Server
227 def startBadDHCPserver(host, gw, dns):
228     info('* Starting bad DHCP server on', host, 'at', host.IP(), '\n')
229     host.cmd('ettercap -T -M dhcp:10.0.0.10-90/255.255.255.0/%s -a etter.conf
230 &>/tmp/ettercap.log &' % host.IP())
231
232 def stopBadDHCPserver(host):
233     info('* Stopping bad DHCP server on', host, 'at', host.IP(), '\n')
234     host.cmd('kill %ettercap')
235     host.cmd('rm /tmp/ettercap.log')
236
237 # Yersinia doesn't appear to execute DHCP attack from the daemon
238 #def startBadDHCPserver(host, gw, dns):
239 #    info('* Starting bad DHCP server on', host, 'at', host.IP(), '\n')
240 #    host.cmd('yersinia -D')
241 #    host.cmd('expect setup_yersinia localhost 12000 %s %s 10.0.0.10 10.0.0.90 7
242 10 255.255.255.0 local &>/tmp/yersinia.log &'
243 #            % (host.defaultIntf(), host.IP()))
244 #
245 #def stopBadDHCPserver(host):
246 #    info('* Stopping bad DHCP server on', host, 'at', host.IP(), '\n')
247 #    host.cmd('killall yersinia')
```

```
248 #    host.cmd('rm /tmp/yersinia.log')
249
250 # Good DNS Server
251 def startGoodDNSserver(host):
252     info('* Starting good DNS server', host, 'at', host.IP(), '\n')
253     host.cmd('dnsmasq -k -x /tmp/dnsmasq.pid -C - 1>/tmp/dns-good.log 2>&1
254 </dev/null &')
255
256 def stopGoodDNSserver(host):
257     info('* Stopping good DNS server', host, 'at', host.IP(), '\n')
258     host.cmd('kill $(cat /tmp/dnsmasq.pid)')
259     host.cmd('rm /tmp/dns-good.log')
260     host.cmd('rm /tmp/dnsmasq.pid')
261
262 # Bad DNS Server
263 def startBadDNSserver(host):
264     info('* Starting bad DNS server', host, 'at', host.IP(), '\n')
265     host.cmd('python dnschef.py --file=dnschef.ini -i 10.0.0.66 1>/tmp/dns-
266 bad.log 2>&1 &')
267
268 def stopBadDNSserver(host):
269     info('* Stopping bad DNS server', host, 'at', host.IP(), '\n')
270     host.cmd('kill %dnschef')
271     host.cmd('rm /tmp/dns-bad.log')
272
273 # Prevent this attack from being possible
274 #
275 # Filter out all DHCP ACKs and Offers coming from the wrong MAC address
276 def startSwitchBlocking(host, realMAC):
277     info('* Starting DHCP blocking on', host, 'at', host.IP(), 'all but from',
278 realMAC, '\n')
279     host.cmd('ebtables -I FORWARD -s \! %s --protocol ipv4 --ip-proto udp --ip-
280 dport 68 -j DROP' % realMAC)
281
282 def stopSwitchBlocking(host):
283     host.cmd('ebtables -F')
284
285 # Instead of just blocking DHCP packets from the wrong MAC, detect those and
286 # then attack the evil server
287 def startSwitchCounterattack(host, realMAC):
288     pass
289     # not finished...
290
291 def stopSwitchCounterattack(host):
292     pass
293
294 # Create a config file before running dhclient that'll specify the MAC address
295 # of the real DHCP server so that the evil one won't even receive the packets
296 # alerting it that somebody is connecting. It does receive the real DHCP server's
297 # offer and ACK, but since those are going to FF:FF:FF:FF:FF:FF, ettercap doesn't
298 # know the MAC of the connecting client, so it can't spoof the ACK.
299 def clientBlockingConfig(host, realMAC):
300     config = (
301         'interface "%s" {' % host.defaultIntf(),
302         'anycast-mac ethernet %s;' % realMAC,
303         '}',
304         '')
```

```
305     with open('/tmp/dhclient.conf', 'w') as f:
306         f.write('\n'.join(config))
307 
308 # Resolve the problem by not even using DHCP and instead just assigning a
309 # static IP and manually setting up the gateway and subnet mask.
310 def setupStaticIP(host, ip, mask, gw, dns):
311     # Setup static IP and gateway, the subnet route is automatically added when
312     # you specify the subnet mask
313     host.cmd('ip addr add %s/%s dev %s' % (ip, mask, host.defaultIntf()))
314     host.cmd('ip route add default via %s metric 100 dev %s' % (gw,
315 host.defaultIntf()))
316 
317     # Tell it which DNS server to use. We can't actually use python to do this
318     # since we need to create the file in the process that has a private /etc
319     # mounted.
320     host.cmd('echo "nameserver %s" > /etc/resolv.conf' % dns)
321 
322 def usage():
323     print """Usage: python2 dhcp_spoof.py interface [preventionTechnique]
324 
325 Interface:
326     the interface on which to get Internet access, e.g. eth0
327 
328 Prevention Technique:
329     0 - none, allow the attack to happen (default)
330     1 - ebtables blocking all but correct DHCP packets
331     2 - dhclient anycast-mac
332     3 - static IP
333 
334 Not yet implemented:
335     4 - counter attack, DHCP starve the attacker
336     5 - ebtables block IP spoofing by looking at MAC
337     6 - use snort (IPS) on switch to detect attacker
338 """
339 
340 if __name__ == '__main__':
341     setLogLevel('info')
342 
343     # Parse arguments
344     controller = None
345     if "remote" in sys.argv:
346         controller = RemoteController
347     if len(sys.argv) >= 2:
348         inetIntf = sys.argv[1]
349         prevent = 0
350     elif len(sys.argv) >= 3:
351         inetIntf = sys.argv[1]
352         prevent = int(sys.argv[2])
353 
354         if prevent < 0 or prevent > 6:
355             print "Error: invalid technique number"
356             sys.exit(1)
357         elif prevent > 3:
358             print "Error: technique not yet implemented"
359             sys.exit(1)
360     else:
361         usage()
```

```
362         sys.exit(1)
363
364     # Setup the virtualized attack
365     with DHCPTopo() as topo:
366         net = Mininet(topo=topo, link=TCLink, switch=OVSSwitch,
367 controller=controller, autoSetMacs=True)
368         h1, h2, dhcp, evil, sw1, sw2, sw0 = net.get('h1', 'h2', 'dhcp', 'evil',
369 's1', 's2', 's0')
370         setupPrivateFS(h1, topo.client_etc1, topo.client_var1)
371         setupPrivateFS(h2, topo.client_etc2, topo.client_var2)
372         rootnode = connectToInternet(net, inetIntf, 's0')
373
374
375         try:
376             raw_input("Press return after you've started wireshark on s1")
377
378             if prevent == 2:
379                 clientBlockingConfig(h1, dhcp.MAC())
380                 clientBlockingConfig(h2, dhcp.MAC())
381
382             # Set up a good DHCP and DNS server
383             startGoodDHCPserver(dhcp, gw=rootnode.IP(), dns=dhcp.IP())
384             startGoodDNSserver(dhcp)
385
386             if prevent == 1:
387                 startSwitchBlocking(sw1, dhcp.MAC())
388                 startSwitchBlocking(sw2, dhcp.MAC())
389                 startSwitchBlocking(sw0, dhcp.MAC())
390             elif prevent == 4:
391                 startSwitchCouterattack(sw1, dhcp.MAC())
392                 startSwitchCouterattack(sw2, dhcp.MAC())
393                 startSwitchCouterattack(sw0, dhcp.MAC())
394
395             # Start up evil DHCP and DNS server
396             startBadDHCPserver(evil, gw=rootnode.IP(), dns=evil.IP())
397             startBadDNSserver(evil)
398             info('\n')
399             h1.cmd('ifconfig', h1.defaultIntf(), '0')
400             h2.cmd('ifconfig', h2.defaultIntf(), '0')
401
402             if prevent == 3:
403                 # We don't have to use DHCP if we already know a configuration
404                 setupStaticIP(h1, h1.IP(), 8, rootnode.IP(), dhcp.IP())
405                 setupStaticIP(h2, h2.IP(), 8, rootnode.IP(), dhcp.IP())
406             else:
407                 # Wait for ettercap to find the hosts
408                 time.sleep(8)
409                 # Let the client connect
410                 startDHCPclient(h1)
411                 waitForIP(h1)
412                 startDHCPclient(h2)
413                 waitForIP(h2)
414
415             # Try to ping Google from the client
416             print "Pinging google.com"
417             h1.cmdPrint('ping -c 2 -w 1 google.com')
418             info('\n')
```

```
419             h2.cmdPrint('ping -c 2 -w 1 google.com')
420             info('\n')
421 
422             # Output the network
423             outputNet([h1, h2, dhcp, evil])
424             print
425 
426             # Debug
427             print "Dropping to CLI, exit to cleanup virtual network."
428             CLI(net)
429 
430         except KeyboardInterrupt:
431             print
432             print "Exiting..."
433 
434         finally:
435             # Clean up everything
436             if prevent == 1:
437                 stopSwitchBlocking(sw1)
438                 stopSwitchBlocking(sw2)
439                 stopSwitchBlocking(sw0)
440             elif prevent == 4:
441                 stopSwitchCounterattack(sw1)
442                 stopSwitchCounterattack(sw2)
443                 stopSwitchCounterattack(sw0)
444 
445             stopBadDNSserver(evil)
446             stopBadDHCPserver(evil)
447             stopGoodDNSserver(dhcp)
448             stopGoodDHCPserver(dhcp)
449             stopDHCPclient(h1)
450             stopDHCPclient(h2)
451             stopNAT(rootnode)
                net.stop()
```

## EK 3. Sahte DNS için konfigürasyon dosyası (dnschef.ini) kaynak kodları

```
[A]     # Queries for IPv4 address records
*.thesprawl.org=192.0.2.1
*.google.com=10.0.0.66
*.*=10.0.0.67
[AAAA]  # Queries for IPv6 address records
*.thesprawl.org=2001:db8::1
[MX]    # Queries for mail server records
*.thesprawl.org=mail.fake.com
[NS]    # Queries for mail server records
*.thesprawl.org=ns.fake.com
[CNAME] # Queries for alias records
*.thesprawl.org=www.fake.com
[TXT]   # Queries for text records
*.thesprawl.org=fake message
```

```
[PTR]
*.2.0.192.in-addr.arpa=fake.com
[SOA]
; FORMAT: mname rname t1 t2 t3 t4 t5
*.thesprawl.org=ns.fake.com. hostmaster.fake.com. 1 10800 3600 604800 3600
[NAPTR]
; FORMAT: order preference flags service regexp replacement
*.thesprawl.org=100 10 U E2U+sip !^.*$!sip:customer-service@fake.com! .
[SRV]
; FORMAT: priority weight port target
*.*.thesprawl.org=0 5 5060 sipserver.fake.com
[DNSKEY]
; FORMAT: flags protocol algorithm base64(key)
*.thesprawl.org=256                                3                                5
AQPSKmynfzW4kyBv015MUG2DeIQ3Cbl+BBZH4b/0PY1kxkmvHjcZc8nokfzj31GajIQKY+5CptLr3buXA10hWq
TkF7H6RfoRqXQeogmMHfpftf6zMv1LyBUgia7za6ZEzOJBOztyvhjL742iU/TpPSEDhm2SNKLijfUppn1UaNvv
4w==
[RRSIG]
; FORMAT: covered algorithm labels labels orig_ttl sig_exp sig_inc key_tag name
base64(sig)
*.thesprawl.org=A  5  3  86400  20030322173103  20030220173103  2642  thesprawl.org.
oJB1W6WNGv+ldvQ3WDG0MQkg5IEhjRip8WTrPYGv07h108dUKGMeDPKijVCHX3DDKdfb+v6oB9wfuh3DTJXUAf
I/M0zmO/zz8bW0Rznl8O3tGNazPwQKkRN20XPXV6nwwfoXmJQbsLNrLfkGJ5D6fwFm8nN+6pBzeDQfsS3Ap3o=
```

## EK 4. Sahte DHCP için ettercap konfigürasyon dosyası (dnschef.ini) kaynak kodları

```
#############################################################################
#                                                                           #
#  ettercap -- etter.conf -- configuration file                             #
#                                                                           #
#  Copyright (C) ALoR & NaGA                                                #
#                                                                           #
#  This program is free software; you can redistribute it and/or modify     #
#  it under the terms of the GNU General Public License as published by     #
#  the Free Software Foundation; either version 2 of the License, or        #
#  (at your option) any later version.                                      #
#                                                                           #
#                                                                           #
#############################################################################
```

```
[privs]
ec_uid = 65534                # nobody is the default
ec_gid = 65534                # nobody is the default
[mitm]
arp_storm_delay = 10          # seconds
arp_poison_warm_up = 1        # seconds
arp_poison_delay = 10         # seconds
arp_poison_icmp = 1           # boolean
arp_poison_reply = 1          # boolean
arp_poison_request = 0        # boolean
arp_poison_equal_mac = 1      # boolean
dhcp_lease_time = 7           # seconds
port_steal_delay = 10         # seconds
port_steal_send_delay = 2000  # microseconds
[connections]
connection_timeout = 300      # seconds
connection_idle = 5           # seconds
connection_buffer = 10000     # bytes
connect_timeout = 5           # seconds
[stats]
sampling_rate = 50            # number of packets
[misc]
close_on_eof = 1              # boolean value
store_profiles = 1            # 0 = disabled; 1 = all; 2 = local; 3 = remote
aggressive_dissectors = 1     # boolean value
skip_forwarded_pcks = 1       # boolean value
checksum_check = 0            # boolean value
submit_fingerprint = 0       # boolean valid (set if you want ettercap to submit unknown
finger prints)
checksum_warning = 0          # boolean value (valid only if checksum_check is 1)
sniffing_at_startup = 1       # boolean value

############################################################################
#
# You can specify what DISSECTORS are to be enabled or not...
#
# e.g.:     ftp = 21            enabled on port 21 (tcp is implicit)
#           ftp = 2345          enabled on non standard port
#           ftp = 21,453        enabled on port 21 and 453
#           ftp = 0             disabled
#
#  NOTE: some dissectors have multiple default ports, if you specify a new
#        one, all the default ports will be overwritten
#
#
```

```
#dissector                default port

[dissectors]
ftp = 21                  # tcp    21
ssh = 22                  # tcp    22
telnet = 23               # tcp    23
smtp = 25                 # tcp    25
dns = 53                  # udp    53
dhcp = 67                 # udp    68
http = 80                 # tcp    80
ospf = 89                 # ip     89  (IPPROTO 0x59)
pop3 = 110                # tcp    110
#portmap = 111             # tcp / udp
vrrp = 112                # ip     112 (IPPROTO 0x70)
nntp = 119                # tcp    119
smb = 139,445             # tcp    139 445
imap = 143,220            # tcp    143 220
snmp = 161                # udp    161
bgp = 179                 # tcp    179
ldap = 389                # tcp    389
https = 443               # tcp    443
ssmtp = 465               # tcp    465
rlogin = 512,513          # tcp    512 513
rip = 520                 # udp    520
nntps = 563               # tcp    563
ldaps = 636               # tcp    636
telnets = 992             # tcp    992
imaps = 993               # tcp    993
ircs = 994                # tcp    993
pop3s = 995               # tcp    995
socks = 1080              # tcp    1080
radius = 1645,1646        # udp    1645 1646
msn = 1863                # tcp    1863
cvs = 2401                # tcp    2401
mysql = 3306              # tcp    3306
icq = 5190                # tcp    5190
ymsg = 5050               # tcp    5050
mdns = 5353               # udp    5353
vnc = 5900,5901,5902,5903  # tcp    5900 5901 5902 5903
x11 = 6000,6001,6002,6003  # tcp    6000 6001 6002 6003
irc = 6666,6667,6668,6669  # tcp    6666 6667 6668 6669
gg = 8074               # tcp    8074
proxy = 8080              # tcp    8080
rcon = 27015,27960        # udp    27015 27960
ppp = 34827               # special case ;) this is the Net Layer code
TN3270 = 23,992           # tcp    23 992
```

```
# you can change the colors of the curses GUI.
# here is a list of values:
#  0 Black     4 Blue
#  1 Red       5 Magenta
#  2 Green     6 Cyan
#  3 Yellow    7 White
[curses]
color_bg = 0
color_fg = 7
color_join1 = 2
color_join2 = 4
color_border = 7
color_title = 3
color_focus = 6
color_menu_bg = 4
color_menu_fg = 6
color_window_bg = 4
color_window_fg = 7
color_selection_bg = 6
color_selection_fg = 6
color_error_bg = 1
color_error_fg = 3
color_error_border = 3

# This section includes all the configurations that needs a string as a
# parmeter such as the redirect command for SSL mitm attack.
[strings]
# the default encoding to be used for the UTF-8 visualization
utf8_encoding = "ISO-8859-1"
# the command used by the remote_browser plugin
remote_browser = "xdg-open http://%host%url"

####################################
#       redir_command_on/off
####################################
# you must provide a valid script for your operating system in order to have
# the SSL dissection available
# note that the cleanup script is executed without enough privileges (because
# they are dropped on startup). so you have to either: provide a setuid program
# or set the ec_uid to 0, in order to be sure the cleanup script will be
# executed properly
# NOTE: this script is executed with an execve(), so you can't use pipes or
# output redirection as if you were in a shell. We suggest you to make a script if
# you need those commands.
```

```
#---------------
#     Linux
#---------------
# if you use ipchains:
   #redir_command_on = "ipchains -A input -i %iface -p tcp -s 0/0 -d 0/0 %port -j
REDIRECT %rport"
   #redir_command_off = "ipchains -D input -i %iface -p tcp -s 0/0 -d 0/0 %port -j
REDIRECT %rport"

# if you use iptables:
   #redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport %port -
j REDIRECT --to-port %rport"
   #redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport %port -
j REDIRECT --to-port %rport"

#---------------
#    Mac Os X
#---------------
# quick and dirty way:
   #redir_command_on = "ipfw add set %set fwd 127.0.0.1,%rport tcp from any to any %port
in via %iface"
   #redir_command_off = "ipfw -q delete set %set"

# a better solution is to use a script that keeps track of the rules interted
# and then deletes them on exit:

 # redir_command_on:
 # ----- cut here -------
#   #!/bin/sh
#   if [ -a "/tmp/osx_ipfw_rules" ]; then
#      ipfw -q add `head -n 1 osx_ipfw_rules` fwd 127.0.0.1,$1 tcp from any to any $2
in via $3
#   else
#      ipfw add fwd 127.0.0.1,$1 tcp from any to any $2 in via $3 | cut -d " " -f 1 >>
/tmp/osx_ipfw_rules
#   fi
# ----- cut here -------

# redir_command_off:
# ----- cut here -------
#   #!/bin/sh
#   if [ -a "/tmp/osx_ipfw_rules" ]; then
#      ipfw -q delete `head -n 1 /tmp/osx_ipfw_rules`
#      rm -f /tmp/osx_ipfw_rules
#   fi
# ----- cut here -------
```

```
#---------------
#   Open BSD
#---------------

# unfortunately the pfctl command does not accepts direct rules adding
# you have to use a script which executed the following command:

# ----- cut here -------
#   #!/bin/sh
#   rdr pass on $1 inet proto tcp from any to any port $2 -> localhost port $3 | pfctl
-a sslsniff -f -
# ----- cut here -------

# it's important to remember that you need "rdr-anchor sslsniff" in your
# pf.conf in the TRANSLATION section.

   #redir_command_on = "the_script_described_above %iface %port %rport"
   #redir_command_off = "pfctl -a sslsniff -Fn"

# also, if you create a group called "pfusers" and have EC_GID be that group,
# you can do something like:
#     chgrp pfusers /dev/pf
#     chmod g+rw /dev/pf
# such that all users in "pfusers" can run pfctl commands; thus allowing non-root
# execution of redir commands.

##########
#  EOF   #
##########
```

# ÖZGEÇMİŞ

1984 yılında Silvan'da doğdum. İlköğretimi Muş'un merkeze bağlı Bağlar Mahallesinde, ortaokulu Alparslan İlköğretim Okulunda ve liseyi Muş Lisesinde tamamladım. 2005 yılında kazandığım Uluslararası Kıbrıs Üniversitesi Mühendis Fakültesi Bilişim Sistemleri Mühendisliği bölümünden 2010 yılında mezun oldum. 2017 yılında Bitlis Eren Üniversitesi Fen Bilimleri Enstitüsü Elektrik Elektronik Mühendisliği Anabilim Dalı'nda yüksek lisansa başladım. Yabancı dilim İngilizcedir.

Cengiz ALMAZ