Batuhan AVLAYAN – batuhanavlayan@etu.edu.tr

Github Repo Link: https://github.com/bavlayan/Usom-IP-Blocker-using-OpenFlow/

Ödev 1 – OpenFlow protokolü kullanılarak SDN anahtarlarında IP adresleri engelleme kurallarının tanımlanması

Projenin amacı, resimdeki topolojide bulunan herhangi bir hosttan usom tarafından zararlı olduğu belirlenmiş yaklaşık 120000 web sitesine giden tüm paketleri python programala dilini ve openflow protokülü kullanarak engelleyen yazılım geliştirmek.

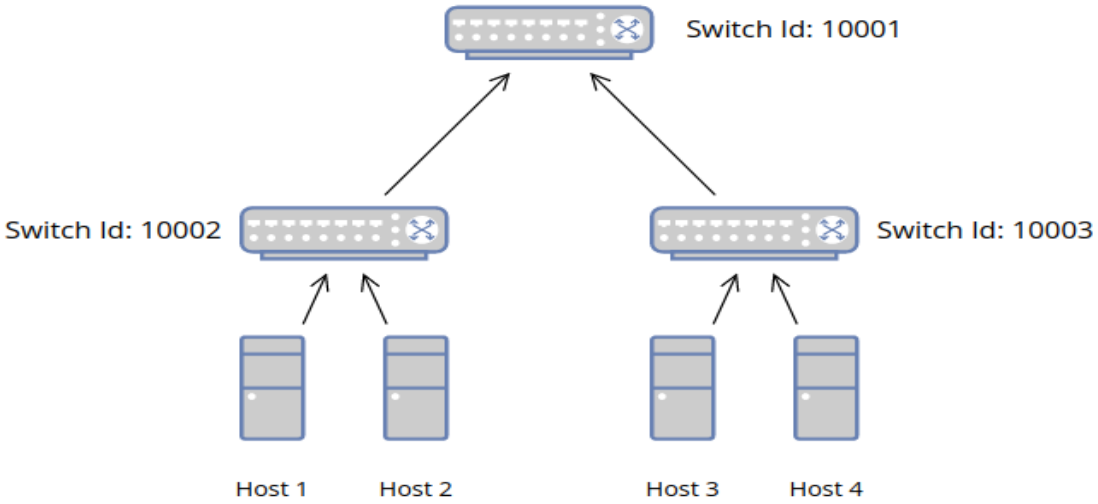


Figure 1: Topoloji

Topolojide toplam 3 Switch ve 4 host bulunmaktadır. Zararlı url'lere giden paketleri engelleme işlemi **Switch ID: 10001** olan switch üzerinde gerçekleşmektedir. Switch üzerinde sadece *ipv4* paketleri engellemesi yapılmaktadır.

- Proje aşağıdaki dosyalardan oluşmaktadır.
  1 - blocked_url.json
  2 - BlockedUrl.py
  3 - Constants.py
  4 - MyTopology.py
  5 - RunTopo.py
  6 - SDNSwitch.py
  7 - UsomUrlHelper.py

1. blocked_url.json

Yasaklı ***aktif*** url bilgilerinin json formatında tutulduğu dosya. Program çalışırken yasaklı url bilgilerini buradan alarak kullanmaktadır. Json formatı aşağıdaki gibidir. Json objesinde url ismi, url'in ip adresi ve url'in aktif olup olmadığını tutan nitelikler bulunmaktadır.

{ "url_name" : "parcakontorbayilik.com", "ip": "154.95.233.67", "is_active":"true" }

2. BlockedUrl.py

BlockedUrl sınıfı yasaklanmış url bilgilerinin tutulduğu özellikleri içermektedir.

```
class BlockedUrl():
    def __init__(self, url_name, ip, is_active):
        self.url_name = url_name
        self.ip = ip
        self.is_active = is_active
```

Figure 2: BlockedUrl Class

3. Constants.py

Tüm program içinden kullanılan sabit değerlerin tutulduğu python dosyası.

4. MyTopology.py

Yukarıda *Figure-1'de* çizilen topolojinin oluşturulması için kullanılan fonksiyon kodlarının bulunduğu dosya. MyTopology.py sınıfı içerisinde *create_host* fonksiyonu ile hostların oluşturulması, *create_switch* fonksiyonu ile Swtichlerin oluşturulaması ve *create_link* fonksiyonu oluşturulan bu switchlerin ve hostların birbirine bağlanması işlemi yapılmaktadır.

5. RunTopo.py

RunTopo.py içerisinde bulunan *run_topo* fonksiyonu ile Mininet kütüphanesi kullanılarak *MyTopology* sınıfı ile oluşturulan topolojinin mininet ortamında simule edilme işlemi yapılmaktadır.

6. SDNSwitch.py

SDNSwitch.py içerisinde Ryu tarafından oluşturacak olan switch ayarlarının yapıldığı ve bloklanacak url'lerin *blocked_url.json* doyasından okunması ve kurallarının eklenmesi işlemi yapılmaktadır.

load_json (figure 3) fonksiyonu ile *blocked_url.json* dosyasının okunup *BlockedUrl* objesinden oluşan *blocked_url_array* dizisine atılması işlemi yapılmaktadır.

```
def __load_json(self):
    f = open(Constants.BLOCKED_URL_JSON_FILE_NAME)
    data = json.load(f)

    for json_data in data:
        blocked_url = BlockedUrl(
            json_data[Constants.URL_NAME],
            json_data[Constants.IP],
            bool(json_data[Constants.IS_ACTIVE])
        )
        self.logger.info("Readed url: %s --- %s", blocked_url.url_name, blocked_url.ip)
        self.blocked_url_array.append(blocked_url)
        self.blocked_url_array = list(filter(lambda x: x.is_active == True, self.blocked_url_array))
    self.logger.info("Blocked url uploaded len: %s", str(len(self.blocked_url_array)))
```

Figure 3: load_json function

packet_in_handler (figure 4) fonksiyonunda bulunan aşağıdaki kod bloğu ile Id'si (dpid) 10001 switch'e engellenecek olan ip adreslerinin kural eklenmesi işlemi yapılmaktadır. Eklenen kural sadece ipv4 paketleri üstünde geçerlidir.

```
if dpid == Constants.FIREWALL_SWITCH_ID:
    for index, blocked_url in enumerate(self.blocked_url_array):
        self.logger.info("added droped packed: %s --- %s", blocked_url.url_name, blocked_url.ip)
        if pkt_ipv4 and (pkt_ipv4.src == blocked_url.ip or pkt_ipv4.dst == blocked_url.ip):
            default_match_1 = parser.OFPMatch(
                eth_type=ether_types.ETH_TYPE_IP,
                ipv4_src=pkt_ipv4.src,
                ipv4_dst=pkt_ipv4.dst
            )

            default_match_2 = parser.OFPMatch(
                eth_type=ether_types.ETH_TYPE_IP,
                ipv4_src=pkt_ipv4.dst,
                ipv4_dst=pkt_ipv4.src
            )

            self.add_flow(datapath, index + 3, default_match_1, [])
            self.add_flow(datapath, index + 2, default_match_2, [])
            self.logger.info("packet drop: FIREWALL_ID: %s - SRC: %s - DST: %s - URL: %s", dpid, pkt_ipv4.src, pkt_ipv4.dst, blocked_url.url_name)
elif out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
    self.add_flow(datapath, 1, match, actions)
```

Figure 4: packet_in_handler code snippet

7. UsomUrlHelper.py

https://www.usom.gov.tr/url-list.txt adresinden çekilen adreslerin ip'lerinin belirlenmesi işlemlerini yapılmaktadır. *get_blocked_urls_from_usom* fonksiyonu usom adresinden alınan zaralı bağlantı adreslerinin parse işlemleri yapılmaktadır.

```
def __get_blocked_urls_from_usom(self):
    try:
        response = urllib.request.urlopen(self.usom_url, timeout=3)
        for url in response:
            url_name = url.decode("utf-8").rstrip("\n")
            is_ip = self.__check_ip(url_name)
            if is_ip is True:
                continue
            blocked_url = BlockedUrl(url_name, '', False)
            self.blocked_url_list.append(blocked_url)
    except:
        print("Url not open")
        sys.exit()
```

Figure 5: get_blocked_urls_from_usom function

get_ip_from_url fonksiyonu ile url adreslerinin ip adresi tespit edilmektedir.

```
def __get_ip_from_url(self, blocked_url_list):
    for blocked_url in blocked_url_list:
        try:
            ip_address = socket.gethostbyname(blocked_url.url_name)
            blocked_url.is_active = True

            if not ip_address:
                print(Fore.YELLOW + blocked_url.url_name)
                blocked_url.is_active = False
                self.blocked_url_list.remove(blocked_url)

            if self.__is_ip_private(ip_address):
                blocked_url.is_active = False
                self.blocked_url_list.remove(blocked_url)

            blocked_url.ip = ip_address
            print(Fore.GREEN + "Url:" + blocked_url.url_name + " Ip:" + blocked_url.ip)
        except:
            blocked_url.is_active = False
            print(Fore.RED + "Url:" +blocked_url.url_name + " IP Address not found")
            self.blocked_url_list.remove(blocked_url)
            continue
```

Figure 6: get_ip_from_url function

set_ip fonksiyonu ile oluşturulmuş olan oluşturalan *blocked_url_list* array'inde bulunan url'lerin ip set edilme işlemi yapılmaktadır. *blocked_url_list array* length'i çok büyük olduğundan burada ip et edilme işlemi thread ile yapılmakatadır.

```
def __set_ip(self):
    try:
        thread_arr = []
        loop_last_index = int(len(self.blocked_url_list) / self.thread_count)
        first_index = 0
        for i in range(self.thread_count):
            partial_blocked_list = self.blocked_url_list[first_index:loop_last_index]
            first_index = loop_last_index
            loop_last_index = loop_last_index * 2
            t = threading.Thread(target=self.__get_ip_from_url, args=(partial_blocked_list,))
            thread_arr.append(t)

        for t in thread_arr:
            t.start()

        for t in thread_arr:
            t.join()
    except:
        print("Getting error in set ip function")
```

Figure 7: set_ip function

check_ip fonksiyonu ile Usom'dan alınan url listesinde bulunan ip adreslerinin tespit edilmesi işlemi yapılmakatadır.

```
def __check_ip(self, url):
    ip_regex = "^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$"
    if(re.search(ip_regex, url)):
        return True
    return False
```

Figure 8: chek_ip function

is_ip_private fonksiyonu bir ip adresinin özel ip blokları arasında olup olmadığını tespit etmektedir.

```
def __is_ip_private(self, ip):
    priv_lo = re.compile("^127\.\d{1,3}\.\d{1,3}\.\d{1,3}$")
    priv_24 = re.compile("^10\.\d{1,3}\.\d{1,3}\.\d{1,3}$")
    priv_20 = re.compile("^192\.168\.\d{1,3}.\d{1,3}$")
    priv_16 = re.compile("^172.(1[6-9]|2[0-9]|3[0-1]).[0-9]{1,3}.[0-9]{1,3}$")
    res = priv_lo.match(ip) or priv_24.match(ip) or priv_20.match(ip) or priv_16.match(ip)
    return res is not None
```

Figure 9: is_ip_private function

create_json_file fonksiyonu *blocked_url_list* array'in json dump edilerek *blocked_url.json* dosyasını oluşturmaktadır.

```
def create_json_file(self):
    self.__get_blocked_urls_from_usom()
    self.__set_ip()
    try:
        with open(Constants.BLOCKED_URL_JSON_FILE_NAME, 'w') as json_file:
            json.dump(self.blocked_url_list, json_file, default=vars)
            print(Fore.GREEN + "blocked_url.json file was created...")
    except:
        print(Fore.RED + "An error occured when creating " + Constants.BLOCKED_URL_JSON_FILE_NAME + " json file")
```

Figure 10: create_json_file

PROGRAMIN ÇALIŞTIRILMASI VE TEST SONUÇLARI

Programın çalıştırılması

Programın çalıştırılması için aşağıdaki komutlar sırası ile çalıştırılmalıdır.

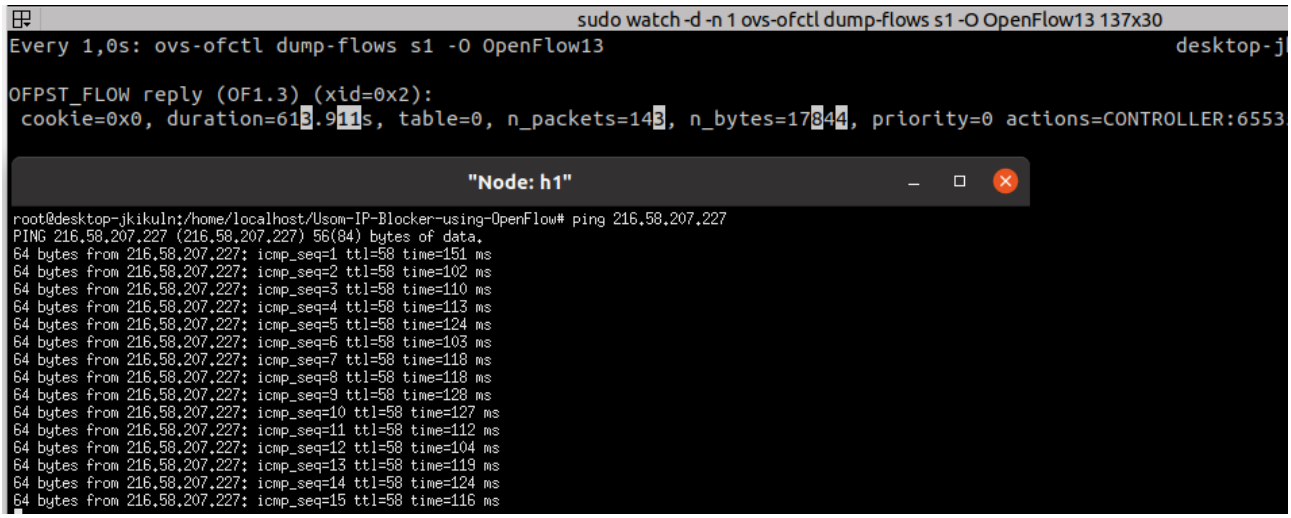
1-python3 UsomUrlHelper.py – komutu ile blocked_url.json dosyasını oluşturuyoruz

2-*sudo ryu-manager SDNSwitch.py* – komutu ile switch'leri oluşturuyoruz.

3-*sudo python3 RunTopo.py* – komutu ile Mininet üzerinde topojimizi oluşturuyoruz.

Program test sonuçları

1. H1 host üstünden üzerinden (216.58.207.227) google.com'a ping attığımızda, işlemin başarılı bir şekilde gerçekleştiğini görebiliriz. Aynı zamanda s1 switch üstünde akış loglarına baktığımızda priority 0 olan flow'un paket sayısını arttığını görebiliriz.



2. H2 host üstünden yasaklı bir url'e gohtci.com (64.111.117.54) ping attığımızda, işlemin gerçekleşmediğini görebiliriz. Aynı zamanda s1 switch üstünde akış loglarına baktığımızda priortiy 23012 olan flow paket sayısının arttığını ve paketin drop edildiğini görebiliriz.