به نام او


# گزارش پروژه درس شبکه های کامپیوتری

استاد میزانیان

حسین گلی

۹۹۱۰۲۱۲۳

تابستان ۱۴۰۳

## محتواها

نحوه کارکرد TUN/TAP ..... ۳

نحوه HandShake بین کلاینت و سرور ..... ۴

بر سر پکت دقیقا چه می آید؟ ..... ۵

EDNS ..... ۶

نتایج ..... ۷

## نحوه کارکرد TUN/TAP

در ابتدا به نحوه کارکرد TUN/TAP Device ها می پردازیم. در واقع TUN به ما امکان ساختن یک interface را می دهد. Interface را می توان مشابه یک لایه بین اینترنت و کامپیوتر ما دید. کارت شبکه ما عملا یک interface می باشد.

در این پروژه از آنجایی که ما میخواهیم پکت های TCP که به سمت اینترنت ارسال می شود را دستکاری و درون یک پکت DNS بنویسیم و میخوایم این کار را با تمامی پکت ها بکنیم بهترین کار ساختن یک interface مجازی با استفاده از امکاناتی که Linux به ما میدهد می باشد. با ساختن یک interface ما تمامی پکت های روی یک سابنت را می توانیم دریافت و هر تغییری که لازم داریم بر روی آن ها اعمال کنیم.

```
    def create_tun_interface(self):
        try:
            self.tun = os.open('/dev/net/tun', os.O_RDWR)
            ifr = struct.pack('16sH', self.tun_name.encode('utf-8'), IFF_TUN | IFF_NO_PI)
            fcntl.ioctl(self.tun, TUNSETIFF, ifr)
            print(f"TUN interface {self.tun_name} created")
            subprocess.run(['sudo', 'ip', 'addr', 'add', self.subnet, 'dev', self.tun_name])
            subprocess.run(['sudo', 'ip', 'link', 'set', 'up', 'dev', self.tun_name])
        except Exception as e:
            print(f"Error creating TUN interface: {e}")
            exit(1)
```

قطعه کد بالا نشان دهنده نحوه ساخت یک interface tun می باشد.

Interface tun که ساخته شد به ما قابلیت خواندن و نوشتن می دهد. تمامی پکت هایی که بر روی سابنتی که ست میکنیم (که در این پروژه از 172.16.0.2/24) استفاده شده است، با استفاده از قطعه کد های زیر قابل خواندن و نوشتن هستند.

(خواندن از اینترفیس TUN)

```
        while True:
            packet = os.read(self.tun, 1500)
            \\do something
```

(نوشتن در اینترفیس TUN)

```
        while True:
            os.write(self.tun, ip_packet)
            \\do something
```

در واقع اینترفیس TUN برای ما مانند یک واسط بین کارت شبکه و پکت هایی که خوانده ایم/ یا بر روی این اینترفیس نوشته ایم عمل خواهد کرد.

## نحوه HandShake بین کلاینت و سرور

در این پروژه برای شروع ارتباط بین کلاینت و سرور نیاز به یک handshake داریم. مکانیزم handshake ما به این صورت می باشد که یک Key سمت سرور ست شده است (هنگامی که آن را ران میکنیم) و کلاینت هنگام شروع باید به سرور آن Key را ارسال کند و اگر درست باشد یک OK دریافت میکند و سرور و کلاینت هردو اینترفیس های TUN را ساخته و بر روی حالت آماده باش برای پراکسی کردن درخواست ها قرار خواهند گرفت.

```
    def start_client(self):
        if self.server is None:
            print('Server IP is required in client mode')
            return
        self.server_host, self.server_port = self.server.split(':')
        self.server_port = int(self.server_port)
        print(f'Sending to {self.server_host}:{self.server_port}')
        self.sock.sendto(self.key.encode(), (self.server_host, self.server_port))
        print('Performing Handshake')
        data, addr = self.sock.recvfrom(1024)
        print(f"Received {data.decode()} from {addr}")
        if data.decode() != 'OK':
            print('Invalid Key')
            return
        print('Connected to server')
    def start_server(self):
        self.sock.bind(('0.0.0.0', self.port))
        ip = socket.gethostbyname(socket.gethostname())
        print(f'Listening on {ip}:{self.port}')
        while True:
            data, addr = self.sock.recvfrom(1024)
            print(f"Received {data.decode()} from {addr}")
            if data.decode() == self.key:
                self.server_host, self.server_port = addr
                self.server_port = int(self.server_port)
                self.sock.sendto('OK'.encode(), addr)
                print('Client handshake success')
                break
            else:
                print('Invalid Key')
```

پس از آنکه هر دو بر روی حالت آماده باش قرار گرفتند 2 ترد برای خواندن از اینترفیس تان و نوشتن بر روی اینترفیس تان شروع می شود. همچنین قابل ذکر است که سرور و کلاینت از طریق یک پورت UDP و UDP connection باهم در ارتباط هستند.

## بر سر پکت دقیقا چه می آید؟

فرض کنید از سمت کلاینت میخواهید یک پکت به سرور بفرستید. گام های آن را به صورت کامل بررسی می کنیم. از سمت کلاینت میخواهیم یک پکت TCP را به دنیا آزاد بفرستیم. ابتدا آن را باید درون یک پکت DNS انکپسولیت کنیم و سپس زیرا آن را به سمت سرور (خارج از کامپیوتر) میخواهیم بفرستیم و کل سیستم را به روی اینترفیس TUN امان هدایت کرده ایم پکت در تابع read data from tun دریافت می شود و پس از انجام پراسسینگ های لازم (که در ادامه آن ها را بررسی میکنیم) برخی فیلد های آن مانند MSS (اگر بسته SYN ) باشد تغییر و درون یک پکت EDNS قرار میگیرد و بر روی سوکت UDP نوشته می شود.

سپس سمت سرور که یک پکت بر روی پورت UDP دریافت کرده است تابع read data from socket یک پکت را دریافت کرده و سپس بسته TCP درون آن را بازیابی (از پکت EDNS) و بر روی interface TUN می نویسد. از آنجا که سمت سرور NAT اینگ را طوری انجام داده ایم که بسته هایی که آدرسشان با interface TUN باشد این بسته به سمت کارت شبکه هدایت می شود و سپس بر روی کارت شبکه دریافت و به read from tun می رسد . سپس مشابه کلاینت این بسته درون پکت EDNS قرار میگیرد و به سوکت UDP ارسال می شود. (اگر دقت کنید میبینید که این فرایند دقیقا متقارن است :) )

در ادامه تغییراتی که بر روی پکت قرار میگیرد را مشاده میکنیم. ( MSSو مصاحبه دوباره چکسام)

```
def process_packet(self, packet):
        ip = IP(packet)
        if ip.proto == 6:  # TCP
            packet = self.modify_tcp_packet(ip)
            edns_packet = self.to_edns(packet)
            self.sock.sendto(edns_packet, (self.server_host,
int(self.server_port)))
            print(f'Sent EDNS packet')
        else:
            print(f'Ignoring packet, protocol is {ip.proto}')

    def modify_tcp_packet(self, ip):
        tcp = ip[TCP]
        if 'S' in tcp.flags:
            tcp.options = self.modify_mss_option(tcp.options)
            self.recompute_checksums(ip, tcp)
        return raw(ip)
```

```
    def modify_mss_option(self, options):
        new_options = []
        for opt in options:
            if opt[0] == 'MSS':
                new_mss = min(MTU, opt[1])
                new_options.append(('MSS', new_mss))  # Set MSS to 1300
            else:
                new_options.append(opt)
        return new_options

    def recompute_checksums(self, ip, tcp):
        del ip.chksum
        del tcp.chksum
        ip.chksum
        tcp.chksum
```

## EDNS

حال به صورت مختصر نحوه کارکرد EDNS را توضیح می دهیم.

پکت ها درون کتابخانه scapy به صورت درختی عمل میکنند. (لایه لایه قرار میگیرند.) پکت هایی که درون EDNS مینویسیم از چندین لایه تشکل شده است. برای دسترسی به ar درون آن لازم است ابتدا به DNSROPT، سپس به rdata آن دسترسی پیدا کنیم. همچنین برای ساختن این نوع پکت ها صرفا لازم است یک پکت DNS به علاوه فیلد های اپشنال DNSROPT را قرار دهیم.

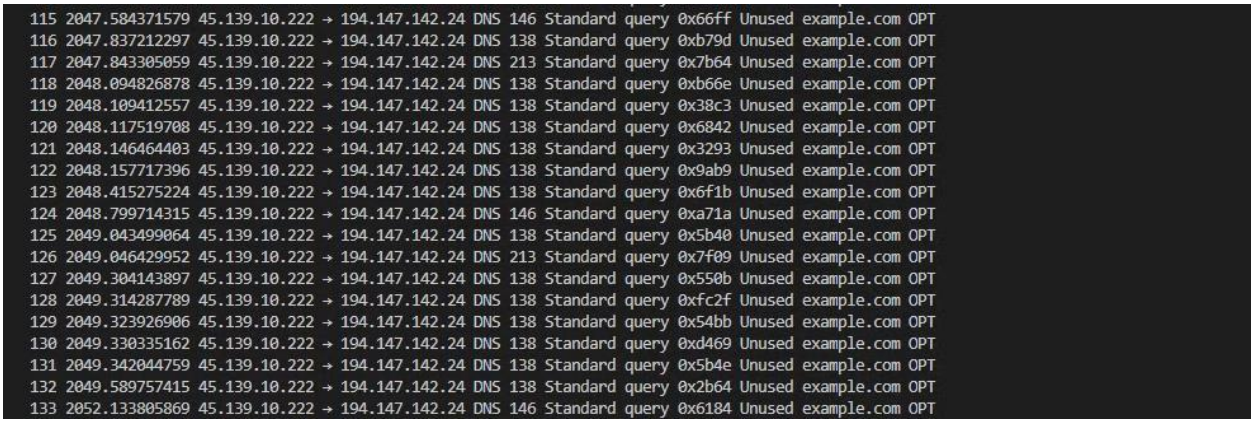
```
    def to_edns(self, payload):
        edns_opt = DNSRROPT(
            rclass=4096,  # UDP payload size
            rdlen=len(payload) + 4,  # Length of the option data plus option
header
            rdata=[EDNS0TLV(optcode=65001, optlen=len(payload), optdata=payload)]
        )
        dns_packet = DNS(
            id=random.getrandbits(16),
            rd=1,
            qd=DNSQR(qname="example.com", qtype="ANY", qclass="IN"),
            ar=edns_opt
        )
        return bytes(dns_packet)

    def from_edns(self, edns_packet):
```

```
        dns_packet = DNS(edns_packet)
        payload = b""
        for additional in dns_packet.ar:
            if isinstance(additional, DNSRROPT):
                for opt in additional.rdata:
                    if isinstance(opt, EDNS0TLV) and opt.optcode == 65001:
                        payload = opt.optdata
        return payload
```

## نتایج

در زیر برخی پکت های EDNS هنگامی که VPN را بر روی سرور و کلاینت ران کرده ایم مشاهده میکند.




همچنین در تصویر بعدی سناریو دانلود یک عکس از سایت ویکی پدیا با استفاده از VPN امان و چک کردن چک سام آن را مشاهده میکنید.

```
--2024-08-07 22:36:16--  https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-logo-v2.png
Resolving www.wikipedia.org (www.wikipedia.org)... 185.15.59.224, 2a02:ec80:300:ed1a::1
Connecting to www.wikipedia.org (www.wikipedia.org)|185.15.59.224|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15829 (15K) [image/png]
Saving to: 'Wikipedia-logo-v2.png'

Wikipedia-logo-v2.png               100%[===================================================================>]  15.46K  --.-KB/s    in 0s

2024-08-07 22:36:16 (51.5 MB/s) - 'Wikipedia-logo-v2.png' saved [15829/15829]

root@srv4684907080:~# md5sum Wikipedia-logo-v2.png
ec8a2441be89c67acacad5bd77ab6bfa  Wikipedia-logo-v2.png
root@srv4684907080:~# nslookup wikipedia.org
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   wikipedia.org
Address: 185.15.59.224
Name:   wikipedia.org
Address: 2a02:ec80:300:ed1a::1

root@srv4684907080:~# sudo ip route add 185.15.59.224 dev tun0
root@srv4684907080:~# wget https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-logo-v2.png
--2024-08-07 22:37:45--  https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-logo-v2.png
Resolving www.wikipedia.org (www.wikipedia.org)... 185.15.59.224, 2a02:ec80:300:ed1a::1
Connecting to www.wikipedia.org (www.wikipedia.org)|185.15.59.224|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15829 (15K) [image/png]
Saving to: 'Wikipedia-logo-v2.png.1'

Wikipedia-logo-v2.png.1             100%[===================================================================>]  15.46K  --.-KB/s    in 0.007s

2024-08-07 22:37:46 (2.07 MB/s) - 'Wikipedia-logo-v2.png.1' saved [15829/15829]

root@srv4684907080:~# md5sum Wikipedia-logo-v2.png.1
ec8a2441be89c67acacad5bd77ab6bfa  Wikipedia-logo-v2.png.1
```