VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

PDS - DHCP ÚTOKY

AUTOR PRÁCE **JAN PAWLUS**

BRNO 2018

Obsah

1 DHCP útoky **2**
1.1 DHCP starvation 2
1.1.1 Discover flood 2
1.1.2 DORA cyklus 3
1.2 DHCP rogue server 4

2 Implementace **5**
2.1 Práce s vlákny 5
2.2 DHCP starvation 5
2.3 DHCP rogue server 6

3 Demonstrace **7**
3.1 Testovací prostředí 7

Literatura **10**

Kapitola 1

DHCP útoky

Před samotným popisem DHCP útoků je třeba pochopit princip samotného DHCP (celá dokumentace a projekt se zabývá pouze *IPv4* implementací). Vzhledem k rozsahu projektu se budu zabývat pouze základními kameny tohoto protokolu a zprávy, které není nutné využívat k úspěšnému útoku, vynechám. Těmito základními kameny jsou: [3]

- *DHCP discover* je první zpráva, vysílaná jako broadcast, kterou klient zasílá, aby získal IP adresu, popř. další informace jako adresa DNS serveru, výchozí brána a jiné. Klient specifikuje další požadované informace dynamickým polem *DHCP options*.
- *DHCP offer* je odpovědí DHCP serveru na *DHCP discover* obsahující navrženou IP adresu a případně další informace, které byly specifikovány klientem. Může být zaslána buď jako broadcast, nebo unicast (specifikuje klient v *DHCP discover* pomocí prvního bitu atributu *flags*).
- *DHCP request* zasílá klient buď jako potvrzení nabídky od serveru, nebo za účelem prodloužení *lease time*. Jelikož klientu mohou přijít zprávy od více DHCP serverů, je nutné specifikovat, kterému serveru je zpráva určena.
- *DHCP ack* stvrzuje předešlou komunikaci. Server danou IP adresu dále nenabízí minimálně do vypršení *lease time*.

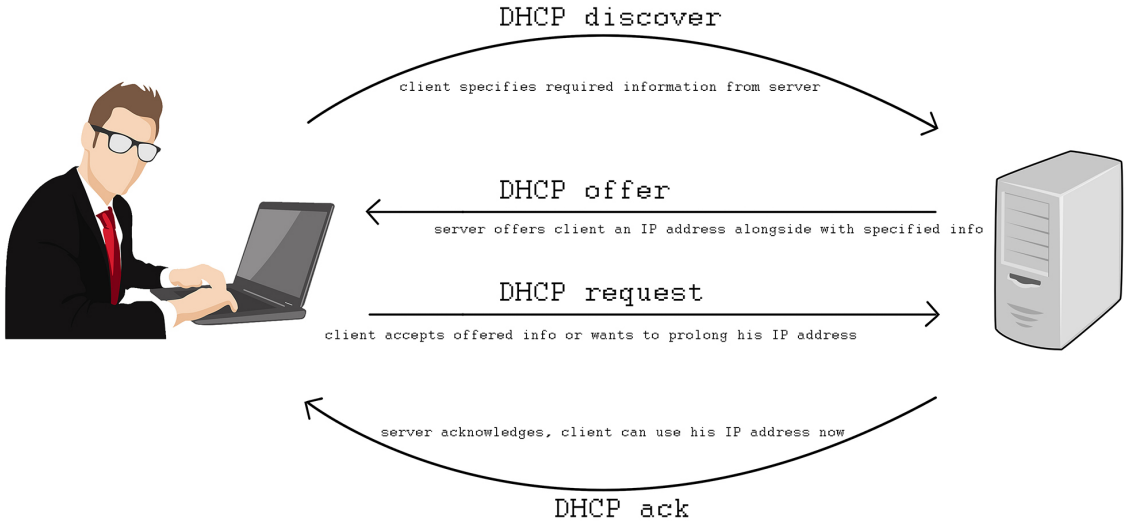
Schéma komunikace je shrnuto na obrázku 1.1

1.1 DHCP starvation

DHCP starvation je útok založený na principu *denial of service*. Útočník zahlcuje legitimní DHCP server falešnými zprávami *DHCP discover* s podvrženými MAC adresami tak, aby rychle vyčerpal *ip address pool* DHCP serveru, který tak nemůže nabízet legitimním klientům nové IP adresy. Této situace může klient docílit více způsoby. [2].

1.1.1 Discover flood

Jednodušší přístup by útočník volil pouze zahlcením zprávami typu *DHCP discover*, na něž server odpovídá zprávami typu *DHCP offer*, kdy útočníku nabízí volnou IP adresu. Jakmile totiž server odešle tuto zprávu, na určitou dobu nenabízí tuto již nabídnutou IP adresu dalším klientům. Tím pádem lze na určitou dobu vyřadit server z provozu.

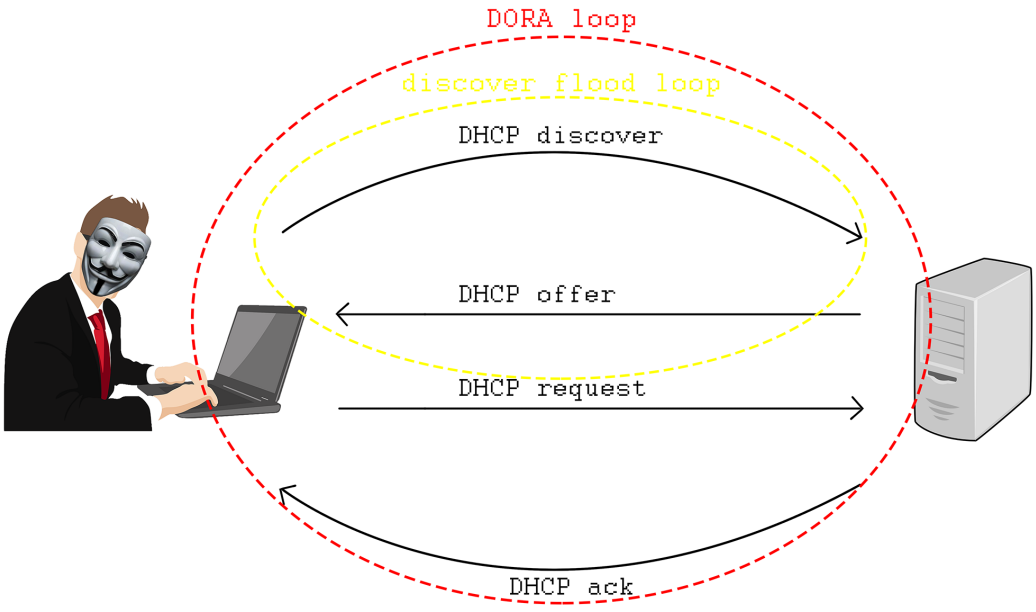


Obrázek 1.1: Schéma komunikace protokolu DHCP.

Problém této varianty je to, že doba, po kterou server nenabízí IP adresu navrženou v *DHCP offer*, je implementačně závislá [3]. Nelze tedy s jistotou říct, na jak dlouho a jestli vůbec takto dokážeme server vyřadit z provozu.

1.1.2 DORA cyklus

Robustnější a lepší variantou útoku je provést celý *DORA* cyklus. Tedy nejen, že útočník zasílá zprávy typu *DHCP discover*, ale také po obdržení *DHCP offer* zareaguje zasláním zprávy *DHCP request* vytvořené na základě přijaté nabídky. V takovém případě má po přijetí *DHCP ack* útočník jistotu, že tuto IP adresu server nenabídne po dobu danou *lease time*. Oba typy útoků lze vidět na obrázku 1.2.



Obrázek 1.2: Schéma DHCP útoků

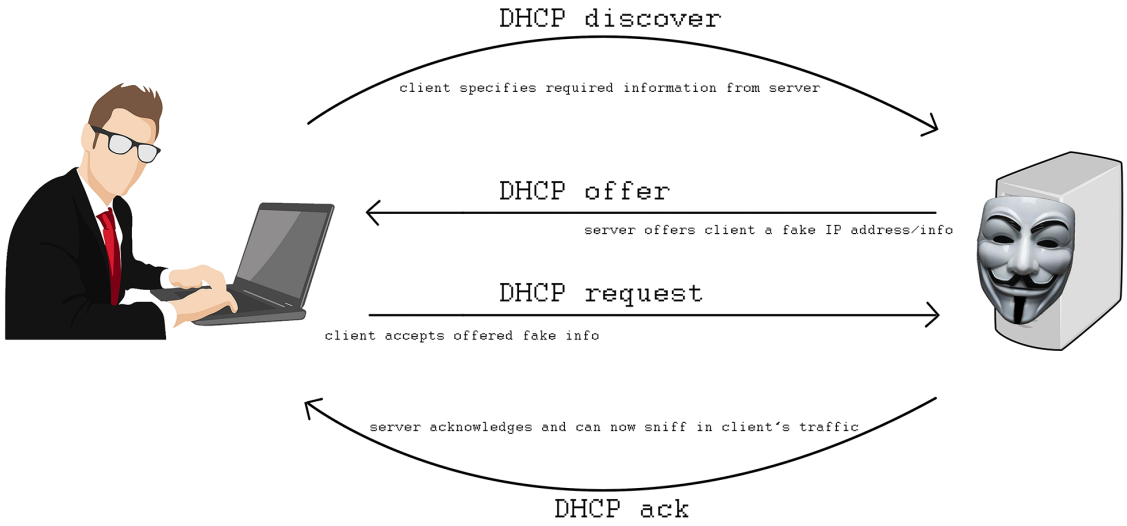
1.2 DHCP rogue server

Popsat tento útok je velmi jednoduché - v podstatě jde „jen“ o to implementovat obyčejný DHCP server s tím rozdílem, že obětem nabízí falešné informace - v našem případě IP adresu, adresu defaultní brány a DNS serveru, hostitelské jméno a dobu platnosti.

Princip tedy spočívá opět v *DORA* cyklu - oběť zasílá pomocí broadcastu *discover*, falešný server odpovídá zprávou typu *offer*, oběť stvrdí zaslané informace pomocí *requestu* a server nakonec zasílá *ack*.

Pokud server klientovi nabídne jako defaultní bránu (případně DNS server) své informace tak, aby komunikace šla přes něj, snadno se může dostat k citlivým informacím. [4] Útok popisuje obrázek 1.3.

Proti tomuto typu útoku se lze bránit technikou, které se říká *DHCP snooping*, která má za útok detekovat tyto rogue servery a případně zlikvidovat pakety přicházející od tohoto serveru. [1]



Obrázek 1.3: Rogue DHCP server

Kapitola 2

Implementace

Má implementace útoků využívá standardních prvků jazyka *C++* a pro práci s pakety knihovnu *libpcap* sloužící jak pro přijímání, tak odesílání paketů. Aplikace používají dvě vlákna - jedno pro přijímání paketů, druhé pro tvorbu a zasílání paketů. V této kapitole budu nebudu popisovat základní věci jako tvorba, odesílání paketů, zpracování argumentů a jiné, jelikož tyto věci byly vyzkoušeny, případně zdokumentovány, již v předmětech *IPK* či *ISA*. Místo toho popíšu kostru obou aplikací, případně princip prvků implementovaných nad rámec základního zadání.

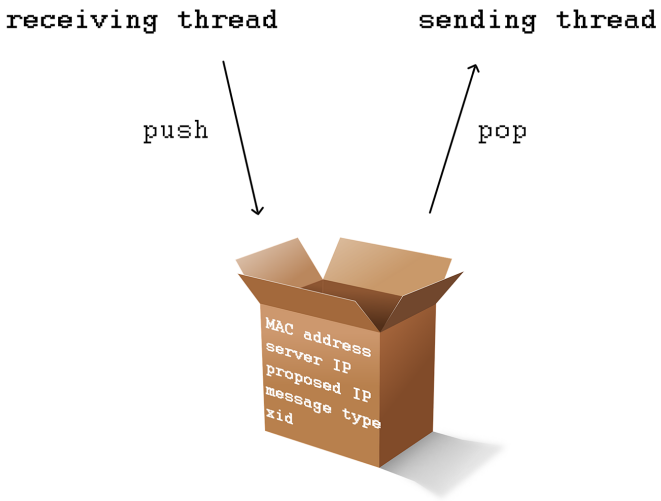
2.1 Práce s vlákny

Obě části útoku využívají společné třídy `Packet`, která zpracovává DHCP pakety a také je do sítě vkládá. Tato třída v sobě nese klíčový prvek celé aplikace, a to vektor obsahující záznamy, na základě kterých jsou tvořeny všechny pakety. Jelikož aplikace využívá více vláken a ne více procesů, je tento vektor sdílený (k jeho přístupu a modifikacím je implementován výlučný přístup za pomoci *mutexu*). Informace potřebné k vytvoření paketu jsou zdrojová MAC adresa, nabízená IP adresa, IP adresa serveru (nutná pouze pro *starvation* útok), identifikátor komunikace a v neposlední řadě typ zprávy (*discover*, *offer*, *request* nebo *ack*).

Princip je tedy takový: vlákno přijímající pakety přijme paket, zpracuje z něj potřebné informace a vloží je do sdíleného vektoru. Poté druhé vlákno, které pakety odesílá, zkontroluje, jestli je vektor neprázdný. V takové situaci odstraní první záznam a na základě jeho informací vytvoří a odešle paket. Princip je graficky znázorněn na obrázku 2.1. Výhoda této třídy tkví v tom, že je navržena tak, aby ji mohly využívat obě aplikace.

2.2 DHCP starvation

Má implementace tohoto útoku je založena na celém *DORA* cyklu. Generický princip funkčnosti obou aplikací byl již popsán výše, zde je tedy nutné popsat rozdíly. Vlákno pro odesílání čte ve smyčce zmíněný vektor. V situaci, kdy je vektor prázdný, tedy není nutné odpovídat na žádný *offer*, aplikace generuje MAC adresy a snaží se server zaplavit zprávami typu *discover*.



Obrázek 2.1: Sdílený vektor obsahující informace pro tvorbu paketů.

2.3 DHCP rogue server

Mimo generického principu funkčnosti server přidává další funkce. Po zapnutí aplikace je vytvořen vektor obsahující všechny možné IP adresy, které bude server nabízet, na základě parametru `-p`. Každý záznam v tomto vektoru mimo IP adresy obsahuje informace o tom, zda je adresa volná, zda byla nabídnuta (*offer*) či zda byla potvrzena (*ack*). Dalším atributem je MAC adresa (když již byla adresa nabídnuta a oběť zaslala *request*, nebo pro případ, kdy klient zasílá pouze *request* pro prodloužení platnosti) a časové razítko, kdy byla adresa nabídnuta, případně potvrzena.

Po vytvoření paketu server aktualizuje svůj IP pool. Pokud již některé z přidělených adres prošla platnost (argument `-l`), uvolní se tato adresa k dalšímu nabízení - stejně tak jako pokud byla nabídnuta, ale nepotvrzena, tzn. oběť nezaslala *request* (tuto dobu jsem stanovil jako dvě minuty).

Program navíc počítá dle IP poolu zadaného parametrem `-p` masku podsítě - hledá pozici prvního bitu, kde se počáteční a koncová IP adresa liší, a na základě této pozice provede bitový posuv masky `255.255.255.255`.

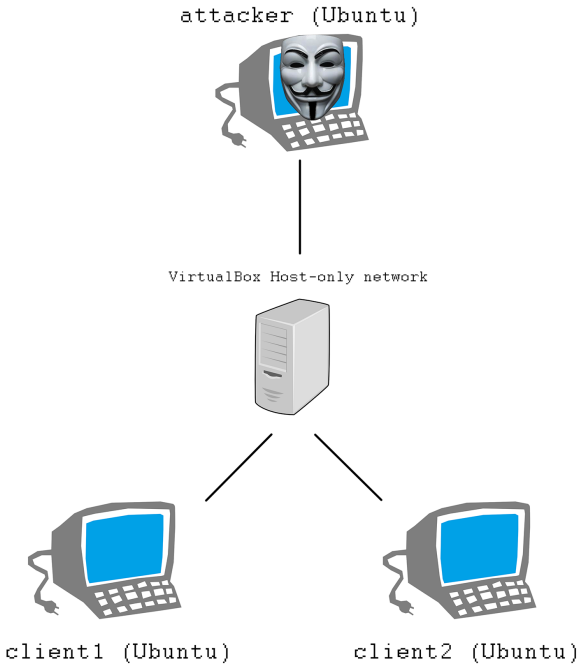
Kapitola 3

Demonstrace

3.1 Testovací prostředí

Obě aplikace jsem testoval ve virtuální síti propojené pomocí *VirtualBox host-only network* adaptéru. V síti byly připojené tři virtuální počítače, všechny nesoucí operační systém *Ubuntu* - dvě oběti a jeden útočník. Topologii sítě znázorňuje obrázek 3.1. Výchozí DHCP server *VirtualBoxu* byl vypnut, namísto toho byl nainstalován a nakonfigurován *isc-dhcp-server* (konfigurace popsána obrázkem 3.2).

Po zkušebním testování v referenčním prostředí zorganizovaném k předmětu jsem zjistil, že *Windows* klienti nereagují na zprávy mého rogue serveru. Do sítě jsem tedy přidal i klienta s operačním systémem *Windows 7*, chybu lokalizoval a opravil, materiál k demonstraci již byl však vytvořený, budu zde tedy popisovat síť bez klienta s *Windows*. Podobný případ se odehrál také s implementací výpočtu masky podsítě.



Obrázek 3.1: Topologie testovací sítě.

```
subnet 192.168.0.0 netmask 255.255.255.0 {
        range 192.168.0.10 192.168.0.25;
        option routers  192.168.0.1;
        option subnet-mask      255.255.255.0;
        option broadcast-address        192.168.0.255;
        option domain-name-servers      192.168.0.2;
        default-lease-time 86400;
        max-lease-time 86400;
}
```

Obrázek 3.2: Konfigurace DHCP serveru.

Síťovou konfiguraci před útokem zachycuje obrázek 3.3.

```
enp0s3    Link encap:Ethernet  HWadr 08:00:27:1f:e8:70
          inet adr:192.168.0.10  Všesměr:192.168.0.255  Maska:255.255.255.0
          inet6-adr: fe80::e40d:473d:3f2d:a17e/64 Rozsah:Linka
          AKTIVOVÁNO VŠESMĚROVÉ_VYSÍLÁNÍ BĚŽÍ MULTICAST  MTU:1500  Metrika:1
          RX packets:302 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1690 errors:0 dropped:0 overruns:0 carrier:0
          kolizí:0 délka odchozí fronty:1000
          Přijato bajtů: 24874 (24.8 KB) Odesláno bajtů: 115438 (115.4 KB)

enp0s3    Link encap:Ethernet  HWadr 08:00:27:2d:cb:82
          inet adr:192.168.0.12  Všesměr:192.168.0.255  Maska:255.255.255.0
          inet6-adr: fe80::7eb4:384f:ae34:38f8/64 Rozsah:Linka
          AKTIVOVÁNO VŠESMĚROVÉ_VYSÍLÁNÍ BĚŽÍ MULTICAST  MTU:1500  Metrika:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:106 errors:0 dropped:0 overruns:0 carrier:0
          kolizí:0 délka odchozí fronty:1000
          Přijato bajtů: 2006 (2.0 KB) Odesláno bajtů: 10323 (10.3 KB)
```

Obrázek 3.3: Konfigurace klienta 1 a klienta 2 před útokem.

Následně obě oběti daly vědět legitimnímu DHCP serveru o vypuštění přidělených IP adres příkazem `dhclient -r`. Poté byl spuštěn útok *DHCP starvation*. Jak lze vidět na obrázku 3.4, po vyčerpání nakonfigurovaného IP poolu již server přestal odpovídat, *denial of service* tedy zafungoval.

```
4318 17238.631917 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4321 17239.634064 192.168.0.11     192.168.0.22       DHCP     342 DHCP Offer    - Transaction ID 0x0
4323 17243.732877 0.0.0.0          255.255.255.255    DHCP     300 DHCP Request  - Transaction ID 0x0
4324 17243.740476 192.168.0.11     192.168.0.22       DHCP     342 DHCP ACK      - Transaction ID 0x0
4325 17248.833216 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4328 17249.833908 192.168.0.11     192.168.0.23       DHCP     342 DHCP Offer    - Transaction ID 0x0
4330 17253.933939 0.0.0.0          255.255.255.255    DHCP     300 DHCP Request  - Transaction ID 0x0
4331 17253.941636 192.168.0.11     192.168.0.23       DHCP     342 DHCP ACK      - Transaction ID 0x0
4332 17259.034045 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4335 17260.035392 192.168.0.11     192.168.0.24       DHCP     342 DHCP Offer    - Transaction ID 0x0
4337 17264.135362 0.0.0.0          255.255.255.255    DHCP     300 DHCP Request  - Transaction ID 0x0
4338 17264.143135 192.168.0.11     192.168.0.24       DHCP     342 DHCP ACK      - Transaction ID 0x0
4339 17269.236398 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4342 17270.237279 192.168.0.11     192.168.0.25       DHCP     342 DHCP Offer    - Transaction ID 0x0
4352 17274.338142 0.0.0.0          255.255.255.255    DHCP     300 DHCP Request  - Transaction ID 0x0
4353 17274.346227 192.168.0.11     192.168.0.25       DHCP     342 DHCP ACK      - Transaction ID 0x0
4354 17279.438275 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4356 17280.439677 192.168.0.11     192.168.0.10       DHCP     342 DHCP Offer    - Transaction ID 0x0
4358 17284.539165 0.0.0.0          255.255.255.255    DHCP     300 DHCP Request  - Transaction ID 0x0
4359 17284.547262 192.168.0.11     192.168.0.10       DHCP     342 DHCP ACK      - Transaction ID 0x0
4362 17289.639392 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4363 17294.740562 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4367 17304.941652 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4368 17310.041893 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
4369 17315.142622 0.0.0.0          255.255.255.255    DHCP     288 DHCP Discover - Transaction ID 0x0
```

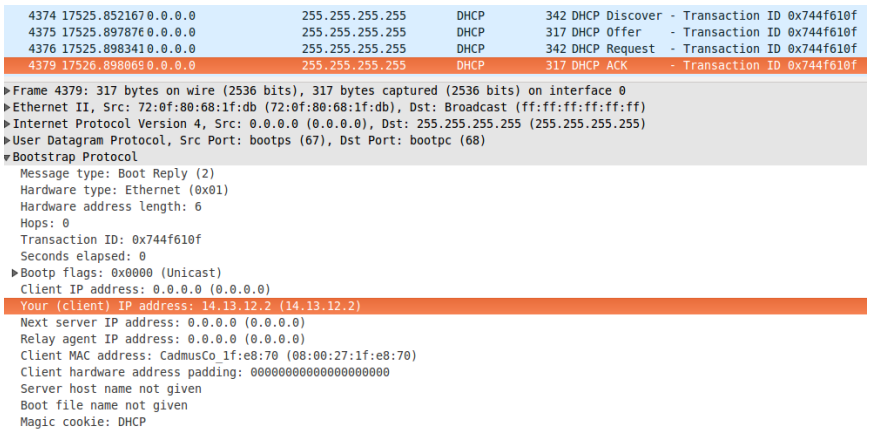
Obrázek 3.4: Wireshark zachycující útok *starvation*.

V této chvíli je tedy čas na spuštění rogue serveru. Ten byl spuštěn s následujícími paremtry (obrázek 3.5):

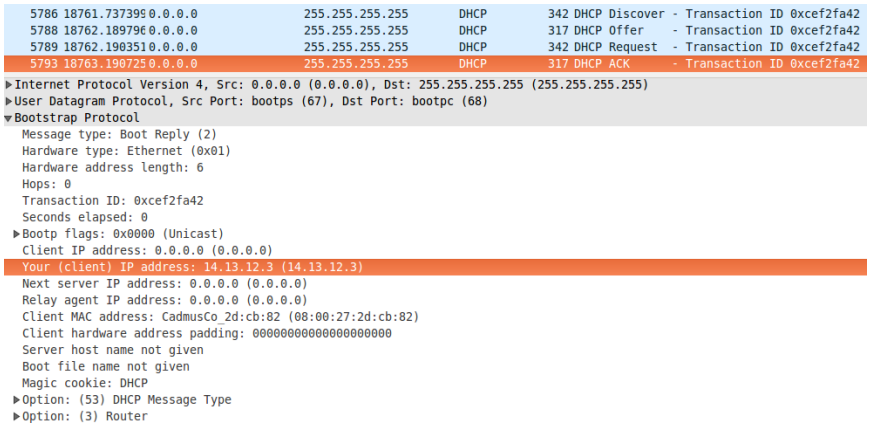
```
./pds-dhcprogue -p 14.13.12.2-14.13.12.88 -d rogue.com -g 14.13.12.0 -n 14.13.12.1 -l 5000 -i eth0
```

Obrázek 3.5: Spuštění rogue DHCP serveru.

Poté oběti zažádaly o nové IP adresy příkazem `dhclient`. Jelikož pool legitimního DHCP serveru byl již vyčerpán, komunikovaly s rogue serverem (obrázek 3.6, resp. 3.7).



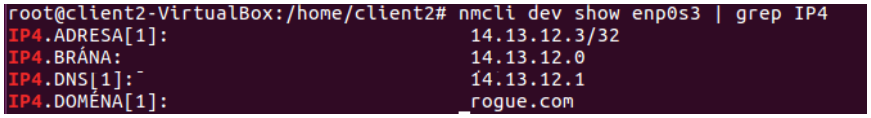
Obrázek 3.6: Oběť 1 žádá o IP adresu.



Obrázek 3.7: Oběť 2 žádá o IP adresu.

Síťovou konfiguraci obětí po dokončení komunikace popisují obrázky 3.8, resp. 3.9.

Obrázek 3.8: Síťová konfigurace oběti 1.



Obrázek 3.9: Síťová konfigurace oběti 2.

Literatura

[1] Banks, E.: *Five Things To Know About DHCP Snooping.*
URL http://packetpushers.net/five-things-to-know-about-dhcp-snooping/

[2] Blacklabs Security: *DHCP Starvation.*
URL http://www.blacklabssecurity.info/dhcp-starvation.html

[3] Droms, R.; RFC 2131: *Dynamic Host Configuration Protocol.*
URL https://tools.ietf.org/html/rfc2131

[4] Straatsma, P.: *Network Takedown Part 2: Rogue DHCP Server with DHCP Starvation and Rogue Routing.*
URL http://www.hackandtinker.net/2013/11/27/going-rogue/