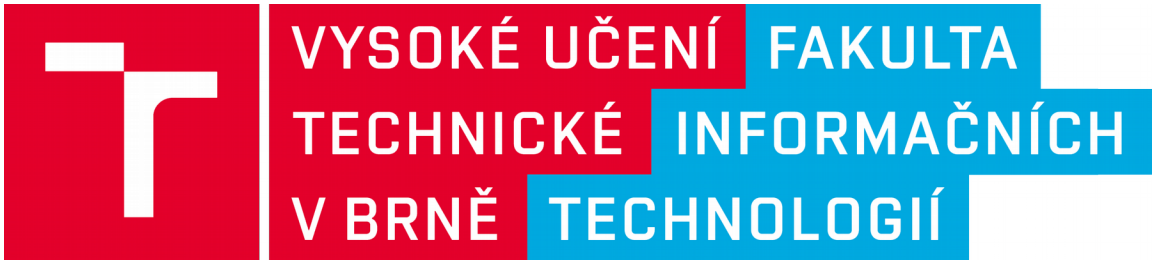


POČÍTAČOVÉ SÍTĚ A KOMUNIKACE

2016/2017

Projekt 2, Varianta 1

ARP Scanner

Vypracoval: Cagaš Martin

Zadávající: Veselý Vladimír, Ing., Ph.D.

Brno, 15. května 2017

Obsah

1 Zadání

Cílem projektu bylo vytvořit komunikující síťovou aplikaci za použití BSD socketů. Aplikace naprogramovaná v C/C++ má za použití RAW socketů a protokolu ARP proskenovat lokální síť a objevit MAC adresy všech existujících zařízení. Výstupem aplikace je XML dokument obsahují MAC adresy nalezených zařízení a jejich mapování na protokolové adresy.

2 Abstrakt

ARP skener je aplikace umožňující proskenování celé lokální sítě a odhalení zařízení na této síti. Je založena na protokolu ARP a využívá RAW BSD socketů pro posílání i přijímání datagramů. Úkolem aplikace je sestavit korektní datagram a poslat jej na každou IP v síti, přičemž si zaznamenává veškeré odpovědi, které obdrží. Aplikace se používá v oblastech, kde uživatel nemá administrátorský píštup pro zjištění struktury sítě, jak jsou například přípravy na síťové útoky.

3 Úvod

V běžném provozu je velmi běžná situace, že je potřeba zaslat datagram adresátovi na stejné podsíti, v jaké jsme my sami. Abychom mohli zaslat datagram přeš Ethernet, kterého mimo jiné využívá i Wi-Fi standard 802.11, potřebujeme znát MAC (Media Access Control) adresu cílového zařízení. Zpravidla ale známe pouze protokolovou (IP) adresu. Řešením je ARP protokol, který slouží k získání MAC adresy na základě známé IP adresy. Když zařízení přijme ARP požadavek, je za běžných podímek nutno na něj odpověděť. Pakliže by tak neučinilo, žadatel by nemohl se zařízením komunikovat. Tohoto faktu právě využívá program běžně označovaný jako ARP skener – software schopný získat MAC adresy všech zařízení připojených do sítě.

3.1 Protokol ARP

Protokol ARP (Address Resolution Protocol) poprvé navrhnul David C. Plummer v dokumentu RFC 826 v listopadu roku 1982.

ARP protokol je dnes integritní součástí veškeré IPv4 komunikace, neboť zprostředkovává propojení mezi linkvou a síťovou vrstvou. Zmínění IPv4 přitom neznamená, že by podobný mechanismus nebyl potřeba v IPv6. Tam tuto funkcionalitu pouze vyplňuje NDP (Neighbour Discovery Protocol), který není v tomto projektu implementován.

V základu je to jednoduchý bezstavový protokol typu dotaz-odpověď. Pokud je při komunikaci potřeba získat hardware adresu cílového rozhraní, je vygenerován jednoduchý ARP request packet, který je poslán do sítě na broadcast. Tento packet tak dostanou všechna zařízení na síti. Pokud se request zařízení týká, odešle zpět odesílateli (jehož hardware i protokolová adresa jsou v packetu uvedeny) ARP response a současně si tyto adresy zapíše do ARP cache. Obdobně učiní i odesílatel a další komunikace už probíhá pouze mezi nimi.

Myšlenkou za vznikem samotného protokolu je vytvoření mezivrstvy, která by dovolila na požádání zjistit cílovou MAC adresu. Při tom se vyhnula nutnosti neustálého rozesílání „odpovědí“ (jsou pouze na vyžádání) a byla nezávislá na síťové vrstvě a přehršli protokolů, které se v době vzniku ARP protokolu používaly.

3.2 Struktura packetu

Podle původní specifikace RFC 826 je struktura ARP packetu následující:

```
Ethernet transmission layer (not necessarily accessible to
        the user):
        48.bit: Ethernet address of destination
        48.bit: Ethernet address of sender
        16.bit: Protocol type = ether_type$ADDRESS_RESOLUTION
Ethernet packet data:
        16.bit: (ar$hrd) Hardware address space (e.g., Ethernet,
                         packet Radio Net.)
        16.bit: (ar$pro) Protocol address space.  For Ethernet
                         hardware, this is from the set of type
                         fields ether_typ$<protocol>.
         8.bit: (ar$hln) byte length of each hardware address
         8.bit: (ar$pln) byte length of each protocol address
        16.bit: (ar$op)  opcode (ares_op$REQUEST | ares_op$REPLY)
        nbytes: (ar$sha) Hardware address of sender of this
                         packet, n from the ar$hln field.
        mbytes: (ar$spa) Protocol address of sender of this
                         packet, m from the ar$pln field.
        nbytes: (ar$tha) Hardware address of target of this
                         packet (if known).
        mbytes: (ar$tpa) Protocol address of target.
```

Struktura ARP packetu je definována dokumentem RFC 826 (Plummer, 1982)

Ethernet transmission layer obsahuje klasickou ethernetovou hlavičku packetu. Cílová adresa u ARP protokolu není známa, proto je nastavena na **ff:ff:ff:ff:ff:ff** (broadcast). Adresa odesílatele je nastavena na odpovídající adresu. Protocol type je podle standardu nastaven na odpovídající hodnotu **0x0806**.

Ethernetový náklad podle specifikace samotného ARP protokolu je vyplňován následovně:

```
(ar$hrd) je nastaven podle typu hardware protokolu. V případě Ethernetu je
     to o 1.
(ar$pro) je nastaven podle typu software protokolu. V případě IPv4 je to
     0x0800 (2048).
(ar$hln) je délka ethernetové adresy v octetech. Konstantně je to 6 byte.
(ar$pln) je délka protokolové adresy v octetech. Konstantně je to 4 byte.
(ar$sha) a (ar$spa), tedy MAC a IP adresa odesílatele, jsou vyplněny
     odpovídajícími informacemi. Špatné uvedení těchto hodnot způsobí
     nedoručení odpovědi.
(ar$tha) je MAC adresa cílového zařízení. Jelikož je pro nás zatím neznámá,
     je tato hodnota nastavena na 00:00:00:00:00:00.
(ar$tpa) je IP adresa cílového zařízení. Je nastavena podle IP adresy
     zařízní, jehož MAC adresu je potřeba zjistit.
```

3.3 ARP Scanner

Úkolem programu ARP scanner je odhalit všechny zařízení na síti pomocí protokolu ARP. Úspěšnost této akce však stále závisí na nastavení hledaného zařízení v odpovídání na ARP dotazy. Jednotlivé úkony, které musí tento program vykonat budou dále popsány v části zabývající se samotnou implementací.

Účel takovéto aplikace je získat povědomí o zařízeních připojených k místní síti.

Praktická aplikace může sloužit k monitorování zařízení připojujících se k síti. Toho je ale mnohem lepší a jednoduší pro správce sítě dosáhnout zaznamenáváním těchto informací na straně routeru. V praxi je ARP scan zpravidla používán uživateli, kteří v síti nemají administrátorský status, pro účely síťových počítačových útoků, jako je například „otrávení ARP cache“, jeden z Man in the Middle útoků. Takovýto úspěšný útok přesměruje komunikaci napadeného zařízení přes útočníkův počítač. Útočník se tak dostává ke všem nezabezpečeným datům oběti, kde mohou být mimo jiné i přihlašovací údaje, hesla a jiné citlivé informace přenášené přes nezabezpečená spojení.

I přes takováto rizika není možné ARP dotazy odmítat, jak již bylo zmíněno, jsou páteřní funkcí síťové komunikace.

4 Implementace

Program je implementován v jazyce C, sestavován za použití GCC překladače a nástroje Make.

4.1 Poznámka k přenositelnosti mezi systémy

Program je navržen pro fungování na všech systémech UNIXového typu. Jelikož ale využívá velice nízko-úrovňových RAW socketů, je jeho funkcionalita limitována podporou těchto socketů v cílovém systému.

Ačkoli program funguje na standartních distribucích Linuxu, vyskytly se v některých specifických případech problémy. Jmenovitě *FreeBSD* nepodporuje použití RAW socketů pro jakékoli protokoly, které kernel sám implementuje a ARP mezi ně beze sporu patří. Dále pak *Microsoft Bash on Windows* rovnou nepodporuje samotné RAW sockety. U ostatních systému nebyly zjištěny problémy s kompatibilitou.

4.2 Parametry programu

Program se spouští s následujícímy parametry:

```
[sudo] ./ipk-scanner -i interface -f file # kde:
# interface (string) specifikuje rozhraní použité pro skenování
# file (path) určuje soubor pro uložení výstupu
```

Poznámky:

- Vytvoření RAW socketu vyžaduje administrátorská oprávnění.
- Pokud již zadaný soubor existuje, bude přepsán. Pokud ne, bude vytvořen.

4.3 Souborová struktura

Program je rozdělen do souborů

```
main.c
     Obsahuje samotný kód a funkční algoritmus.

main.h
     Hlavičkový soubor. Obsahuje importy, prototypy funkcí a definice
některých konstant.

arph.h
     Obsahuje definici struktury představující ARP packet.

ethh.h
     Obsahuje definici struktury představující ethernetovou hlavičku.
```

4.4 Funkční bloky

4.4.1 Získání IP a MAC adresy rozhraní

Základním problémem celého programu je naplnění struktury packetu odpovídajícími daty. Proto je potřeba nejprve zjistit MAC i IP adresu rozhraní, přes které budeme packety zasílat. Dále pak je potřeba zjistitt masku sítě pro výpočet celkového počtu adres v síti.

Tato problematika je implementována vytvořením socketu a následným voláním funkce *ioctl* pro každou informaci, kterou je potřeba zjistit.

Po dokončení této operace je možné socket bezpečně uzavřít.

4.4.2 Zjištění celkového počtu adres v síti

Celkový počet adres v síti je zjištěn pomocí masky sítě. Dále je vypočítána adresa sítě, která je následně použita pro iteraci přes všechny adresy v sítě s jednoduchým počítadlem.

Pole *src_ip* a *net_mask* obsahují jednotlivé octety.

```
uint32_t mask;
uint32_t ip;
uint32_t count;
ip = (uint32_t) src_ip[3] + (((uint32_t) src_ip[2]) << 8) + (((uint32_t)
src_ip[1]) << 16) + (((uint32_t) src_ip[0]) << 24);
mask = (uint32_t) net_mask[3] + (((uint32_t) net_mask[2]) << 8) +
(((uint32_t) net_mask[1]) << 16) + (((uint32_t) net_mask[0]) << 24);
uint32_t network_addr = ip & mask;
count = ~mask;
```

Proměná *mask* obsahuje masku sítě.

Proměná *count* počet IP adres v síti. Iterování bode probíhat do tohoto počtu. Počet lze získat z masky jednoduchou bitovou inverzí.

4.4.3 Naplnění statické části ARP packetu

```
/** FROM main.h
 * #define ETH_P_ARP 0x0806
 */

// ETHER header
memset(&ethhdr.dest, 0xff, 6 * sizeof (uint8_t));
memcpy(&ethhdr.source, src_mac, 6 * sizeof (uint8_t));
ethhdr.eth_type[0] = ETH_P_ARP / 256;
ethhdr.eth_type[1] = ETH_P_ARP % 256;

// ARP header
arphdr.htype = htons(1);
arphdr.ptype = htons(2048);
arphdr.hlen = 6;
arphdr.plen = 4;
arphdr.op = htons(1);
memcpy(&arphdr.sha, src_mac, 6 * sizeof (uint8_t));
memcpy(&arphdr.spa, src_ip, 4 * sizeof (uint8_t));
memset(&arphdr.tha, 0x00, 6 * sizeof (uint8_t));
```

Tuto část lze naplnit a nechat konstantní, protože se pro jednotlivé ARP dotazy nemění.

4.4.4 Iterace přes celý adresový prostor

Je zajištěna jednoduchou *for* smyčkou. Využívá se znalosti adresy sítě a celkového počtu adres.

```
for (uint32_t i = 1; i < count; i++) {
     ip = network_addr | i;
     arphdr.tpa[0] = ip >> 24;
     arphdr.tpa[1] = (ip >> 16) & 0xff;
     arphdr.tpa[2] = (ip >> 8) & 0xff;
     arphdr.tpa[3] = ip & 0xff;
     // ...
}
```

Ple cílové protokolové adresy struktury ARP hlavičky je poté naplněnou takto získanou hodnotou.

4.4.5 Přijímání odpovědi

Příjímání odpovědi se potýká se dvěma problémy. Jak zajistit, abychom neminuli odpověď. Operační systém je velmi nevyzpytatelný. Jak zajistit, abychom nedostali přerušení v momentě, kdy jsme odeslali požadavek ale ještě nezačali čekat na odpověď. Druhým problémem je volba vhodného časového omezení. Čekání na odpověď musíme přerušit ve vhodný moment. Dotazy totiž pozíláme slepě na všechny adresy sítě a nemáme způsob, jak zjistit jestli nám daná adresa neodpověděla, protože neexistuje cílové zařízení, nebo protože se odpověď jen zdržela na síti.

První problém je řešen pomocí vytvoření druhého procesu. Ten se vytvoří a okamžitě připraví na poslouchání odpovědi. Až potom se vyšle ARP požadavek. V případě úspěchu child proces zapíše do výstupního souboru a ukončí se. Rodič po odeslání čeká na ukončení potomka a nepokračuje dále ve smyčce.

Druhý problém je vyřešen nastavením časového limitu na přijímací socket. Ten bylo potřeba nastavit rozumně – socket přijímá veškerou síťovou dopravu ale úspěch zaznamená pouze v případě, že přijatý packet byl protokolu ARP. Pokud nebyl, socket se ve smyčce opět připraví na poslouchání. Tento jev ale samozřejmě obnoví časový limit. Tento časový limit je nastaven na *50 ms*.

```
reciever = fork();
if (reciever < 0) {
     // ...
} else if (reciever == 0) {
     while (1) {
                signal(SIGINT, SIG_IGN);
                int recieve = recv(recv_socket, recv_ether_frame,
IP_MAXPACKET, 0);
                // ...
}
```

4.4.6 Výstup do souboru

Výstup do souboru je řešen na bázi volného zápisu. Před počátkem prohledávání je do souboru vložená XML hlavička a otevírací tag kořenového elementu. Po ukončení smyčky je vložen uzavírací tag kořenového elementu. Pokud poslouchací proces nahlásí úspěch, vytvoří nový zápis. Vždy po zápisu je soubor opětovně uzavřen, proto je potřeba jej opakovaně otevírat před zápisem.

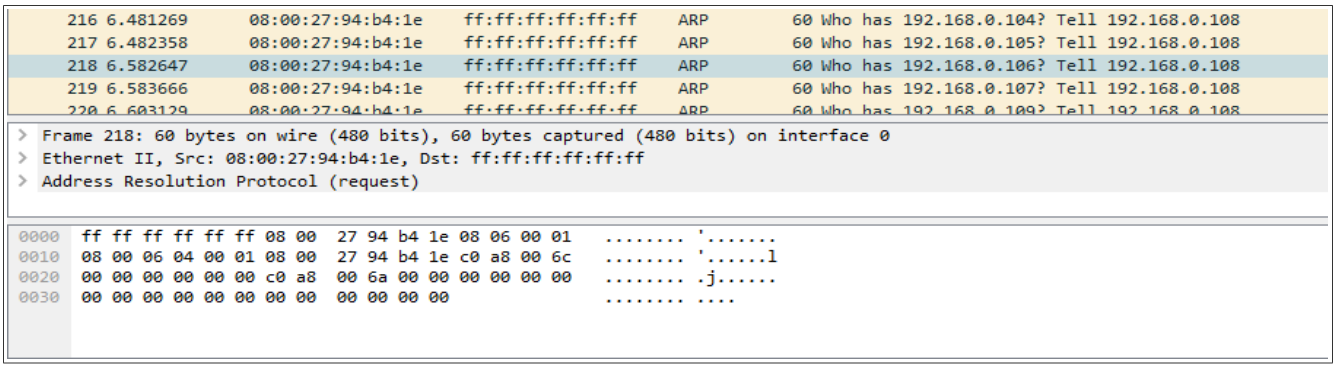
Příklad kódu formátujícího výstup:

```
output = fopen(filepath, "a");

fprintf(output, "\t<host mac=\"%02x:%02x:%02x:%02x:%02x:%02x\">\n",
recv_arphdr->sha[0], recv_arphdr->sha[1], recv_arphdr->sha[2], recv_arphdr-
>sha[3], recv_arphdr->sha[4], recv_arphdr->sha[5]);

fprintf(output, "\t\t<ipv4>%u.%u.%u.%u</ipv4>\n", recv_arphdr->spa[0],
recv_arphdr->spa[1], recv_arphdr->spa[2], recv_arphdr->spa[3]);

fprintf(output, "\t</host>\n");
fclose(output);
```

Kde *recv_arphdr* je ukazatel do paměti s uloženým přijatým packetem.

4.5 Příklad použití

Příklad ARP dotazů generovaných programem zachycených v programu Wireshark (wireshark.org) na druhém počítači používajícím systém Windows:



5 Závěr

Projekt kombinuje vzdělávací záměr s praktickým využitím, což nebylo doposud v našem studiu běžné. Znalosti fungování sítí a síťových útoků jsou nutné ne jen na samotné provádění, ale i na obranu, která se v době, kdy přesouváme čím dál, tím více našich osobních životů do počítačů, stává naprosto nepostradatelnou.

Má implementace projektu nepatří mezi nejrychlejší, ale doufám, že ji tento nedostatek vybaví větší bytelností a odolností na síťové chyby, jako je zpoždění packetů.

6 Bibliografie

1. PLUMMER, David C. *RFC 826* [online]. 1982-11 [cit. 2017-05-15]. Dostupné z: https://tools.ietf.org/html/rfc826
2. VESELÝ, Vladimír. *ipk2017L-p08-linkova.pdf*. Brno: Přednáška pro FIT, 2017.
3. OČENÁŠEK, Pavel. *ibs03.pdf*. Brno: Přednáška pro FIT, 2017.
4. Wikipedia: The Free Encyclopedia (Wikimedia Foundation Inc.) *Address Resolution Protocol* [onlne]. aktualizováno 2017-05-15 [cit. 2017-05-15]. Dostupné z: https://en.wikipedia.org/wiki/Address_Resolution_Protocol
5. Wikipedia: The Free Encyclopedia (Wikimedia Foundation Inc.) *ARP spoofing* [onlne]. aktualizováno 2017-05-01 [cit. 2017-05-15]. Dostupné z: https://en.wikipedia.org/wiki/ARP_spoofing
6. Wireshark Wiki (The Wireshark Team) *Address Resolution Protocol (ARP)* [onlne]. aktualizováno 2017-05-14 [cit. 2017-05-15]. Dostupné z: https://wiki.wireshark.org/AddressResolutionProtocol

Pokud není uvedeno jinak, kód i ilustrace pochází z vlastních archivů autora.