Fakulta informačních technologií
Vysoké učení technické v Brně

Dokumentace k projektu

Mapování adresového prostoru IPv6 pomocí OUI

19. listopadu 2012

Vojtěch Dlápal
xdlapa01@stud.fit.vutbr.cz

Obsah

1 Úvod **2**

2 Úvod do problematiky **2**
2.1 OUI 2
2.2 EUI-64 2
2.3 ICMPv6 2

3 Návrh aplikace **2**
3.1 Generování adres 2
3.2 Odesílání paketů 3
3.3 Příjem paketů 3

4 Popis implementace **3**
4.1 Generování adres 3
4.2 Odesílání paketů 4
4.3 Příjem paketů 4

5 Základní informace o programu a testování **4**

6 Návod k použití **4**

1 Úvod

Tato dokumentace popisuje postup práce a implementace při realizaci projektu Mapování adresového prostoru IPv6 pomocí OUI do kurzu ISA – Síťové aplikace a správa sítí, kdy obecným úkolem bylo vytvořit komunikující aplikaci podle konkrétní vybrané specifikace pomocí síťové knihovny BSD sockets, a to v jazyce C/C++.

2 Úvod do problematiky

Vytvořený program má vyhledávat modifikované EUI-64 adresy vytvořené ze zadaných OUI v zadané síti, a to pomocí ICMPv6 zpráv typu Echo Request a takto nalezené adresy vypisovat na standardní výstup spolu s případným popisem uvedeným na řádku s příslušným OUI v souboru se zadanými OUI. V první řadě tedy bylo nutné nastudovat, co je to OUI a jak vytvořit modifikovanou EUI-64 adresu.

2.1 OUI

OUI neboli Organizationally Unique Identifier je unikátní 24bitové číslo, jednoznačně identifikující organizaci, která si toto zaregistrovala. To je zapisováno jako hexadecimální číslo o šesti oktetech oddělených po dvou pomlčkou, a nebo dvojtečkou která reprezentuje bitově obrácený a nekanonický zápis [6], který je využíván a výhradně podporován v mém programu, a to z důvodu jednoduššího převodu na číslo pro porovnání s náležitou částí příchozí adresy a také jednoduššího otestování vzhledem k splnění požadavků vzhledem k příkladům zveřejněným k zadání.

2.2 EUI-64

Modifikované EUI-64 neboli IEEE-defined 64-bit extended unique identifier je 64bitové číslo složené z OUI a rozšířeným identifikátorem, který přiděluje organizace vlastnící OUI. [1] Samotné složení probíhá rozšířením OUI o číslo 0xFFFE za tímto číslem je samotný rozšířený identifikátor. Nakonec je třeba invertovat sedmý bit OUI. [2] Nás však zajímá pouze OUI a proto je potřeba generovat EUI-64 pro všechny rozšířené identifikátory s přihlédnutím k tomu, že prostřední část čísla, tedy číslo 0xfffe je vždy stejné.

2.3 ICMPv6

Vyhledávání probíhá pomocí odesílání ICMPv6 zpráv typu Echo request. ICMPv6 neboli Internet Control Message Protocol version 6 je implementace ICMP protokolu pro IPv6 sítě. Tento protokol definuje zprávy o stavu sítě. Hlavička ICMPv6 paketu obsahuje typ zprávy, kód zprávy a kontrolní součet. Pro zprávu Echo request je typ `ECHO_REQUEST`, kód zprávy 0 a kontrolní součet v mém případě nechávám počítat jádro operačního systému a proto jej nastavuji na 0. Hlavička tohoto typu dále obsahuje číslo sekvence a id, které je vhodné pro párování s příchozími zprávami typu Echo reply. [4] Hlavní myšlenkou skenování pomocí zasílání Echo request zpráv založená na skutečnosti, že pokud zařízení obdrží jemu náležící Echo request odešle zpátky zprávu typu Echo reply se stejným číslem sekvence. Je tedy možné rozeslat Request zprávy na danou množinu adres a podle příchozích Echo reply zpráv zjistíme, která zařízení byla aktivní.

3 Návrh aplikace

Dekompozicí jsem rozdělil aplikaci na základní části: generování adres, odesílání paketů, přijímání paketů a ostatní režie. Program po ověření parametrů příkazové řádky získá seznam hledaných OUI a dále vytvoří samostatné vlákno pro přijímání paketů. Toto vlákno bude ukončeno s ukončením celé aplikace a to s prodlevou kvůli zpožděným příchozích paketům. V hlavním vláknu probíhá generování adres a příprava a odesílání Echo request paketů.

3.1 Generování adres

Pro generování adres bylo potřeba rozdělit adresu na části a uvědomit si, které části se mění a které zůstávají vždy stejné. Z definice modifikovaného EUI-64 plyne, že v první řadě je třeba generovat poslední 3 byty adresy, kde se nachází rozšířený identifikátor (vendor supplied id) a ten může být jakýkoliv. Generuji se tedy inkrementací od nuly po maximální hodnotu, což je 0xFFFFFF. Poté následuje vložená část EUI-64, což jsou oktety 0xFE a 0xFF. Ta je vždy stejná, a proto je toto vytvořeno již s první vygenerovanou adresou a dále se tato část při generování přeskakuje.

Následují 3 oktety značící OUI. Proto se vždy při vyčerpání všech možností rozšířeného identifikátoru vloží nové OUI u kterého se invertuje sedmý bit, protože se jedná o modifikované EUI-64, a rozšířené identifikátory se pro toto generují znova. Zbývá vrchních 64 bitů adresy. Která část je neměnná záleží na zadaném prefixu a do této se vkládá

číslo z adresy zadaného adresového prostoru. Případná zbylá část se inkrementuje stejným způsobem jako rozšířené identifikátory do vyčerpání všech možností a to vždy o jedno po vyčerpání všech kombinací v předchozích částech adresy.

3.2 Odesílání paketů

Pro odesílání je použito socketů typu raw. Původně bylo zamýšleno použití knihovny libnet, ale od toho jsem po nastudování podrobností ustoupil, jelikož snadnou práci s ICMPv6 pakety umožňují až verze vyšší než libnet-1.1.6, a takovouto nebylo kvůli požadavkům zadání použít. Cílem bylo přenechat co nejvíce práce operačnímu systému, proto je v jeho režiji ponechána tvorba IPv6 hlavičky i počítání kontrolního součtu pro ICMPv6 hlavičku. Před odesláním bylo nutné implementovat zpoždění. To je provedeno jako uspání vlákna na uživatelem. Pro odeslání jednoho /64 bloku s jedním OUI do jedné hodiny, by bylo potřeba nastavit prodlevu menší než 214 micro sekund a to dle výpočtu 3600000000 / 0xFFFF. Takováto prodleva však nebyla na referenčním stroji OS Linux použitelná, více viz 5. Po nezdařeném odeslání jsem se rozhodl vložit prodlevu 3 sekundy, kterou jsem experimentováním určil jako nejmenší prodlevu po které se chyba neopakuje, a odeslat daný paket znova a zamezit tak ztrátě paketů na straně odesilatele.

3.3 Příjem paketů

Příjem paketů běží v samostatném vlákně. K příjmu je použit socket typu raw. To znamená, že na socket přichází veškerá ICMPv6 komunikace a to zejména Neighbor Solicitation pakety. To může znamenat rychlé zahlcení a ztrátu paketů. Tomu by šlo předejít filtrováním pomocí knihovy libpcap, avšak zde jsem se střetl s problémem nespolehlivosti funkce pro automatické rozpoznání výchozího rozhraní a nastavení rozhraní přímo by znamenalo omezení portability. Možností donutit uživatele rozhraní zadat pomocí povinného parametru příkazové řádky avšak toto jsem uznal jako zbytečné obtěžování uživatele a zbytečné zvyšování složitosti funkce pro zpracování parametrů vzhledem k nepovolení knihoven, které toto provádějí. K úplnému zamezení ztráty paketů v reálných podmínkách není možné dospět. Vzhledem k nutnosti používat prodlevy považuji stávající řešení za dostatečné.

Při příjmu Echo reply paketu zjistím adresu odesilatele a porovnám její OUI část se všemi OUI v seznamu zadaných OUI a v případě shody vypíšu adresu na standartní výstup s případným popisem. Nevyužívám tedy párování dotazů a odpovědí pomocí identifikátoru a čísla sekvence z ICMPv6 hlavičky. Tímto postupem je možné dospět k duplicitám, například pokud se na síti objeví Echo reply packet, který není odpovědí na žádost naší aplikace, ale toto je krajní případ a nejedná se o chybu, protože pokud z adresy přišel Echo reply paket, znamená to, že zařízení je aktivní, a tedy nás zajímá. Navíc výpis duplicit zadání nezakazuje.

4 Popis implementace

Implementace byla provedena v jazyce C++ za použití standardních knihoven a knihoven pro práci se sítí. Přenositelná je mezi systémy Linux a Unix (FreeBSD).

4.1 Generování adres

Pro práci s adresami používám objektový přístup, kvůli možnosti lépe zapouzdřit jednotlivé operace. Slouží k tomu třída `Adress`. Objekt této třídy si drží aktuální adresu jako bitové pole, ke kterému primárně přistupuji jako ke kontejneru `bitset<128>`. V konstruktoru se nastaví iterátor seznamu zadaných OUI na první záznam a také příznak první adresy, kvůli negenerování další adresy po prvním nastavení. Adresový prostor se nastaví zkopírováním zadané adresy převedené na číslo na adresu bitového pole objektu. S adresou pracuji v síťovém pořadí bitů, protože je to výhodnější z důvodu její jednodušší inkrementace a také přetypování na typ `in6 addr`.

K vygenerování další adresy slouží metoda `nextAddr()`, která v případě vyčerpání adres vrací NULL. V té dochází k využití dalších generujících metod popořadě pro inkrementaci rozšířeného identifikátoru, nastavení dalšího OUI, inkrementace adresy z adresového prostoru. Při možnosti vygenerovat další adresu níže postavenou metodou je tato adresa vrácena, jinak je její část adresy resetována a pokračuje se výše postavenou metodou.

Samotná inkrementace spočívá v průchodu bitovým polem a přičtení jedničky nastavením jedničky na správný bit. Procházím tedy z prava a pokud je na indexu jednička, nastavím na nulu a pokračuji na další index. Pokud je na místě nula nastavím ji na jedničku a vrátím adresu. Problémem bylo pořadí bitů v rámci jednotlivých bytů. Byty jsou seřazeny vhodně díky síťovému pořadí bytů. Proto jsem zavedl falešný index pomocí kterého přistupuji k bitům tak, jako kdyby bylo pořadí bitů v bitu obráceně a tedy tak, že celá adresa by byla zapsána jako jediné binární čílo s LSB na pravé straně. Tento index je korigován pro každý bit pomocí operace modulo. Kromě něj si udržuji také druhý index, který značí index bitu v reprezentaci adresy jako binárního čísla a pomocí něj určuji, zda jsem již dosáhl poslední adresy pro danou masku (čili masku 104 pro rozšířený identifikátor z EUI-64 nebo prefix sítě pro adresový prostor).

4.2 Odesílání paketů

Pro každou vygenerovanou adresu je vytvořen raw socket a pomocí něj odeslán paket s vloženou ICMPv6 hlavičkou. Ten je vždy stejný a proto je na všechny adresy poslán jeden předpřipravený. Bylo by možné pro všechny pakety použít jednou vytvořený socket, ale experimentováním jsem došel k závěru, že tento způsob je výhodnější z důvodu nižší nutné prodlevy mezi jednotlivými pakety nebo vyšším počtem odeslaných paketů bez prodlevy. Je možné, že toto je částečně způsobeno prodlevou tvořící navíc vytváření nového socketu v každém cyklu, avšak i tak je tato implementace výhodnější. Před každým odeslání je vlákno uspáno funkcí `usleep()` na čas zadaný uživatelem, nebo na výchozích 2500 mikro sekund. Při neúspěšném odeslání se vypíše chybové hlášení, provede se nastavená prodleva 3 sekundy a pokus se zopakuje.

4.3 Příjem paketů

Ve vláknu pro příjem paketů je vytvořen socket typu raw a zbytek se vykonává v nekonečném cyklu. Po příjmu paketu na socket pomocí funkce recvfrom je nahlédnuto do ICMPv6 hlavičky, a je zkontrolován typ zprávy a pokud se jedná o jinou zprávu, než Echo reply tak je okamžitě zahozena. V opačném případě je ze struktury s adresou vyextrahováno třetí 32bitové číslo obsahující OUI. Číslo s OUI je z tohoto získáno pomocí bitového posunu, který zahodí poslední dva oktety obsahující 0xFF a převrácením univerzálního bitu. Toto číslo se porovná s každým číslem OUI ze seznamu OUI, které se převede z řetězce převedením na textovou reprezentaci hexadecimálního čísla a následně na číslo pomocí proudu `stringstream`. Jelikož se jedná o 32bitová čísla je nutné řešit pořadí bytů a jedno z čísel pro porovnání převést. To je provedeno u čísla z příchozí adresy kvůli možnosti provést zmíněný bitový posun.

Při nalezení se vypíše adresa a popis jejího OUI na standardní výstup a pokračuje se v cyklu. Možné vylepšení je odsunout proces hledání OUI do samostatného vlákna kvůli urychlení výpočtu vedoucímu k příjmu dalšímu paketu a snížit tak riziko ztráty paketu. Toto jsem však neimplementoval, jelikož možnost šance, že přijde bezprostředně zasebou větší množství Echo reply paketů je nízká a nastaly by problémy se synchronizací vláken. Vlákno příjmu paketů se ukončí s hlavním programem po ukončení odesílání a vypršení prodlevy, kterou jsem empiricky nastavil na 6 sekund.

5 Základní informace o programu a testování

Program byl vyvíjen a testován na OS Linux na virtuální síti v prostředí Virtualbox a pomocí nástroje Wireshark. V tomto prostředí jsem narazil na několik problémů. Tím největším je nemožnost dosáhnout požadované rychlosti odesílání paketů. Pokud jsem Echo request pakety posílal pomocí předem vytvořeného raw socketu podařilo odeslat nejvíce 466 packetů, poté následovala chyba nedostatku místa ve vyrovnávácí paměti. Aby se tomuto zamezilo, bylo třeba nastavit prodlevu mezi odesláním jednotlivých paketů větší než 6000 mikro sekund.

Nyní je před každým odesláním paketu vytvořen socket nový. Chyba se objeví po odeslání 1022 paketů. Aby k ní nedošlo, je třeba nastavit prodlevu vyšší než 2500 mikro sekund, což je o řád víc než potřebných méně než 200 mikro sekund pro odeslání jednoho /64 bloku pod jednu hodinu. Částečným řešením může být navýšit velikost výstupních systémových bufferů. Pokud je prodleva uživatelem nastavena pod dostatečnou mez z důvodu urychlení, tak k tomu nedojde, jelikož při chybě při odesílání se provede druhý pokus o odeslání s prodlevou tří sekund, z důvodu zamezení ztráty paketů na výstupu. Na systému FreBSD se tyto problémy neprojevovali, ovšem testování nebylo dostatečně důkladné.

Dalším problémem může být zahození příchozího paketu s odpovědí z důvodu zahlcení sítě či zaplnění vstupní vyrovnávací paměti. To se dá částečně vyřešit zvýšením kapacity vstupní vyrovnávací paměti.

6 Návod k použití

Program je třeba spouštět s administrátorkými právy. Program byl vyvýjen a testován pod operačním systémem Linux. Překlad je třeba provést pomocí programu make a přiloženého Makefile.

Použití: ouisearch -p síť/rozsah -d filename [-s zpoždění v mikrosekundách]

Výchozí hodnotou zpoždění mezi odeslánimi je 2500 mikro sekund. Tato hodnota je také doporučena pro plynulý běh programu. Pokud je program nutné spustit tak, aby prohledával jeden /64 blok adres za čas menší než jedna hodina, je třeba jej spustit s parametrem -s 200, To je reálné pouze na testovacím systému FreeBSD, tam ovšem není projekt řádně otestován a proto nemohu ručit za případné další problémy.

OUI v souboru filename je třeba zadat ve formátu xx:xx:xx, každé nové na další řádek.

Literatura

[1] RFC 2373: IP Version 6 Addressing Architecture, http://tools.ietf.org/html/rfc2373

[2] Guidelines for 64-bit Global Identifier (EUI-64™), http://standards.ieee.org/develop/regauth/tut/eui64.pdf

[3] RFC 5342: IANA Considerations and IETF Protocol Usage for IEEE 802 Parameters, http://tools.ietf.org/html/rf

[4] RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specif
http://tools.ietf.org/html/rfc4443

[5] RFC 5952: A Recommendation for IPv6 Address Text Representation, http://tools.ietf.org/html/rfc5952

[6] Organizationally unique identifier, en.wikipedia.org/wiki/Organizationally_unique_identifier, modifikováno: 30.3.2012

[7] SATRAPA, Pavel. *IPv6: internetový protokol verze 6.*, 3. vydání, Praha: CZ.NIC, 2011, 407 s. CZ.NIC. ISBN 978-80-904248-4-5

[8] draft-gont-opsec-ipv6-host-scanning-01: Network Reconnaissance in IPv6 Networks, http://tools.ietf.org/html/
gont-opsec-ipv6-host-scanning-01