VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ - FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IPK – projekt č. 2 - [ZETA] Packet sniffer

xdanca01

2019/2020

Obsah

Úvod

Cílem projektu bylo vytvoření programu, který dokáže zachytit na zvoleném síťovém rozhraní packety, vyfiltrovat je a vypsat jejich obsah. Já jsem se zaměřil na podporu síťového rozhraní Ethernet. V zadání nebylo specifikováno, jaké protokoly má sniffer podporovat kromě TCP/UDP, proto jsem se zaměřil hlavně na funkčnost těchto dvou a to jak pro IPv4, tak i pro IPv6, u které si nejsem jistý správnou funkčností z důvodu testování.

Návrh

Při návrhu snifferu jsem se přiklonil k doporučením a program implementovat s využitím knihovny libpcap, která sama o sobě dokáže packety zachytávat jen se musí správně nastavit filtr a na jakém rozhraní se mají packety zachytávat (filtry a rozhraní), a knihovny netinet, ze které jsem využil některých struktur pro filtraci dat packetů.

Popis implementace

Na začátku programu ve funkci main se nastaví maximální délka packetu snaplen, místo pro zápis chybového hlášení, který se předá funkcím z knihovny libpcap, počet packetů k zachycení, u kterého je defaultní hodnota 1 a nastaví se pole s možnostmi pro funkci getopt_long. Po deklaraci a definici proměnných, se cyklicky volá funkce getopt_long, které se předá počet argumentů, argumenty a možnosti pro argumenty, a podle které se nastavuje prostředí. Pokud při nastavení prostředí se nenastaví rozhraní pro odposlech, tak se na stdout vypíše seznam dostupných rozhraní.

Proces získávání packetů začíná u funkce pcap_open_live, která slouží pro získání handlu k packetům na rozhraní. Po návratu z funkce se kontroluje, jestli byl handle předán, pokud handle předán nebyl, tak se vypíše error z předaného bufferu na error. Po správném předání handle se vytvoří filter s dynamicky alokovaným místem v paměti, který se předá funkci pcap_compile. Tato funkce přepíše filter na filter program, který se aplikuje pomocí funkce pcap_setfilter na získaný handle.

Samotné zpracování packetů začíná od funkce pcap_loop, které se předá počet packetů pro zachycení a funkce callback_for_packet, které se má packet s dalšími daty předat. Po předání packetu funkci se vyfiltruje ethernetová hlavička strukturou ether_header, která je potřeba pro rozlišení typu internetového protokolu. Na základě typu internetového protokolu se získávají data z packetů. Přesněji u IPv4 se nejdříve získá velikost hlavičky, která je na ethernetovém rozhraní posunutá vždy o 14 bajtů (14 + 0) a zároveň je zapsána ve spodní půlce bajtu, proto provedu operaci AND s hodnotou 15, která má ve 4 spodních bajtech same jedničky. Poté co získám délku IP headeru, tak můžu získat zdrojový a cílový port. Zdrojový port se nachází hned za koncem hlavičky pro IP, tudíž je na pozici 14 (délka ethernet headeru) + velikost hlavičky IP, a cílový port se vyskytuje 2 bajty za zdrojovým. Každý z portů se skládá ze 2 bajtů, které si převedu na správný tvar pomocí funkce ntohs. Po získání portů si nadefinuju struktury in_addr pro získání dat na zjištění IP adres. Zdrojová IP adresa začíná vždy na 26. bajtu, tedy pokud se před IP protokolem nachází ETHERNET header, a cílová adresa začíná vždy za stejných podmínek na 30. bajtu. Získané bajty si pomocí funkce inet_ntoa převedu do správného zápisu IP adres, z kterých se pokusím získat hostname funkcí getnameinfo. Podle výsledku funkcí se vypíše buď ip adresa, nebo hostname. Po výpisu informací hlavičky packetu se začnou zpracovávat a vypisovat všechna data packetu.

Data packetu se postupně vypisují v hexa tvaru a zároveň se každá hodnota bajtu přepíše pokud možno na tisknutelný znak do alokovaného pole, které se vypíše po 16 bajtech a nebo na konci dat packetu.

Testování

Testování proběhlo odchytáváním packetů na různých rozhraních, kde jsem výstup porovnával s daty ve wiresharku. Při testování jsem zkoušel i IPv6, ale jen pomocí ping -6 localhost a to se mi zobrazil pouze protokol ICMPv6.