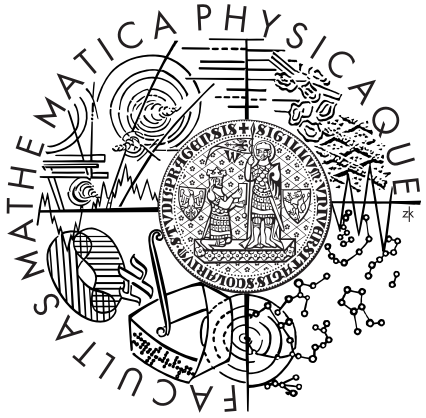
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Martin Nepivoda

Analýza možností zapouzdření síťových protokolů do aplikačních a metod detekce na firewallech

Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Jiří Kosina

Studijní program: Informatika, Softwarové systémy

Rád bych poděkoval vedoucímu své práce Mgr. Jiřímu Kosinovi za inspiraci, názory a rady při tvorbě této práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 4. srpna 2008

Martin Nepivoda

Obsah

Název práce: Analýza možností zapouzdření síťových protokolů do aplikačních a metod detekce na firewallech

Autor: Martin Nepivoda

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Jiří Kosina

E-mail vedoucího: jikos@jikos.cz

Abstrakt: Cílem diplomové práce je podrobně analyzovat možné způsoby zapouzdření protokolů síťové vrstvy do protokolů vrstvy aplikační – například za účelem vytvoření síťového „tunelu“, obcházejícího pravidla síťových firewallů. Výsledkem práce jsou na základě vzniklé analýzy navržené způsoby detekce tohoto zapouzdřování na síťových firewallech, s cílem zamezit vytváření síťových tunelů skrz aplikační protokoly. Součástí práce je referenční implementace některého z navržených řešení.

Klíčová slova: počítačové sítě, bezpečnost, tunely

Title: Analysis of possibilities of encapsulation of network protocols into application protocols and methods of detection on firewalls

Author: Martin Nepivoda

Department: Department of Software Engineering

Supervisor: Mgr. Jiří Kosina

Supervisor's e-mail address: jikos@jikos.cz

Abstract: The goal of this work is to analyze possible ways of network protocol into application protocol encapsulation – as used for example to create network „tunnel“ or bypass firewall rules. Methods to detect such encapsulation on network firewalls aimed to prevent such application protocol tunnels are proposed. The work includes reference implementation of one proposed method.

Keywords: computer networks, security, tunnels

Kapitola 1

Úvod

Bezpečnost je důležitým požadavkem na počítačovou síť a její zajištění bývá základní náplní práce správců. Obvyklým nástrojem k tomu jsou firewally, které slouží k aplikaci stanovených pravidel provozu sítě i připojených klientských počítačů. Při praktické realizaci těchto pravidel však bývají opomíjeny některé možnosti, které mohou uživatelé sítě použít k jejich obcházení – ať už z důvodu náročnosti nasazení nezbytných opatření (třeba u připravených síťových řešení), nedostupnosti vhodných nástrojů nebo prosté nevědomosti správců.

Cílem této práce je upozornit na některé způsoby obcházení síťových pravidel na firewallech, které využívají povolené služby, nabídnout možnosti jejich detekce a postupy, které mohou tomuto nežádoucímu obcházení zabránit nebo ho omezit.

První část popisuje některé v současnosti používané způsoby zneužívání služeb včetně použitých programů a jejich chování. Druhá se zaměřuje na dva základní přístupy k detekci takové činnosti a navrhuje možné způsoby jejich použití u jednotlivých služeb a nasazení v boji s nežádoucím chováním. Následuje popis praktické implementace jednoho z navržených opatření a závěrečné shrnutí celé práce.

1.1 Základní informace

Transportní protokoly, jak je patrné z jejich názvu, slouží pro přenos obecných dat s různými vlastnostmi – spolehlivost, jedinečnost, dodržování pořadí atp. – a jsou pro tuto činnost navrženy a optimalizovány. Nejznámějším

z nich jsou s Internet Protocolem (IP) spojené Transmission Control Protocol (TCP) a User Datagram Procotol (UDP) [14].

Aplikační protokoly pak slouží pro potřeby komunikace konkrétní aplikace nebo skupiny aplikací a již dávají datům konkrétní význam. Pro přenos těchto dat pak používají právě transportní protokoly. Hierarchie vrstev protokolů, jak ji definuje TCP/IP, je shrnuta v obrázku 1.1 [12].

Aplikační vrstva
Transportní vrstva
Internetová („síťová“) vrstva
Linková vrstva

Obrázek 1.1: Hierarchie vrstev TCP/IP

Tato hierarchie určuje, jakou nižší vrstvu ten který protokol používá a je do ní tedy zapouzdřen. Je však možné tuto jednoduchou posloupnost porušit a zapouzdřit transportní protokol znovu do protokolu aplikačního, který je pak opět postoupen nižším vrstvám – vznikají tedy cykly.

Tento na první pohled nesmyslný postup, který komplikuje proces přenosu dat, přidává režii dalších protokolů a často využívá aplikační protokoly s nevhodnými vlastnostmi, může mít několik důvodů:

1. obcházení pravidel sítě
2. soukromí (např. sledování spojení na proxy normálně poskytne seznam navštívených stránek, ale u tunelu bez hlubší analýzy přenášených dat nikoliv)
3. přenos nepodporovaného transportního či síťového protokolu

První důvod je z pohledu správce dané sítě jasně nežádoucí a dodržování nastavených pravidel používání sítě je také hlavním cílem této práce. Další dva body ale mohou být zcela legitimní a někdy až nezbytné. Například ochrana soukromí by měla být pro uživatele samozřejmostí. A fungování nepodporovaných protokolů na Internetu může být nutné při provozování existujících aplikací.

Avšak „soukromí“ mohou vyhledávat také nežádoucí programy, instalované na klientských počítačích – spyware, backdoors apod. Ty slouží ke sledování uživatele nebo zneužití jeho počítače, třeba pro odesílání spamů – a to nejen e-mailových, ale i do webových diskusních fór – či jako odrazového můstku

pro útoky na další počítače v dané síti i mimo ni. Takovéto programy mohou poskytovat též služby vzdáleného přístupu na daný stroj, aby ho umožnily útočníkovi lépe ovládat. V těchto případech se autor daného programu snaží skrýt skutečnost, že k nějaké komunikaci dochází a že je pomocí ní přenášen nežádoucí obsah.

Běžným případem tunelování problematického protokolu je podpora pouze IPX u starších počítačových her – pro jejich funkčnost s TCP/IP a Internetem je nutné vytvoření virtuální sítě. K tomu slouží například populární program Hamachi [3]. Virtuální sítě (obvykle nazývané Virtual Private Networks, VPN) se zcela běžně používají například pro bezpečné připojení do firemních sítí z vnějších částí Internetu. U nich jde však obvykle o zapouzdření pomocí speciálního, k tomu navrženého protokolu, standardně přenášeného pomocí TCP či UDP, a jejich popis či analýza nejsou cílem této práce.

Tunelovací programy nabízí tři základní způsoby komunikace:

1. Tunelování jediného spojení – v tomto případě umožňuje tunel přenos jediného spojení na vybraný cílový stroj. Dá se využít buď pro jednu konkrétní službu, kterou uživatel požaduje (například instant messaging), nebo rozšířit na obecnější možnosti třeba použitím SSH[1].

2. SOCKS proxy – tunelovací program vytvoří na klientském počítači proxy, která umožňuje vhodně upraveným programům komunikovat pomocí TCP a UDP obvyklým způsobem [15]. Proxy se stará o doručení dat na požadovanou adresu, umožňuje komunikaci s více stroji atp. SOCKS proxy sice vyžaduje, aby byl daný program pro komunikaci pomocí ní naprogramován, ale lze využít speciálních nástrojů, které umožňují využívat proxy i neupraveným programům – například FreeCap [2]. Jednou z možností, jak takovou změnu zajistit, je zachycení volání systémových síťových služeb a jejich převod na nastavenou proxy.

3. Virtuální síťové rozhraní – nejpohodlnější situace, kdy je tunel reprezentován síťovým rozhraním, se kterým lze pracovat standardními systémovými nástroji. V tomto případě není nutné nijak nastavovat nebo upravovat používané programy a tunel podporuje přenos všech protokolů. Možnou nevýhodou je nutnost po spuštění tunelu správně nastavit síť (především routování), ačkoliv to už obvykle pokročilejší tunely zvládnou samy, a použití administrátorských práv.

První dva popsané způsoby mohou využívat i uživatelé bez vyšších práv.

[1] Secure SHell

Kapitola 2

Metody tunelování

V této kapitole jsou popsány metody nejčastěji využívané pro tunelování transportních a síťových protokolů pomocí aplikačních. U každé z nich je základní popis situace, ve které bývá daná metoda využívána, a je vysvětlen princip metody tunelování včetně zhodnocení jejích vlastností, výhod a nevýhod. Jsou též podrobněji popsány konkrétní aplikace, které se pro vytváření a správu takového tunelu používají.

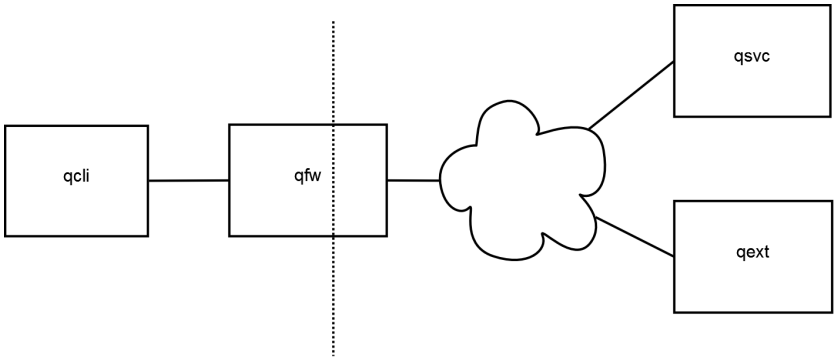
Všechny popsané metody byly prakticky vyzkoušeny. Analýza protokolů jednotlivých tunelovacích metod byla prováděna ze zachycené komunikace a ze zdrojových kódů, pokud byly k dispozici. Pro zjednodušení celého procesu a lepší kontrolu nad ním byla využita virtuální síť několika emulovaných počítačů.

Cílem bylo obvykle zapojení s klientským počítačem uvnitř sítě, ze kterého se uživatel pokouší o komunikaci, firewallem/routerem vnitřní sítě, cílovým serverem, kterého se uživatel snaží dosáhnout, a volitelně další uživatelův počítač vně omezené sítě. Schéma zapojení by tedy mělo vypadat jako na obrázku 2.1.

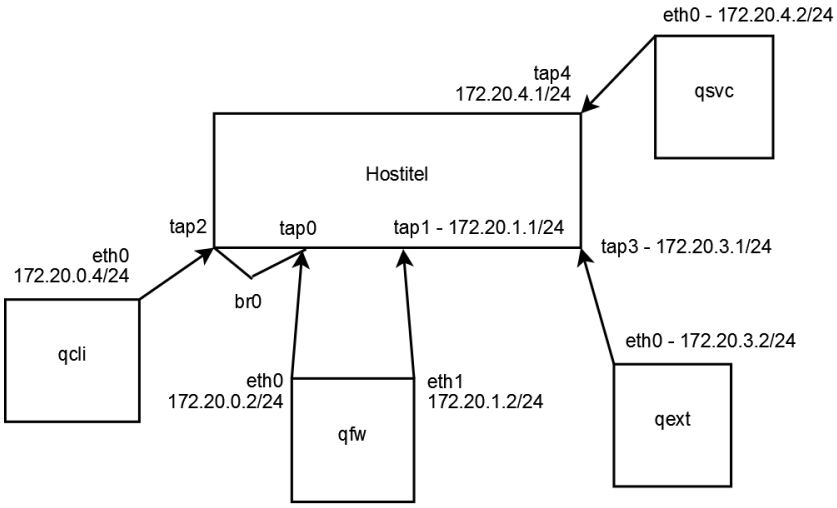
Klientský počítač `qcli` uvnitř sítě a další vně (`qext`) má pod plnou kontrolou uživatel, který se snaží o použití tunelu. Firewall/router `qfw` omezené sítě naopak ovládá cizí správce, který chce tunelování zabránit. Server s cílovou službou `qsvc` spravuje cizí, třetí osoba, se kterou uživatel ani správce nekomunikují a nemohou ji ovlivnit.

Toto logické zapojení bylo prakticky realizováno podle obrázku 2.2.

Virtuální stroje běžely emulovaně na hostitelském počítači, který také zajišťoval routování mezi jednotlivými sítěmi. Každý virtuální stroj má alespoň



Obrázek 2.1: Logické schéma zapojení



Obrázek 2.2: Fyzické schéma zapojení

jedno síťové rozhraní, které na jeho straně vystupuje jako běžná ethernetová síťová karta a u hostitele jako zařízení `tap`. Omezená, vnitřní síť byla realizována síťovým mostem `br0` mezi vnějšími konci síťových rozhraní virtuálního klientského počítače a routeru/firewallu této sítě (`tap0` a `tap2`). Na straně hostitele tedy nemají jejich síťová rozhraní přiřazenu IP adresu a pakety mezi nimi nejsou routovány klasickým způsobem jako mezi zbylými virtuálními stroji, ale předávány mostem na linkové vrstvě [6]. Provoz na virtuální síti byl sledován programem Wireshark [8].

Popis a konfigurace hostitelského počítače

Pentium-M 1,1 GHz s 1 GB RAM

OS GNU/Linux Debian Sid, jádro 2.6.25

uml-utilities 20070815-1.1, bridge-utils 1.4-4

Popis a konfigurace virtuálních počítačů

QEMU 0.9.1, kqemu 1.3.0pre11

CPU řady i686, 64 MB RAM

OS GNU/Linux Debian 4.0 Etch, jádro 2.6.18

2.1 HTTP

Situace

Nejjednodušší a obvyklou situací je HTTP proxy [11]. Přístup mimo síť je zakázán na routeru/firewallu a jedinou poskytovanou službou je právě HTTP proxy. Základní důvody pro toto nastavení jsou dva:

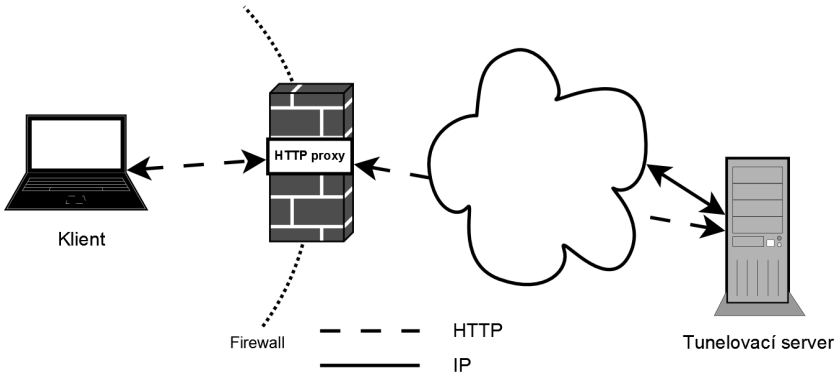
- Firemní síť – Z důvodu bezpečnosti je jedinou povolenou službou Internetu WWW (případně e-mail na lokálním serveru). Pro zjišťování informací potřebných pro práci obvykle web stačí a snižuje se riziko provozování jiných programů, které pak už často nebývají kontrolovány přímo na jednotlivých uživatelských stanicích.
- Veřejné/placené připojení – I v současnosti je někdy přístup k webu a e-mailu zaměňován za plnohodnotné připojení k Internetu, případně je toto nabízeno jako levnější varianta. Často se to také děje u veřejných přípojek (např. hot-spoty v hotelech).

Web je dnes – společně s e-mailem, který je však také často zpřístupňován přes webové rozhraní – nejmasověji používanou službou Internetu. Pro některé uživatele dokonce tyto dva pojmy splývají. Postupně se na web přesouvají i další služby, jako jsou instant messaging, práce s dokumenty a jejich sdílení atd. Proto je přístup k webu základní službou, jejíž funkčnost je v běžných sítích prakticky nezbytná a jejich správci podporovaná.

Rozšířenost a dostupnost webu je proto velmi zajímavá i pro tvůrce aplikačních tunelů, které umožňují provoz nepřizpůsobených aplikací nebo přímo

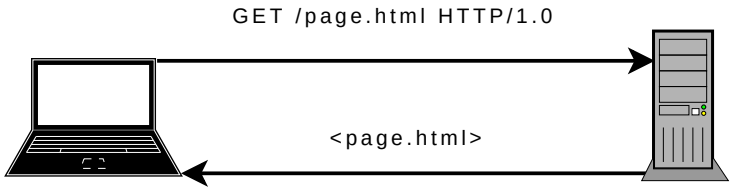
nahrazují síťové rozhraní a využívají protokol HTTP. Důležitým požadavkem na tunely také bývá schopnost pracovat s připojením přes HTTP proxy.

HTTP tunel přenáší data mezi klientem uvnitř omezené sítě a serverem mimo ni pomocí vlastního protokolu, zapouzdřeného v HTTP tak, aby prošel přes proxy. Tento server pak posílá tunelovaná data již klasicky zpět do Internetu – celý proces ilustruje obrázek 2.3. Je tedy nutné vlastnit (nebo mít možnost využívat) z Internetu dostupný server, přes který je všechen provoz směrován.



Obrázek 2.3: Obecné schéma zapojení HTTP tunelu

HTTP je pro obecnou obousměrnou komunikaci nevhodný, neboť pracuje stylem dotaz klienta – odpověď serveru, viz obrázek 2.4, takže server nemůže sám od sebe poslat klientovi data, kdy chce.



Obrázek 2.4: Zjednodušené schéma HTTP komunikace

Pro příjem dat jsou tedy nutné opakované dotazy klienta na vzdálený server, tzv. polling. Od HTTP verze 1.1 naštěstí není nutné pro každý dotaz vytvářet vlastní TCP spojení, ale je možné používat spojení persistentní, které je udržováno pro několik dotazů a jejich odpovědí. To snižuje vytížení serveru i HTTP proxy. Persistence spojení klienta s proxy a proxy s cílovým serverem se nemusejí shodovat.

Programy

HTTP Tunnel

Domovská stránka: `<http://the-linux-academy.co.uk/downloads.htm>`

Testované verze: 0.9.4 a 1.1.1

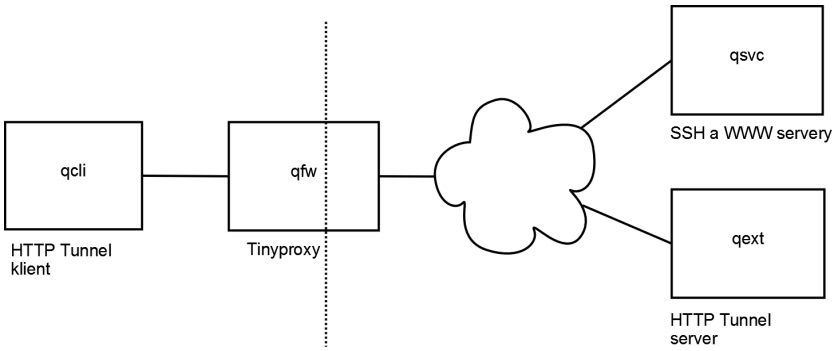
HTTP Tunnel je open source program s GPL licencí, který nabízí několik možností tunelování spojení přes HTTP proxy:

- možnost přímého tunelování jednoho portu u klienta na zadanou adresu v Internetu, včetně UDP
- provoz SOCKS proxy poslouchající na vybraném portu

Dále umožňuje šifrování spojení tunelových klientů se serverem pomocí SSL a autentizaci klientů.

Program se snaží používat vlastní HTTP hlavičky (HT-Tid, HT-Seq apod.) a při sestavování tunelu testuje, zda použitá proxy přenos těchto hlaviček na server podporuje. Jejich obsah je v prvním dotazu uložen také jako součást dotazu metody GET, aby byl zajištěn jejich přenos v každém případě. Pokud proxy tyto doplňkové hlavičky cílovému serveru neposílá, jsou v dalších dotazech přenášeny už pouze jako součást dotazu.

Pro analýzu bylo použito následující zapojení virtuální sítě s jednotlivými službami:



Obrázek 2.5: Zapojení sítě pro testování programu HTTP Tunnel

Žádost o sestavení tunelu ukazuje výpis 1. Obsahuje mj. speciální vlastní hlavičky s identifikací tunelu (HT-Tid), číslem dotazu (HT-Seq) nebo cílovým bodem tunelovaného spojení (HT-Dst).

```
Internet Protocol, Src: 172.20.1.2, Dst: 172.20.4.2
Hypertext Transfer Protocol
  GET /a.htm?HT-Tid%3A%201%3A1865%3Aqemu2c8f4%3A%3A1234
      %3A0%0D%0AHT-Seq%3A%201%0D%0AHT-Ack%3A%201%0D%0AHT
      -Cst%3A%200%0D%0AHT-Tst%3A%200%0D%0AHT-Des%3A
      %20172.20.3.2%3A22%0D%0AHT-Cvr%3A%201.1.1%20LINUX
      %0D%0AHT-Dat%3A%20%22%22%0D%0A...
  Host: 172.20.4.2
  Connection: close
  Via: 1.1 tinyproxy (tinyproxy/1.6.3)
  Cache-Control: no-cache
  HT-Cvr: 1.1.1 LINUX
  HT-Tst: 0
  HT-Dln: 0
  HT-Tid: 1:1865:qemu2c8f4::1234:0
  HT-Ack: 1
  HT-Seq: 1
  HT-Dat: ""
  HT-Des: 172.20.3.2:22
  HT-Cst: 0
  HT-Spl: 0
```

Výpis 1: Žádost klienta o sestavení tunelu

Jak je vidět, proxy předala serveru i zmiňované speciální hlavičky tunelovacího klienta (HT-*). Následuje odpověď serveru ve výpisu 2.

```
Internet Protocol, Src: 172.20.4.2, Dst: 172.20.1.2
Hypertext Transfer Protocol
  HTTP/1.1 200 OK
  Cache-Control: no-cache
  Connection: keep-alive
  Content-Length: 0
  HT-Tid: 1:1865:qemu2c8f4::1234:0
  HT-Seq: 1
  HT-Ack: 2
  HT-Cst: 1
  HT-Quo: 0,0
```

Výpis 2: Odpověď serveru

Klient také pravidelně odesílá na server prázdný požadavek, aby zjistil, zda tam pro něj nečekají nějaká data.

Existuje též komerční služba založená na programu HTTP Tunnel, nazvaná Firedrill [1]. Nabízí klienty pro Linux, Unixy, MacOS X a MS Windows, ale především možnost využít placených serverů provozovatele služby. Tím pro uživatele odpadá nutnost instalovat a spravovat vlastní servery, a použití HTTP tunelu se tedy zjednodušuje pouze na instalaci klienta. Propagované využití programu je pro přístup k Internetu z omezených sítí zákazníků.

HTTP-Tunnel

Domovská stránka: `<http://www.http-tunnel.com/>`

Testovaná verze: 4.4.4000

Tento komerční program je určen pro systémy Windows. Umožňuje vytvoření lokálního SOCKS proxy serveru nebo přesměrování jednotlivých místních portů na vybrané cíle. V základní verzi je zdarma, kdy program používá vybrané pomalejší servery provozovatele. Za poplatek zhruba čtyři dolary měsíčně je možné využít speciální servery, které zaručí lepší odezvu i vyšší datové toky.

Při spuštění si program dotazem na hlavní server zjistí adresu přiděleného tunelovacího serveru a svůj identifikátor. Pak umožní zadání registračního klíče nebo provoz v bezplatném režimu – ten byl také testován. Další komunikace už probíhá s přiděleným serverem. Pro komunikaci používá především metodu POST a sadu serverových stránek (Update.htm, Data.htm), se kterými komunikuje vlastním binárním protokolem. Pro tunelované spojení program používá dotaz na stránku Data.htm s fiktivní velkou délkou odpovědi a možností žádat jeji konkrétní úsek (hlavička Accept-Ranges[1]).

Výhodou programu je podpora OS Windows, jednoduchá instalace a nastavení – např. možnost dynamicky měnit seznam přesměrovaných portů – a možnost použít i v neplacené verzi tunelovací servery provozovatele, takže uživatelům odpadá nutnost mít vlastní stroj s instalovaným tunelem. To umožňuje použití HTTP tunelování prakticky komukoliv i bez hlubších znalostí této problematiky.

[1] [11], kapitola 14.5

2.2 HTTPS

Situace

Obdobná jako v předchozím případě s HTTP tunelem, obzvlášť u firemních sítí.

Při použití HTTP proxy z předchozí kapitoly klienti posílali této proxy požadavky, na jejichž základě pak proxy žádala o data cílové servery. V případě HTTPS je situace složitější, protože komunikace klienta s cílovým serverem je šifrovaná. Šifrování u HTTPS, které končí na vzdáleném serveru, může z pohledu klienta za proxy začínat na dvou místech:

1. Na proxy – Komunikace klienta s místní HTTPS proxy je nešifrovaná nebo šifrovaná odděleně. Proxy tak vidí obsah celé komunikace a pracuje podobně jako klasická HTTP proxy.
2. Přímo u klienta – HTTPS proxy nerozumí obsahu šifrované komunikace mezi klientem a serverem, pouze umožňuje jejich spojení.

HTTPS proxy obvykle využívají druhého přístupu, který je z hlediska klienta bezpečnější, a nabízí proto metodu CONNECT [16]. Ta umožňuje vytvořit TCP spojení klienta s určeným portem na cílovém serveru bez dalších zásahů do komunikace nebo schopnosti proxy zjistit obsah přenášených dat. Cílový port bývá někdy omezen pouze na standardní 443, kde je obvykle služba HTTPS provozována [13].

```
CONNECT services.addons.mozilla.org:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; cs
    ; rv:1.9) Gecko/2008052906 Firefox/3.0
Proxy-Connection: keep-alive
Host: services.addons.mozilla.org
```

Výpis 3: Sestavení spojení pomocí příkazu `CONNECT`

To značně zjednodušuje tunelování, neboť je možné použít libovolný vlastní protokol a není třeba se striktně držet standardu HTTP, kdy navíc může proxy z předchozího případu komunikaci upravovat. Pro tunelování jediného spojení, například SSH, také není potřeba server ve vnější sítí, ale je možné se připojit přímo k cílové službě. V obecném případě na vnějším serveru běží služba, která pro klienta zprostředkovává veškerou komunikaci s ostatními stroji.

Programy

HTTP-Tunnel

Tento program byl uveden již v předchozí kapitole. Kromě tunelování přes HTTP umožňuje využít také HTTPS proxy a je to i autory doporučovaná metoda.

ProxyTunnel

Domovská stránka: `<http://proxytunnel.sourceforge.net/>`

Testovaná verze: 1.9.0

Program pouze zprostředkovává spojení pomocí příkazu `CONNECT` na proxy, jinak je zcela transparentní. Využití najde například v kombinaci se SSH, kdy je možné jednoduše nastavit průchod přes proxy pro konkrétní servery. V konfiguračním souboru `~/.ssh/hosts` se spojení s takovým serverem nastaví následujícím záznamem:

```
Host stroj
  ProtocolKeepAlives 30
  ProxyCommand /path/to/proxytunnel -p proxy.customer.
     com:8080 -u user -s password -d mybox.atho.me:443
```

Pak lze použít `ssh stroj` jako u připojení bez proxy a ProxyTunnel automaticky zajistí vytvoření spojení přes lokální HTTPS proxy příkazem `CONNECT`.

2.3 ICMP

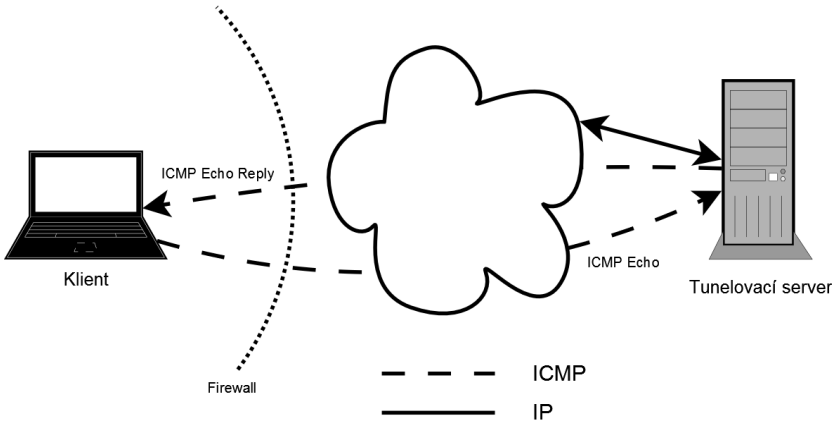
Dalším protokolem, který je možné použít pro tunelování transportních protokolů, je Internet Control Message Protocol (ICMP) [19]. Tento protokol je součástí Internet Protocol (IP) a je nezbytný pro jeho správné fungování – proto také bývá často povolený a použitelný [14]. Tunelování přes ICMP se může hodit mj. v případě, že jsou dostupné aplikační protokoly omezeny nebo sledovány.

Využívá se zpráv typu Echo (8) a Echo Reply (0), které obsahují volitelnou datovou část. Ta je použita k přenosu uživatelských dat. Struktura takových paketů je na obrázku 2.6 – paket Echo Reply vypadá stejně, pouze pole Type obsahuje hodnotu 0.

Bity	0–7	8–15	16–23	24–31
0	hlavičky IP (mj. zdrojová a cílová adresa)			
160	Type = 8	Code = 0	Checksum	
192	ID		Sequence	
224+	Data			

Obrázek 2.6: Struktura ICMP paketu Echo Request

Tunel se skládá ze dvou částí: klientské uvnitř sítě a serverové vně. Klientský program uloží přenášená data do datové části ICMP paketu a odešle jej na zvolený stroj ve vnější síti. Zde běží serverová služba, která zpracovává přijaté ICMP pakety a posílá zpět podobně upravené odpovědi s daty pro klienta. Schéma zapojení demonstruje obrázek 2.7.



Obrázek 2.7: Obecné schéma zapojení ICMP tunelu

Běžně nastavené stavové firewally sledují počet odeslaných a přijatých ICMP zpráv, takže jich pak není možné jednoduše posílat z tunelovacího serveru libovolné množství – v takovém případě je nutné použít polling jako u HTTP. I přesto je tunel přes ICMP výrazně jednodušší a má velmi malou režii přenosu dat.

Programy

ICMPTX

Domovská stránka: `<http://thomer.com/icmptx/>`

Testovaná verze: 0.01

ICMPTX je open source program s licencí GPL. Pracuje v režimu klient/server způsobem popsaným v předchozí obecné části. Na obou stranách vytváří nové, virtuální rozhraní `tun` a IP pakety pro něj balí přímo do datové části ICMP zpráv Echo/Echo reply. ICMP zprávy mají nastavený předem zvolený identifikátor, podle kterého jsou tunelovací pakety rozlišovány. Jak je vidět ze zachycené zprávy ve výpisu 4 a ze zdrojového kódu, sekvenční číslo ICMP paketů se nemění a je stále nula.

```
Internet Protocol, Src: 172.20.0.4, Dst: 172.20.3.2
Version: 4
Header length: 20 bytes
Total Length: 91
Identification: 0x0000
Flags: 0x04 (Don't Fragment)
Fragment offset: 0
Protocol: ICMP (0x01)
Header checksum: 0xdf73 [correct]
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x492c [correct]
  Identifier: 0x6a1d
  Sequence number: 0
  Data (63 bytes)
```

Výpis 4: Tunelovací ICMP zpráva programu ICMPTX

Při spouštění serverové části se určí adresa z nepoužívaného rozsahu, která je pak nastavena tomuto rozhraní. Klientská část dostane jako parametr skutečnou adresu stroje, kde běží server. Vytvořenému rozhraní se nastaví adresa ze stejné sítě, jako má nastavenou serverová část.

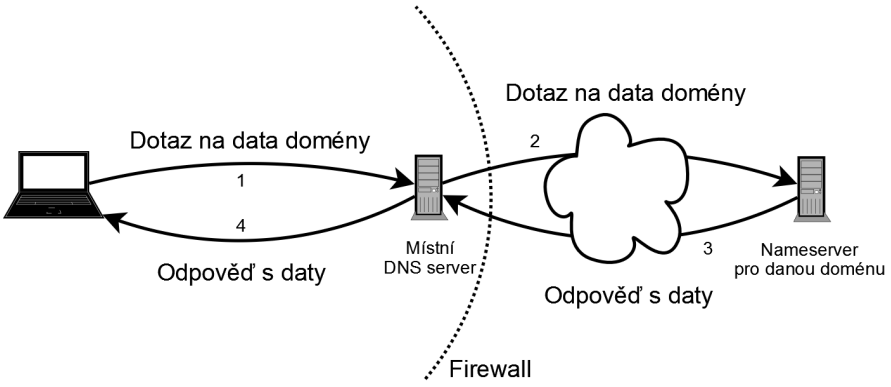
U klienta je pak třeba pro směrování veškerého provozu změnit výchozí bránu na adresu serverové části tunelu s tím, že je nutné vytvořit specifický záznam pro skutečnou adresu tunelového serveru, který směruje provoz na něj přes původní bránu. To zajistí, aby ICMP pakety tunelovaného spojení putovaly správně pomocí spojení skutečného.

Na serveru je potřeba dostat příchozí spojení z tunelového rozhraní do Internetu – nejjednodušší je použití NAT/masquerade tunelového rozhraní za skutečným.

2.4 DNS

Situace

Domain Name Service (DNS) je hierarchický distribuovaný systém pro přiřazení hodnot jménům [17, 18]. Tyto hodnoty (záznamy) mají různé významy a dělí se do několika typů[2] – nejběžnějšími jsou IP adresa (záznam typu A, resp. AAAA) či poštovní server pro danou doménu (typ MX), ale je možné ukládat i další informace, včetně obecného textového (typ TXT) a binárního (typ NULL) záznamu. Klientský počítač se pro získání údajů z DNS obrací na svůj nadřazený DNS server, který pro něj požadovanou informaci získá (obvykle řetězem dotazů na další servery) a předá mu odpověď. Proto klientovi pro používání DNS stačí přístup pouze k místnímu serveru za předpokladu, že ten už má přístup dál. Postup zodpovězení dotazu a putování dat popisuje obrázek 2.8.



Obrázek 2.8: Ukázka putování dat při DNS dotazu

V některých případech sítí, které vyžadují pro přístup mimo lokální síť registraci (obvykle placené hot-spoty na letištích, v hotelích apod.), je v rámci služeb místní sítě dostupný DNS server a přístup ven omezuje firewall na routeru svázaný s registračním/platebním systémem. Ten spojení neregistrovaných uživatelů mimo síť přesměrovává na platební (registrační) portál. Po registraci či zaplacení je uživateli na firewallu povolena komunikace se zbytkem Internetu. Z důvodu jednoduchosti nastavení není DNS server omezen na překlad adres v závislosti na stavu registrace konkrétních uživatelů, takže může uživatel žádat např. o IP adresy vnějších strojů, i když k nim pak nemá přímý přístup.

[2][17], kapitola 3.6. Resource Records

Toho lze využít ke komunikaci s vnějším serverem, na který je delegována vybraná doména. Klientský tunel odesílá data na tento server ve formě dotazů na adresy v této doméně. Tyto dotazy směřují na lokální DNS server, který zařídí jejich zodpovězení, a klient tedy vůbec nekomunikuje se svým serverem v Internetu přímo. Dotazy na adresy zpracovává serverová část tunelu vystupující jako DNS server, která zpět posílá svá data v odpovědích.

DNS je opět založen na modelu komunikace žádost-odpověď, takže pro přenos dat ze serveru je nutné využít polling. Odezva bývá delší než například u ICMP tunelu a datový tok naopak menší. Praktická využitelnost DNS tunelu však nad těmito nevýhodami převažuje, a proto je často používán.

Programy

NSTX

Domovská stránka: `<http://thomer.com/howtos/nstx.html>`

Testovaná verze: 1.1-beta6

NSTX je open source program s licencí GPL od téhož autora jako ICMPTX. Také emuluje další rozhraní a nastavení routování a NAT/masquerade je tedy u obou shodné. Funguje systémem klient/server tak, jak byl popsán v obecné části.

Klient posílá svá data jako dotazy na TXT záznamy pro domény, jejichž jméno zkonstruuje převodem binárních dat na sérii písmen a číslic (vždy čtyři znaky ze tří bytů) [18]. Příklad takového dotazu je ve výpisu 5.

Server pak odpoví s daty uloženými v binární podobě s vlastní hlavičkou v odpovědi záznamu TXT, jak ukazuje výpis 6.

```
Internet Protocol, Src: 172.20.0.4, Dst: 172.20.0.2
User Datagram Protocol, Src Port: 2048, Dst Port:
   domain (53)
Domain Name System (query)
Transaction ID: 0xed54
Flags: 0x0100 (Standard query)
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
  Name: cTabvhuuaadWtr-aaqaytCWOaaaikaaabeS3e-
     OkECLmaaaaaOaiw-k9eaaacba.w-
     baiicGajyrqaaaaaaqmdaqaa.moje.do
  Type: TXT (Text strings)
  Class: IN (0x0001)
```

Výpis 5: Tunelovací DNS dotaz

Odpovědi mají nastavenu dobu platnosti (Time To Live) na nulu, aby se servery po cestě zbytečně nezatěžovaly ukládáním těchto dat a případně tím i později nevracely nesprávné údaje.

Protože je protokol DNS iniciovaný pouze klientem, je možné přenášet data od serveru jen jako odpověď na dotaz. Klient proto posílá pravidelně „prázdné“ dotazy, aby umožnil serveru poslat čekající data (již zmiňovaný polling).

```
Internet Protocol, Src: 172.20.0.2, Dst: 172.20.0.4
User Datagram Protocol, Src Port: domain (53), Dst Port
    : 2048
Domain Name System (response)
Transaction ID: 0xed54
Flags: 0x8180 (Standard query response, No error)
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
Queries
  Name: cTabvhuuaadWtr-aaqaytCWOaaaikaaabeS3e-
      OkECLmaaaaaOaiw-k9eaaacba.w-
      baiicGajyrqaaaaaaqmdaqaa.moje.do
  Type: TXT (Text strings)
  Class: IN (0x0001)
Answers
  Name: cTabvhuuaadWtr-aaqaytCWOaaaikaaabeS3e-
      OkECLmaaaaaOaiw-k9eaaacba.w-
      baiicGajyrqaaaaaaqmdaqaa.moje.do
  Type: TXT (Text strings)
  Class: IN (0x0001)
  Time to live: 0 time
  Data length: 58
```

Výpis 6: Odpověd tunelovacího serveru

Iodine

Domovská stránka: `<http://code.kryo.se/iodine/>`

Testovaná verze: 0.4.1-4

Iodine je novější program pro tunelování IP přes DNS. Přístup k tunelu je chráněn heslem, bylo zjednodušeno nastavení a zlepšen způsob přenosu dat. Základ fungování je obdobný jako u programu NSTX, včetně vytváření tunelovacího rozhraní. Iodine však používá záznamy typu NULL, využívá rozšíření EDNS0, které umožňuje například použít pro komunikaci větší UDP pakety, a data přenášená pomocí DNS komprimuje [20].

Serverová část si udržuje seznam připojených klientů a zajišťuje jejich automatickou konfiguraci při připojení – IP adresu klientského tunelovacího

rozhraní a volitelně MTU[3]. Klienti se k serveru přihlašují pomocí hesla. Server data mezi jednotlivými klienty předává přímo, tj. nevyužívá v takovém případě routování hostitelského systému.

V ukázce je server iodined nastaven s tunelovacím rozhraním o adrese 10.0.0.1 a doménou moje.do:

```
iodined 10.0.0.1 moje.do
```

Server se po spuštění zeptá na heslo (je možné ho předat jako parametr). Klientská část je ještě jednodušší, zadává se pouze tunelovací doména:

```
iodine moje.do
```

Klient při připojování k tunelovacímu serveru ověřuje heslo. Od serveru získá adresu pro své tunelovací rozhraní z rozsahu toho serverového (tj. například 10.0.0.2) a MTU. Nastavení routování na obou stranách je shodné s ostatními tunely přes ICMP a DNS.

Samotná komunikace vypadá v praxi takto:

```
Queries
  Name: Vaaaaiaqaac.moje.do
  Type: NULL (Null resource record)
  Class: IN (0x0001)
```

Výpis 7: První zpráva od klienta s číslem verze

Klient poslal serveru zprávu s číslem své verze. Server ji ověří a v případě úspěchu odpoví klientovi s jeho identifikačním číslem a výzvou (challenge) pro přihlášení:

```
Answers
  Type: NULL (Null resource record)
  Class: IN (0x0001)
  Time to live: 0 time
  Data length: 9
  Data 56 41 43 4b 72 1c 5d 90 00 (VACKr.]..)
```

Výpis 8: Odpověď serveru s potvrzením verze

Klient spočítá na základě hesla odpověď a pošle ji serveru. Ten ověří přihlášení a pošle klientovi konfiguraci s IP adresami obou konců tunelu a MTU:

[3]Maximum Transmission Unit

```
Answers
  Type: NULL (Null resource record)
  Class: IN (0x0001)
  Time to live: 0 time
  Data length: 22
  Data 31 30 2e 30 2e 30 2e 31 2d 31 30 2e 30 2e 30 2e
     32 2d 31 30 32 34 (10.0.0.1-10.0.0.2-1024)
```

Výpis 9: Odpověď serveru s konfigurací klienta

Klient nejpozději každou sekundu pošle serveru dotaz – ten slouží nejen k umožnění přenosu dalších dat od serveru ke klientovi (zmiňovaný polling), ale také k udržení spojení jako aktivního z pohledu tunelovacího serveru.

Kapitola 3

Způsoby detekce

Obecným cílem je zamezit provoz nežádoucích a neznemožnit použití legitimních služeb. Je tedy často nemožné zakázat zcela nějakou službu (protokol) nebo sadu služeb, neboť by takové opatření zasáhlo všechny uživatele sítě.

Protože také obecně není možné předem a vždy poznat, že konkrétní paket či spojení slouží k jinému účelu, než je žádoucí (a to bez ohledu na mlhavost samotné definice), nemělo by být cílem detekce zcela zamezit provozu nežádoucích služeb, ale reálně omezit nežádoucí provoz a služby na míru, která nebude omezovat nebo narušovat správné fungování sítě, nebo na takový provoz upozornit správce (s minimem falešných hlášení, aby nebyl zahlcen zbytečným šumem). Lepších výsledků se dá dosahovat kombinací s dalšími postupy (například detekce a postihování konkrétních uživatelů, kontrola uživatelských stanic apod.).

Metody detekce lze rozdělit na dva základní druhy:

1. analýza přenášeného obsahu
2. statistická analýza toku dat

Analýza obsahu zkoumá obsah přenášených paketů, jejich hlavičky i datovou část, případně se pokouší porozumět obsahu použitého aplikačního protokolu.

Statistická analýza se zaměřuje na obecnější charakteristiky provozu – velikost a pravidelnost paketů, časy, objemy a poměry přenášených dat apod. Samozřejmě k výběru požadovaného druhu provozu často používá i analýzu obsahu, ale ta zde vystupuje jen jako filtr a není pro samotné vyvozování závěrů rozhodující.

V dalších kapitolách budou uvedeny analýzy dat získaných při testování jednotlivých metod tunelování, jak byly popsány v předchozí části, a rozebrány možnosti omezení a detekce daných technik.

3.1 Metody analýzy obsahu

3.1.1 HTTP

Sledování obsahu u HTTP je snadno realizovatelné – provoz je směrován přes HTTP proxy, takže je jeho obsah pod kontrolou správce.

S vlastním filtrováním je to ale již složitější – v poslední době je pomocí HTTP (a především pak webu) realizována velká škála různých služeb. S rozmachem AJAXu[1] se navíc mění i charakter provozu.

Jak bylo předvedeno u tunelovacího programu HTTP Tunnel, součástí alespoň prvního dotazu jsou specifické hlavičky. Bylo by tedy možné při jejich detekci celé spojení na cílový server ukončit a zakázat. Nicméně už samotná implementace počítá s tím, že hlavičky nemusí přes proxy server projít, a implementuje možnost jejich přenosu v rámci požadavku metody GET.

Navíc hlavičky nemusí být vždy jednoznačně omezené a dané, neboť webové služby, a tedy různé knihovny na jejich realizaci, zahrnuje stále více programů, a podle toho se mohou použité hlavičky výrazně měnit. I v případě omezení prohlížeče a striktního filtrování hlaviček nebrání nic zásadního tunelovacím programům v napodobování charakteristik dotazu vybraného prohlížeče – naopak, tato metoda se již v současnosti používá například u automatizované replikace webů, které se snaží proti takovým robotům kontrolou hlaviček (neúspěšně) bránit.

3.1.2 HTTPS

Analýza obsahu přenášených dat je u HTTPS téměř nemožná – jeho smyslem je právě zamezit třetím osobám v přístupu k těmto datům. Odhalení obsahu přenášených dat by bylo možné v případě, že šifrované spojení se vzdáleným serverem navazuje za klienta až jeho proxy, jak bylo zmíněno v kapitole 2.2. Toto řešení se však v praxi nepoužívá, takže může HTTPS proxy pouze zjistit, kam dané spojení vede, protože tuto informaci získá od klienta při žádosti o jeho sestavení. Používají se například omezení na vzdálený port 443, který

[1] Asynchronous Javascript And XML

HTTPS standardně používá, jenže to nejen nezabrání využití tunelu, který na tomto portu poběží, ale může i omezit uživatele legitimních HTTPS služeb, které jsou provozovány na portu jiném.

Další variantou by bylo využití tzv. Man in the middle útoku, kdy by se proxy vydávala za onen vzdálený server a podvrhla klientovi vlastní certifikát. Ten by si toho sice mohl velice jednoduše všimnout – webové prohlížeče přicházejí se stále nápadnějšími způsoby upozornění. V praxi ale uživatelé podobná varovná hlášení příliš nečtou, protože výchozí seznam certifikačních autorit v prohlížečích je omezený a správci webových služeb často používají i tzv. self-signed certifikáty, takže se s hlášeními o neznámých certifikátech setkávají velice často a už je téměř automaticky povolují. Takový „útok“ by měl tedy slušnou šanci na úspěch. To však nic nemění na jeho nesprávnosti, například právě vzhledem k další nežádoucí podpoře nesprávných bezpečnostních návyků uživatelů i jiným bezpečnostním rizikům spojeným s takovým řešením.

V tomto případě je tak nutné použít statistické metody, které na znalost obsahu dat nespoléhají. Jejich využití s HTTPS se věnuje kapitola 3.2.2.

3.1.3 ICMP

Pevné filtry

Nejjednodušším způsobem (po úplném zákazu ICMP zpráv) by mohlo být zakázání konkrétních tunelovacích paketů existujících implementaci, například podle výskytu IP hlavičky na začátku datové části zpráv z ICMPTX nebo konstantního, pevně daného identifikačního čísla ICMP paketu tamtéž. To by šlo realizovat existujícími filtry pro linuxový firewall iptables (modul `string`), ale toto řešení by bylo velmi náchylné na jednoduché změny ve způsobu přenášení dat tunelovacími programy a také by se nedokázalo vyrovnat s výskytem programů nových [4].

Implementace ICMP Echo/Reply (`ping`) v běžných operačních systémech plní datovou část zpráv předem daným vzorkem dat – Windows (XP) sekvencí písmen, Linux osmibytovým časem odeslání následovaným vzestupnou sekvencí bytů. Bylo by tak možné povolit pouze ty. Kromě problémů s výskytem neznámých systémů (nebo jejich IP stacků) znemožňuje tato metoda také využití možnosti plnění datové části uživatelsky definovaným vzorkem dat, kterou některé systémy poskytují (například program ping v Linuxu).

Učící se odchozí filtr

Jak bylo při testování nebo ověřením zdrojových kódů zjištěno, běžné implementace pingu v systémech nemění mezi jednotlivými pakety obsah datové části, a to ani v případě, že byl vzorek pro datovou část zadán uživatelem.

Naopak tunel, aby mohl přenášet uživatelská data, musí datovou část paketu měnit. Je tedy možné sledovat sérii paketů odeslaných jedním uživatelem na konkrétní stroj a v případě, že se obsah datové části mění, tyto přímo zahazovat nebo označovat pro práci s nimi v jiných částech firewallu.

Filtr se tedy bude učit vzorky pro jednotlivé klienty/cíle. Při přijetí ICMP Echo paketu od klienta pro konkrétní cíl zjistí, zda se shoduje s předchozím paketem pro tuto dvojici klient/cíl, a pokud ano, propustí ho dál. V případě, že se liší, je větší pravděpodobnost, že se jedná o tunelovací ICMP paket, a filtr jej tedy jako takový označí.

Tímto způsobem bude zachována možnost klientů používat libovolný obsah datové části ICMP Echo paketů bez speciální konfigurace na firewallu. Jediná situace, kdy bude postižen legitimní uživatel, bude případ, kdy bude opakovaně v krátkém intervalu měnit ručně obsah datové části ICMP Echo paketů. Ale i tehdy bude zasažen jen první paket z dané série, ostatní již projdou filtrem bez problémů. Tento dopad je možné dále redukovat vhodným nastavením stárnutí záznamů ve filtru. Bylo by též možné omezit tuto kontrolu na pakety se stejným identifikátorem, ale to by velmi usnadnilo obcházení filtru úpravou tunelovacího programu, kde by stačilo jen nahradit stávající rozlišování podle tohoto identifikátoru např. speciální hlavičkou v datové části atp.

U tohoto typu filtru je nutné počítat s proměnlivou první částí výchozího datového obsahu, kde je v případě linuxového pingu uložena časová značka paketu. To nabízí – byť omezený – prostor pro ukrytí uživatelských dat.

Sledování obsahu spojení

Další možnost detekce vychází z požadavku RFC 792, že cílový server musí v odpovědi odeslat datovou část ICMP paketu tak, jak ji obdržel [19]. Firewall tedy může sledovat, zda obsah přijatých odpovědí odpovídá dříve odeslaným dotazům na dané stroje. Znamená to sice skladovat pro odeslané ICMP pakety jejich obsah (nebo hash), ale stavové firewally stejně udržují seznam odeslaných dotazů, aby mohly rozhodnout, zda přijatá odpověď k nějakému z nich patří – jde o tzv. connection tracking, jehož cílem je zabránit průniku libovolných ICMP paketů do sítě (třeba za účelem napadení chybného IP

stacku) nebo přesměrovat odpověď u překladu adres na správný stroj uvnitř sítě.

Šlo by tedy o rozšíření tohoto mechanismu o sledování nejen adres, identifikátoru a počtu nepotvrzených paketů, ale i jejich obsahu. Na rozdíl od např. linuxového jádra, které si pamatuje pro dané spojení (definované adresami obou konců, typem, kódem a identifikátorem ICMP Echo/Echo reply) pouze počet paketů, ke kterým ještě nepřišla odpověď, je nutné si ukládat také označení jednotlivých zpráv. To sice zvyšuje paměťové nároky nebo snižuje možnosti uživatelů, hlavně u „flood pingu“, kdy jsou zprávy odesílány maximální možnou rychlostí, takže může snáze dojít k přeplnění vyhrazené paměti a nesprávnému zahazování odpovědí, ale zároveň zlepšuje bezpečnost sítě.

Tato metoda byla prakticky implementována a vyzkoušena. Při testech dokázala spolehlivě znemožnit použití ICMP tunelu již při první výměně zpráv a zároveň nekladla překážky legitimnímu využití zpráv ICMP Echo. Podrobnosti o implementaci jsou uvedeny v kapitole 4.

Metoda je velmi spolehlivá, neboť nespoléhá na detekci konkrétních vzorů ve sledovaných zprávách, takže ji nelze obejít jednoduchou úpravou tunelovacího programu, která by hledaný vzor změnila. Místo toho striktněji vyžaduje plnění použitých norem a principiálně znemožňuje přenos informací v datové části i hlavičkách. Je tedy použitelná obecně pro libovolný program tunelující data přes zprávy ICMP Echo, i když nebyl v době tvorby filtru znám.

3.1.4 DNS

Jak bylo předvedeno v analýze tunelování přes DNS, používá program NSTX pro přenos dat dotazy na záznamy typu TXT u dlouhých doménových jmen s obvykle dlouhými odpovědmi. Nabízí se tedy možnost takové dotazy omezit či zakázat. Podobným směrem se vydali například autoři pravidla pro IDS[2] Snort, které je k dispozici v balíku Community Rules [7]. Pravidlo ukazuje výpis 10.

[2]Intrusion Detection System

```
alert udp any any -> any 53 (msg:"COMMUNITY MISC
   Tunneling IP over DNS with NSTX"; byte_test:
   1,>,32,12; content: "|00 10 00 01|"; offset: 12;
   rawbytes; threshold: type threshold, track by_src,
   count 50, seconds 60; reference:url,nstx.dereference
   .de/nstx/; reference:url,slashdot.org/articles
   /00/09/10/2230242.shtml; classtype:policy-violation;
    sid:100000208; rev:1;)
```

Výpis 10: NSTX pravidlo pro Snort

Znamená to, že program sleduje dotazy delší než 50 bytů (32h), které jsou na záznamy třídy IN a typu TXT (reprezentované čtveřicí bytů `00 10 00 01`) a které se opakují častěji než padesátkrát za minutu.

Takové pravidlo se samozřejmě dá jednoduše obejít zkrácením maximální délky dotazu, i když to znamená snížení dosažitelné datové propustnosti tunelu, omezením četnosti dotazů za cenu odezvy nebo využitím jiného typu záznamu než TXT.

To poslední činí další zmiňovaný tunelovací program Iodine, který využívá záznamy typu NULL. Navíc má ale pevný formát zpráv při sestavování tunelu a přihlašování k tunelovací službě, takže by při využití stavového filtru (aby mohl sledovat komunikaci klienta se serverem přes několik dotazů) šlo takové spojení odhalit a zakázat.

Není však nutné využívat méně běžné druhy velkých DNS záznamů – klient může na dotaz na adresu dané domény (záznamy typu A nebo AAAA) získat více krátkých odpovědí, do kterých je možné data uložit. To může komplikovat situaci správcům, kteří by se rozhodli bojovat s tunelováním právě omezením množiny povolených typů záznamů.

3.1.5 Shrnutí

Metody analýzy obsahu se obecně spoléhají na hledání výskytu konkrétního vzorce v přenášených datech. Nejlépe se znalostí pouze jednoho, právě analyzovaného paketu – sledování spojení přes více paketů je paměťově i časově náročnější. Tyto metody jsou velmi náchylné na změny v tunelovacích programech – často jednoduchá úprava použitého programu lehce obejde pracně nasazovaný filtr obsahu, který je třeba znovu nastavit.

I přes zmíněné nevýhody jsou však metody analýzy obsahu, v poslední době známé pod zkratkou DPI – Deep Packet Inspection, velmi oblíbené pro svou jednoduchost.

3.2 Statistické metody

3.2.1 HTTP

Zajímavý příklad statistické detekce provozu přes HTTP uvádějí autoři Borders a Prakash v [9]. Zaměřili se na detekci nežádoucích programů běžících na klientských počítačích, především spyware a trojských koňů, které komunikují s útočníkem pomocí HTTP dotazů a tunelů. K jejich odhalení sledovali několik hodnot:

1. Tvar hlaviček HTTP dotazů – tento způsob je založen na analýze obsahu a slouží jako nejjednodušší filtr.

2. Rozestupy mezi dotazy – sledovali rozložení rozestupů mezi jednotlivými dotazy pro dané klienty na cílové servery. Z těchto dat byly vytvořeny souhrnné statistiky, které odhalily například automatické dotazy po několika různých intervalech (především půl minuty, čtyři a pět minut).

3. Velikost dotazů – podobným způsobem jako u časových rozestupů sestavili rozdělení velikosti dotazů a zvolili limit, který zahrnoval přes 99 % všech dotazů.

4. Celkový objem odchozích dat – tato statistika byla rozdělena podle cílových serverů. Zvolený limit opět zahrnoval přes 99 % serverů, což představovalo 48 překročení během týdne.

5. Pravidelnost provozu – zde měřili, kolik procent dne (v pětiminutových úsecích) probíhá se servery komunikace. Tím se dají odhalit pravidelné dotazy spywaru na ovládací server. Dále sledovali směrodatnou odchylku pravidelnosti komunikace – nízká hodnota ukazuje na zcela automatizované dotazy, neboť lidé odesílají požadavky nahodile a ve větších dávkách.

6. Doba provozu – neboť cílem bylo sledování nežádoucího chování nezávislého na uživateli, byly pro jednotlivé uživatele vypracovány profily

využívanosti proxy. Automatické programy tento vzor obvykle nedodržují, takže lze zjistit, zda nějaký takový program na klientském počítači běží.

Ve svém hodnocení popisují úspěšné odhalení všech použitých tunelovacích programů i úmyslně nasazeného spywaru a také několik výskytů nečekaných. Vyhodnocování dat však bylo prováděno ručně, navíc zjevně za spolupráce sledovaných osob. To je pro reálné nasazení velmi nepraktické, obzvlášť když by měli sledovaní uživatelé důvod použití tunelu tajit.

Autoři též přiznávají slabiny použitých sledovacích metod včetně zmínění možností, jak je obelstít. Navíc s rozmachem některých interaktivních webových služeb bude stále obtížnější rozeznat legitimní a nežádoucí druh provozu. Na web se přesouvají služby s dříve samostatnými klienty, jako jsou třeba diskuse a chaty, které mohou generovat pravidelné dotazy na server. Časté používání AJAXu přibližuje komunikační chování webového prohlížeče klasickým aplikacím. A využití stavových formulářů (především v ASP) nebo různých služeb založených na uživatelském obsahu výrazně zvyšuje objem odchozího provozu.

Ani ruční sledování podezřelých serverů nemusí být zcela účinné: pro útočníka by pak bylo výhodné umístit komunikační skript na veřejný hosting, kde by se ztratil ve skupině ostatních, legitimních webových stránek. Daný skript ani nemusí implementovat kompletní tunel, stačí jako předávací bod mezi klientem a cílovým tunelem. V tomto případě by bylo nutné sledovat konkrétní webovou stránku, ideálně ve spolupráci s uživatelem, neboť by jinak mohla být maskována za legitimní službu z výše uvedené skupiny.

3.2.2 HTTPS

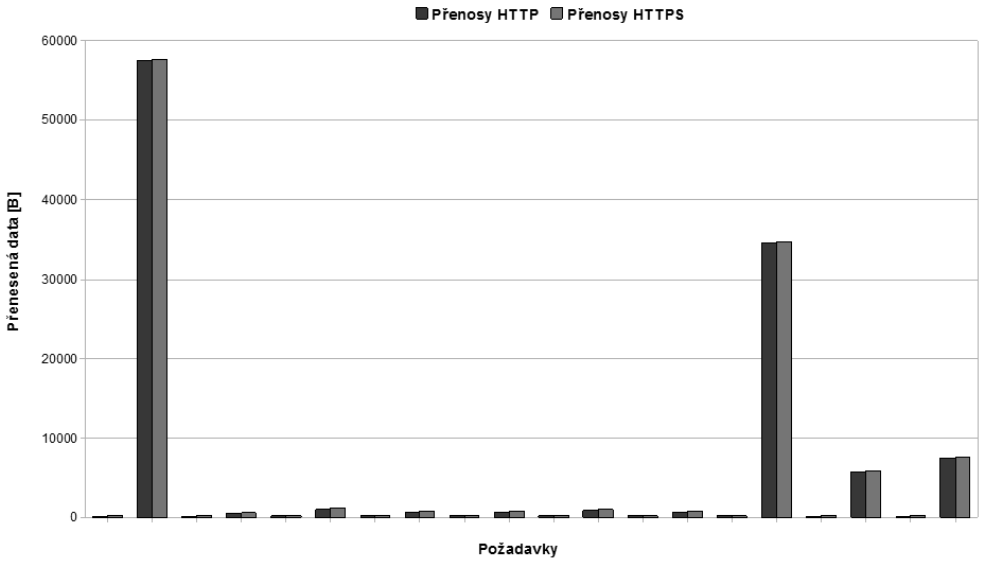
Jak již bylo zmíněno, statistické metody jsou prakticky jedinou možností, jak bojovat s nežádoucím tunelováním provozu přes HTTPS proxy.

Nejjednodušší naivní metody se mohou pokusit využít skutečnosti, že tunelovací programy se pochopitelně snaží udržet jedno spojení dlouhou dobu – viz například tunelování SSH v kapitole 2.2. Prohlížeče obvykle spojení po přenesení několika souborů ukončí, aby zbytečně nezatěžovaly vzdálené servery nečinnými spojeními, i když u HTTPS je jeho sestavení náročnější, a tak nemusejí být v jejich rozpojování tak aktivní jako u klasického HTTP.

Proxy servery tedy mohou dlouhotrvající spojení, která nepřenáší data, přerušovat uměle – v případě procházení webu je pouze nutné sestavit spojení

znovu, ale uživatel jinak nezaznamená žádné problémy. Často padající SSH spojení v případě tunelu však může jeho uživateli značně znepříjemnit práci. Na druhou stranu sofistikovanější tunelovací programy mohou použít vlastní protokol, kdy přerušení HTTPS spojení nebude znamenat pády tunelovaných připojení, nebo posílat keep-alive pakety generující zdánlivý provoz. Také je třeba vhodně zvolit okamžik přerušení spojení na proxy, nejlépe po získání odpovědi od serveru a před případným odesláním další žádosti. To je možné odhadnout ze směru přenášených dat.

Lze však použít zajímavějších možností. HTTPS spojení je totiž sice šifrované, tj. není možné přímo číst obsah přenášených dat, ale na jejich charakteristikách toku (round-tripy, četnost a téměř ani velikosti) šifrování nic nemění. To demonstruje graf na obrázku 3.1, porovnávající velikosti dat přenášených přes nešifrované HTTP a totéž pomocí HTTPS.



Obrázek 3.1: Velikosti požadavků při přenosu přes HTTP a HTTPS

V obou případech byly přeneseny ze stejného serveru tytéž stránky s připojenými daty z několika umístění, včetně navození chybového stavu (žádost o neexistující soubor). Charakter velikostí přenášených dat je zjevně podobný (rozdíl je především v přídavných hlavičkách samotného SSL) a počet přenosů je zcela shodný (po vynechání známých paketů sloužících pro sestavení samotného SSL spojení, které je možné rozeznat podle obsahu a se znalostí protokolu).

Je tedy možné použít takové metody detekce, které nespoléhají na znalost obsahu přenášených dat. Příkladem je [10], kde se autoři pokoušeli deteko-

vat tunelování spojení pomocí SSH. Cílem bylo odlišit „legitimní“ využití SSH – interaktivní sezení a přenos souborů přes SCP[3] – od spojení sloužících k tunelování jiných protokolů.

Sestavili charakteristiku série paketů poslaných mezi klientem a serverem (tzv. „okno“), u nichž sledovali jejich velikosti a rozestupy mezi nimi. Charakteristiky byly vytvářeny pro každý směr komunikace zvlášť. Velikosti byly brány pro Ethernet, tj. v rozsahu 40–1500 bytů a rozestupy na logaritmické škále.

Pro klasifikaci byl použit naivní bayesovský filtr – ten nejprve prošel fází učení na sérii zhruba čtyř tisíc zachycených interaktivních sezení a SCP přenosů. Poté byl tento naučený filtr použit na další sadu SSH spojení a byla vyhodnocena úspěšnost. Seznam druhů provozu, úspěšnosti jejich správného zařazení a absolutního počtu takových spojení uvádí tabulka 3.1.

Protokol	Úspěšnost	Počet spojení
SSH	98,95 %	600
SCP	99,65 %	1700
POP3 přes SSH	87,89 %	2360
SMTP přes SSH	99,93 %	4300
CHAT přes SSH	88,31 %	2100
P2P přes SSH	88,77 %	1600

Tabulka 3.1: Úspěšnost klasifikace provozu

Celý experiment však pro svou úspěšnost vyžadoval několik předpokladů. Zásadní byl výběr prvního paketu se skutečným užitečným obsahem pro dané sezení – především tedy vynechání přihlašovací fáze. To bylo zajištěno požadavkem na jednoduché přihlašování pouze asymetrickými klíči. Dalším požadavkem bylo nepoužívání komprese přenášených dat, kterou protokol SSH umožňuje.

Statistické metody tedy zjevně mohou nabízet zajímavé možnosti odhalování provozu přes jinak neprostupně šifrovaná spojení.

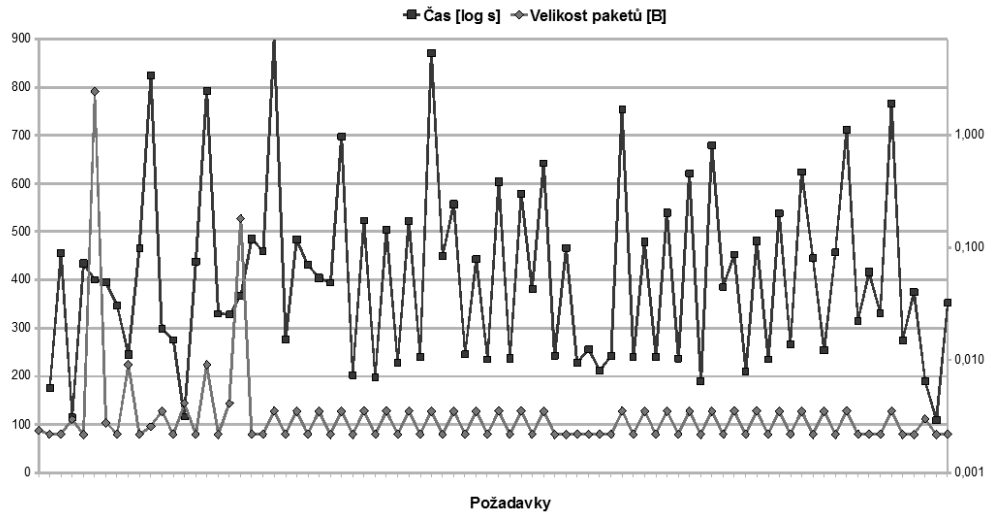
3.2.3 ICMP

Základní údaje

Programy testující dostupnost strojů odesílají zprávy ICMP Echo v pravidelných intervalech a se stejnou velikostí, jak již bylo uvedeno v kapitole o ana-

[3]Secure CoPy

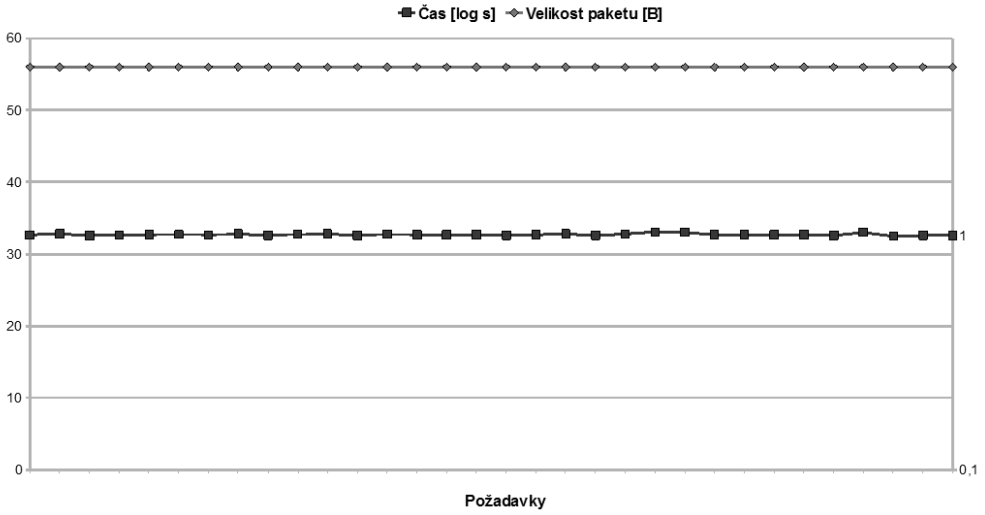
lýze obsahu. Naopak testované tunelovací programy odesílají data, když je mají k dispozici, a kolik jich mají k dispozici. To bylo ověřeno studiem zdrojových kódů i prakticky a demonstrují to grafy velikosti a časového rozestupu paketů odesílaných tunelovacím programem ICMPTX na obrázku 3.2.



Obrázek 3.2: Velikosti a časové rozestupy ICMP zpráv odesílaných programem ICMPTX

V tomto testovacím případě bylo přes tunel sestavováno SSH spojení. Změny velikosti jednotlivých odesílaných paketů jsou jasně patrné a neodpovídají očekávanému charakteru při běžném testu dostupnosti programy ping, jako je na obrázku 3.3.

Stejně ani rozestupy nejsou pravidelné a kopírují charakteristiky tunelovaného protokolu. Pro kvantifikaci těchto změn byly navrženy dva jednoduché postupy, které sledují intervaly mezi jednotlivými odesílanými pakety a zjišťují jejich změny.



Obrázek 3.3: Velikosti a časové rozestupy ICMP zpráv odesílaných při běžném spuštění programu ping

Změna rozestupu

První charakteristika počítá změnu doby odeslání aktuálního paketu proti předchozímu intervalu:

$$d_i = \left| \frac{I_i - I_{i-1}}{I_i} \right| * 100 \tag{3.1}$$

I_i je rozestup mezi dvěma následujícími pakety a d_i je vypočtená změna. Je vyjádřena v procentech aktuálního rozestupu, aby byl snížen vliv drobných nepřesností při delších pravidelných intervalech. Tabulka 3.2 ukazuje hodnoty získané pro skutečný ping a pro spojení tunelované pomocí ICMP.

Občasné skoky u reálného pingu mohou být způsobeny časem mezi různými spuštěními programu (data jsou vybírána pouze podle adres zdrojového a cílového stroje) – v praxi by tak byly izolované výkyvy ignorovány, příp. by byla data dále rozdělena podle identifikátoru, který by se měl při opakovaných spuštěních pingu měnit, zatímco u tunelu zůstává stejný.

U tunelovaného spojení ukazuje tabulka hodnoty o několik řádů vyšší, které jsou též mnohem častější, což vypovídá o velkých změnách v intervalech odesílání. Rozdíly mezi oběma druhy provozu jsou tedy v této charakteristice jasně patrné a navíc je její výpočet velmi snadný, rychlý a vyžaduje ukládání jen dvou hodnot – předchozího rozestupu I_{i-1} a času poslední odeslané ICMP zprávy.

Normální ping	ICMP tunel
0,0025	23809,5455
0,0007	92,6446
0,0005	85,7967
0,0036	2053,2209
0,0102	69,8660
0,0060	154,8488
0,0009	93,6498
0,0021	9,2331
0,0020	580,6685

Tabulka 3.2: Změny rozestupu d_i pro normální ping a ICMP tunel

Variační koeficient

Další vypočtenou charakteristikou je variační koeficient pro „okno“ stanovené velikosti, tj. sérii za sebou jdoucích paketů o zvolené délce L.

Okno W_i je definováno jako $W_i = \{I_x \mid i \geq x > i + L\}$.

Vzorec pro koeficient v_i vypadá takto:

$$v_i = \frac{s_i}{\overline{W_i}} * 100 \tag{3.2}$$

Zde je s_i směrodatná odchylka rozestupů v okně W_i, $\overline{W_i}$ je pak jejich průměr. Hodnoty v_i jsou opět v procentech. V tabulce 3.3 jsou výsledky pro klasický ping a tunelované spojení – stejná situace jako u předchozí charakteristiky.

Normální ping	ICMP tunel
1,8195	139,2744
0,0026	143,9107
0,0037	152,3699
0,0037	144,3178
0,0037	110,4778
0,0037	111,2362
0,0028	102,0037
0,0009	92,7936
0,0011	217,2458

Tabulka 3.3: Variační koeficient pro normální ping a ICMP tunel

V tomto případě se rozestupy mezi spuštěními projevují delším úsekem vysokých hodnot, je tedy nutné brát na zřetel vyšší hodnoty až pro úseky delší, než je zvolená velikost okna L. U normálních pingů jsou hodnoty obvykle pod jednou desetinou promile, u tunelu pravidelně překračují sto procent.

Problémy navržených charakteristik

Problémem pro tyto jednoduché způsoby sledování jsou však dvě situace:

1. Tzv. „flood ping“, tedy odesílání ICMP paketů nejvyšší možnou rychlostí. Rozestupy mezi pakety jsou tak malé, že i drobné odchylky v pravidelnosti odesílání, kterým není možné se na normálním systému vyhnout, působí velké nepravidelnosti.

2. Adaptivní ping, kdy program odesílá vždy jen jeden paket a čeká na jeho návrat před odesláním dalšího. V tomto případě jsou rozestupy silně ovlivněny stavem sítě mezi odesilatelem a adresátem paketů, který je mimo kontrolu uživatele a který se může velmi často měnit. Navíc ztráta paketu způsobí nárůst rozestupu až na hodnotu maximální doby čekání.

V obou těchto případech vycházejí charakteristiky obdobně jako u tunelovacího programu, neboť i rozestupy mohou být obdobné. Vypočtené charakteristiky pro ně a srovnání s dřívějšími hodnotami uvádí pro rozestupy tabulka 3.4 a pro variační koeficient tabulka 3.5.

Adaptivní ping	Flood ping	Normální ping	ICMP tunel
0,095819	20,7077	0,0025	23809,5455
0,002689	25,3903	0,0007	92,6446
92,335558	22,0372	0,0005	85,7967
157,368146	38,7556	0,0036	2053,2209
407,597121	15,0302	0,0102	69,8660
0,916701	17,0645	0,0060	154,8488
1,201418	10,5222	0,0009	93,6498
80,253755	24,4922	0,0021	9,2331
0,075651	37,7778	0,0020	580,6685

Tabulka 3.4: Procentuální doba změny rozestupu

V těchto případech může posloužit jako vodítko velikost paketů, kterou ping za běhu nemění, a závěry by byly vyvozovány z obou těchto veličin.

Adaptivní ping	Flood ping	Normální ping	ICMP tunel
0,0702	10,2485	1,8195	139,2744
158,0226	10,7335	0,0026	143,9107
124,0348	13,6334	0,0037	152,3699
124,0376	15,1070	0,0037	144,3178
123,9616	14,0720	0,0037	110,4778
123,9997	14,2175	0,0037	111,2362
84,6856	13,9303	0,0028	102,0037
84,5984	17,1903	0,0009	92,7936
64,6164	15,8357	0,0011	217,2458

Tabulka 3.5: Variační koeficienty

Další metody a shrnutí

Dalšími použitelnými metodami mohou být například analýza provozu z kapitol 3.2.1 a 3.2.2 o metodách HTTP a HTTPS, neboť IP pakety jsou obsaženy přímo v datové části ICMP zpráv a program ICMPTX je odesílá okamžitě, jakmile je má k dispozici. Tunel tedy nemění charakteristiky provozu, který přepravuje.

Ale tyto charakteristiky lze jednoduše změnit tak, aby odpovídaly očekávanému provozu přes ICMP, jak se právě u této metody nejvíce nabízí. Stačí doplňovat pakety na zvolenou konstantní velikost a odesílat je v předem stanovených intervalech – tím budou tunelovací ICMP zprávy vypadat jako běžný ping a metody pro analýzu charakteristiky paketů nebude možné použít.

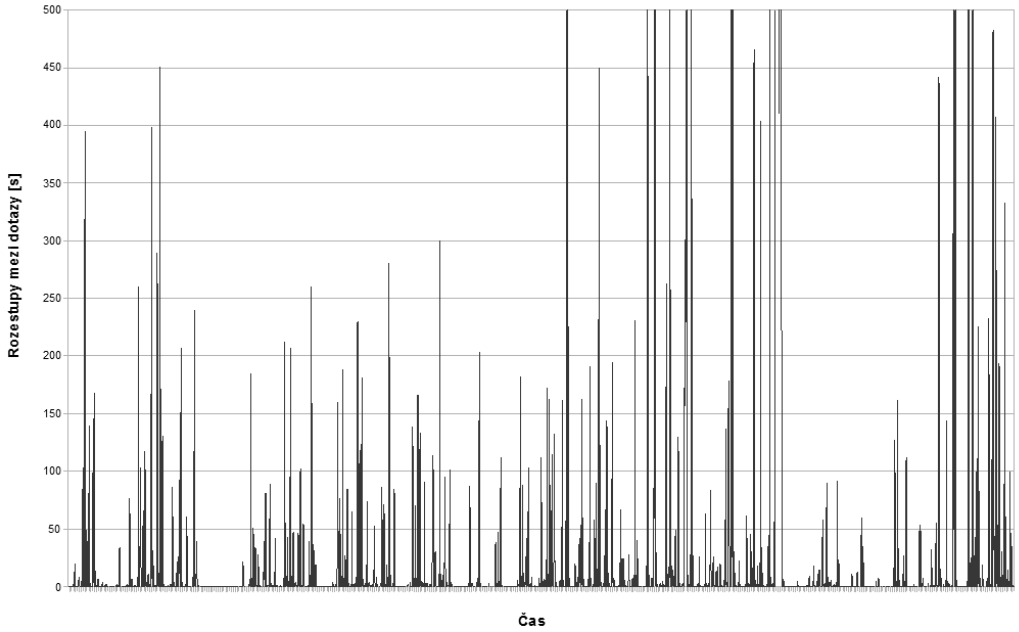
V tomto případě lze sledovat objemy provozu pro jednotlivé klienty a cílové hostitele, podle kterých lze usuzovat na nadměrný provoz. Ale pro testování sítě a problémů s fragmentací se někdy používají i ICMP pakety o velikosti až desítek kilobytů. Tím mohou být statistiky ovlivněny, i když tento způsob provozu nebývá příliš běžný.

Účinným řešením je metoda analýzy provozu uvedená v kapitole 3.1.3, která znemožňuje obousměrný přenos dat bez ohledu na velikost a časování paketů a která by spolehlivě postihla i upravený tunelovací program.

3.2.4 DNS

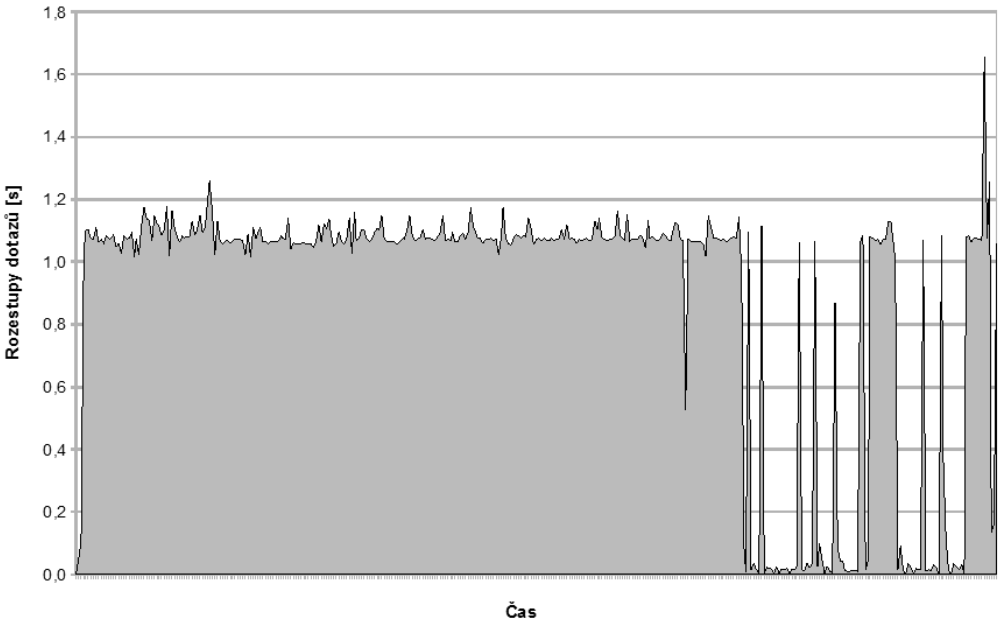
Jak demonstruje obrázek 3.4 získaný analýzou dlouhodobějšího DNS provozu „běžného“ klienta na několika tisících dotazech, jsou tyto dotazy vytvářeny ve

shlucích s nepravidelnými většími pauzami podle toho, jak sledovaný člověk požaduje různé služby. Na rozdíl od toho jsou rozestupy dotazů tunelovacích programů shora pevně omezeny: to je zřejmé z obrázku 3.5 a ze zdrojových kódů obou programů. Tyto programy se kvůli jednosměrnosti DNS a pro zachování rozumné míry odezvy pravidelně ptají „prázdnými“ dotazy svého vzdáleného serveru, aby mu umožnily zaslat klientovi data, i když ten sám právě žádné datové dotazy nevytváří. Drobná překročení programového limitu jedné sekundy jsou způsobena nepřesnostmi v generování dotazů knihovnami a operačním systémem.



Obrázek 3.4: DNS provoz „normálního“ uživatele (rozestupy)

Toho lze využít k detekci takového DNS tunelu – u něj nebudou mezi dotazy větší mezery, ale provoz bude pravidelně pokrývat souvislý časový úsek, během nějž je tunel v provozu. Naopak nepravidelnost provozu u normálního uživatele lze ještě zvýraznit použitím cachovacích nameserverů, které si pamatují výsledky předchozích žádostí a vracejí je bez opakování dotazu na jiné servery – vliv takového serveru (pro jednoduchost s pevně nastavenou životností záznamů na jednu hodinu) na naměřená uživatelská data ukazuje vzestup průměrné doby mezi dotazy z 30,76 sekundy na 68,33 sekundy a mediánu z 0,03 s na 0,22 s. Jak bylo předvedeno v kapitole 2.4 u popisu programu NSTX, jsou tunelovací dotazy generovány na různá doménová jména a odpovědi tunelovacího serveru mají nastavenu nulovou platnost, takže cachovány nebudou a ve statistice se projeví veškerý takový provoz.

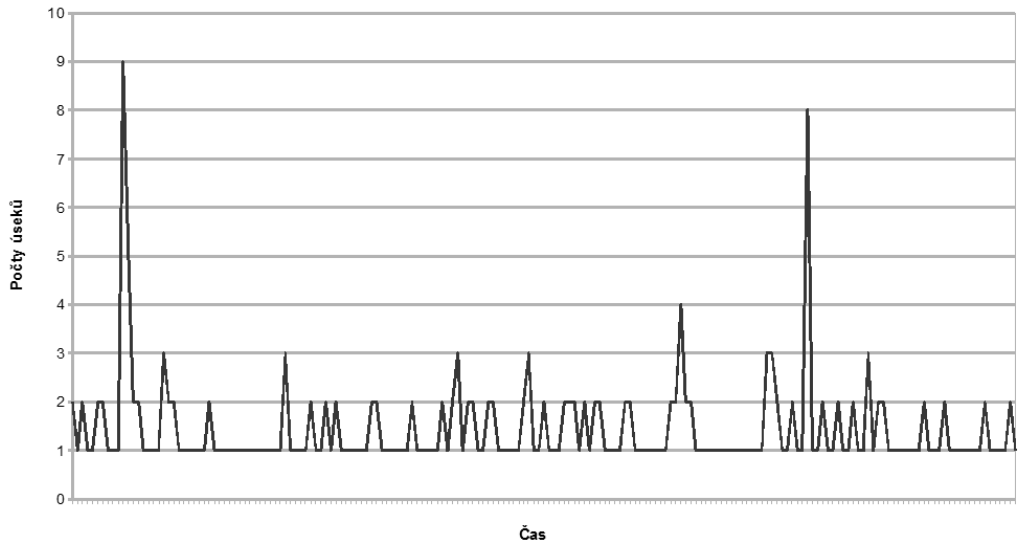


Obrázek 3.5: DNS provoz programu Iodine

Pro sledování pravidelnosti provozu byly zvoleny úseky o délce pěti sekund – pokud položil v daném časovém úseku klient nějaký dotaz, byl tento úsek pokládán za aktivní. Délka souvislých aktivních úseků u legitimního provozu je v grafu 3.6 – obvykle se pohybuje do 2 (průměr je 1,44, medián 1), maximum je pak 9 souvislých aktivních úseků. U tunelovacího programu jsou v takovém případě aktivní úseky všechny. Detekce tunelu tedy spočívá ve vhodném nastavení velikosti jednoho úseku a limitu na maximální počet souvisle aktivních. Délka úseku by měla být větší než očekávaný automatický časový limit tunelovacího programu a dostatečně malá, aby výsledná měření vystihla pauzy v legitimním provozu.

Dalších zlepšení by se dalo dosáhnout rozdělením dotazů podle nejvyšší specifické domény, tj. takové, která není TLD[4] nebo jinou obecnou doménou (to jsou zejména národní poddomény typu co.uk). Zatímco aktivita u normálního uživatele by byla rozprostřena mezi více takových domén, tunelovací server se dotazuje stále na tutéž doménu, která je delegována na tunelovací server.

[4]Top Level Domain



Obrázek 3.6: Délky souvislých aktivních úseků u legitimního DNS provozu

3.2.5 Shrnutí

I statistické metody mají slabiny, které vycházejí z veličin, jež sledují a podle nichž se rozhodují. Jsou jimi především velikost paketů, příp. její změny, průběh objemu přenesených dat a nepravidelnosti v jejich přenosu.

Tunelovací programy mohou velké množství těchto charakteristik ovlivňovat: Rozptyl velikostí odesílaných paketů lze snížit jejich doplňováním nedůležitými daty na požadovanou hodnotu, tzv. paddingem. Podobně lze upravovat též poměry v objemech odeslaných a přijatých dat. Rozptyl v časech odesílání jednotlivých paketů lze zase minimalizovat větší kontrolou nad jejich zasíláním třeba využitím front a časovačů.

Tyto metody se v současnosti u tunelů ještě nepoužívají. Ale je to především proto, že k tomu nemají autoři tunelovacích programů důvod – pokud by se rozšířily statistické metody filtrace a odhalování tunelů založené na sledování uvedených veličin, došlo by k odpovídající úpravě i u tunelů.

Kapitola 4

Popis praktické implementace

Pro implementaci byl vybrán filtr ICMP navržený v poslední části kapitoly 3.1.3, založený na analýze obsahu spojení. Jeho cílem je především zabránit použití ICMP zpráv pro tunelování.

Princip fungování byl popsán již ve zmiňovaném textu – filtr si ukládá otisky odesílaných ICMP zpráv Echo a zpět propouští jen takové odpovědi, které odpovídají některé z dříve odeslaných žádostí. Byl by tedy umístěn na firewallu vnitřní, chráněné sítě. Filtr je implementován jako samostatný program pro OS Linux – ten je často nasazován právě na firewally. Využívá knihovnu `libnetfilter_queue` pro práci s paketovou frontou firewallu iptables [5].

Data o odeslaných zprávách jsou ukládána do hashovací tabulky. Při jejím naplnění nad danou hranici se provede vyřazení tolika nejstarších záznamů, aby se naplnění snížilo pod další určený limit (ještě nižší – to zajišťuje, že se nebude tabulka opakovaně vyprazdňovat při oscilaci kolem jediné hodnoty naplnění). Pro rychlý a jednoduchý výběr těchto starších záznamů jsou prvky udržovány také ve zvláštním spojovém seznamu, do kterého jsou při uložení zařazeny vždy na konec – seznam je tak uspořádán podle stáří prvků. Hashovací tabulka používá výhradně separované řetězce bez uspořádání. I do těchto řetězců jsou záznamy ukládány na konec.

Tyto struktury byly zvoleny proto, že uložení nového prvku probíhá v čase $O(1)$, stejně jako vyřazení prvku, pokud je na něj znám ukazatel (prosté vypojení z řetězce a seznamu). Stejnou časovou složitost má i vyřazení nejstaršího prvku, který je vždy na začátku spojového seznamu. Jedinou náročnější operací je tak nalezení prvku, které má nejhorší časovou složitost $O(n)$, kde n je počet uložených záznamů. V praxi však odpovědi chodí obvykle v tom pořadí, v jakém byly odeslány žádosti, takže jsou hledané prvky na

začátku řetězců (opět využití jejich uspořádání podle stáří). Při testech bylo naměřeno zhruba 1,01–1,5 prohledaných prvků pro každé hledání.

Při běhu filtru se také nealokuje ani neuvolňuje žádná paměť haldy – všechny alokace jsou provedeny při startu programu, aby později nezdržovaly jeho běh. Nepoužité bloky jsou ukládány ve zvláštním zásobníku – přidání i vyjmutí tak trvá opět konstantní čas.

Implementace byla zkoušena s tunelovacím programem ICMPTX, kterému dokázala účinně zabránit – nepropustila zpět ani jedinou jeho zprávu. Prakticky tím ověřila funkčnost navržených principů filtrování. Filtr přitom propouštěl legitimní provoz ICMP paketů, takže jeho nasazení neomezovalo ostatní uživatele. Byly tedy splněny oba požadavky na účinné a cílené filtrování zvoleného protokolu.

Technické podrobnosti a přesný popis použití filtru jsou popsány v dokumentaci u zdrojových kódů programu a v příloze A.

Kapitola 5

Závěr

V první části práce byl představen problém aplikačních tunelů, jejichž existence může být pro mnohé správce sítí novinkou. Zabezpečení sítí bývá často prováděno pouhým nastavením několika jednoduchých pravidel firewallu a povolením vybraných služeb, bez jejich další kontroly. S rozšiřováním počítačových sítí mezi laiky, hlavně s rozmachem bezdrátových připojení, přenosných počítačů či tzv. chytrých telefonů, jsou také častěji sítě nasazovány nepoučenými nebo jen mírně poučenými osobami s využitím předpřipravených řešení. Typickým případem jsou hotely, restaurace či jiné společnosti, které chtějí svým zákazníkům poskytnout za úplatu možnost bezdrátového připojení k Internetu – pro minimalizaci nákladů jsou však nasazována tzv. „krabicová“ řešení, bez další kontroly či správy, nebo je realizací takového projektu pověřena nedostatečně poučená osoba. Všechny tyto situace otevírají možnosti zneužití nabízených prostředků způsobem, který je v rozporu s úmysly provozovatele.

Druhá část předvedla reálné programy, které takové možnosti využívají a poskytují připojeným klientům širší možnosti, než bylo zamýšleno. Někdy může jít o „neškodné“ provozování instant messagingu na pracovním počítači, ale třeba u zmiňovaných placených hot-spotů jde o jasnou finanční škodu.

Praktické testy také poskytly důležité informace pro analýzu použitých tunelovacích technik. Tyto informace byly použity v další části, která nabídla dva směry při odhalování aplikačních tunelů – analýzu obsahu přenášených dat a statistickou analýzu provozu. U obou přístupů byly zhodnoceny možnosti jejich využití pro daný druh aplikačního tunelu a byly navrženy konkrétní metody jejich využití. U každého druhu tunelu byly uvedeny i nedostatky těchto řešení a z tohoto pohledu byly zhodnoceny i oba obecné přístupy. Zajímavá metoda zabezpečení ICMP provozu, která vycházela ze standardu

a principů protokolu, a byla tedy dostatečně obecná a účinná, byla i prakticky implementována a úspěšně vyzkoušena.

Navržené metody samozřejmě nejsou jedinými způsoby, jak se před zneužíváním aplikačních protokolů tunely bránit. Například v uváděném příkladě DNS tunelu u placeného připojení s registraci z kapitoly 2.4 je mnohem lepším řešením než složitá analýza provozu jednoduše správná konfigurace sítě, která neumožní nedovolený přístup ke zdrojům mimo ni. To však naráží na již zmiňované problémy kvalifikovanosti obsluhy a možností úprav dodávaných řešení.

Navíc zabezpečení sítě není jednorázový úkon, po kterém již není nutné se o ni dále starat – byly uvedeny možnosti, které mají autoři tunelovacích programů k obcházení i pečlivěji nastavených filtrů. A je též důležité zvážit náklady na samotné nasazení striktnějších kontrol a pečlivějšího zabezpečení, aby nebyly zbytečně vynakládány prostředky na nástroje, které mohou uživatelé tunelů mnohem jednodušeji znovu obejít, a cena za boj s nežádoucím chováním nebyla větší než jím skutečně způsobené škody.

Dodatek A

ICMP filtr – programátorská dokumentace

`icmp-f` je filtrovací program pro ICMP. Jeho úkolem je především zabránit provozování tunelů přes tento protokol, ale umožnit využití ICMP legitimními uživateli. Sleduje odesílané zprávy ICMP Echo (typ 8) a přijímané ICMP Echo Reply (typ 0). U odchozích zpráv si ukládá jejich popis (zdrojová a cílová adresa, identifikační a sekvenční číslo a otisk dat) a při příchodu odpovědi zjišťuje, zda jde o reakci na zaslanou žádost. Pokud ne, takovou odpověď zahodí.

Jde o samostatný program pro OS Linux, který využívá knihovnu `libnetfilter_queue` pro práci s frontou paketů jaderného firewallu iptables [4, 5].

Použití

Program k běhu vyžaduje superuživatelská práva. Při spuštění je možné použít několik parametrů:

`-s <velikost>` Určuje velikost tabulky, do které jsou ukládána data o přijatých paketech. Výchozí hodnota je 1024.

`-h <horní>` Horní hranice naplnění tabulky (jako desetinné číslo). Výchozí hodnota je 0.9, tedy 90% naplnění. Při dosažení této hranice jsou z tabulky odstraněny nejstarší záznamy, aby bylo uvolněno místo.

`-l <dolní>` Dolní hranice naplnění tabulky (jako desetinné číslo). Výchozí hodnota je 0.8, tedy 80% naplnění. Při čištění tabulky je odstraněno vždy tolik záznamů, aby naplnění kleslo pod tuto hranici.

`-t <čas>` Maximální doba platnosti záznamu v sekundách. Výchozí hodnota je 300, tedy pět minut.

Pakety se z iptables programu předávají pomocí pravidla s cílem `QUEUE`. Je tedy nutné přidat podobné pravidlo:

```
iptables -A FORWARD -p icmp -j QUEUE
```

To zajistí předání ICMP paketů, které mají být směrovány, do fronty ke zpracování. Filtr pak rozhodne, zda budou pakety přijaty, nebo odmítnuty. Po ukončení programu je nutné toto pravidlo odstranit, jinak se budou pakety ve frontě hromadit a nebudou zpracovávány!

Popis implementace

Po spuštění programu jsou zpracovány případné parametry určující vlastnosti datového úložiště. To je následně inicializováno (`init_structures()`). Funkce `do_work()` se připojí k frontě netfilteru, zaregistruje callback `pkt_cb()` pro zpracování paketů a pak z fronty vyzvedává čekající data a předává je ke zpracování funkcí `nfq_handle_packet()`.

Zpracování paketů

Callbacková funkce `pkt_cb()` ověří vlastnosti přijatého paketu (tj. protokol a typ) a předá informace o paketu ke zpracování – to jsou zdrojová a cílová adresa paketu, identifikační a sekvenční číslo ICMP zprávy, její datový obsah a čas přijetí. V případě, že jde o žádost (zpráva ICMP Echo), uloží paket pro pozdější kontrolu odpovědi (funkce `store_icmp()`) a paket přijme (`NF_ACCEPT`). U odpovědi si naopak ověří, zda k ní byla někdy dřív odeslána příslušná žádost (funkce `check_icmp()`). Pokud ano, paket přijme, jinak ho zahodí.

Funkce `store_icmp()` spočítá otisk dat ICMP zprávy a společně s popisem ho uschová do úložiště (funkce `save_pkt()`). Vyhledávací funkce `check_icmp()` spočítá stejnou charakteristiku zprávy a dotáže se na její existenci úložiště zpráv. Zdrojová a cílová adresa jsou zaměněny, protože paket putoval opačným směrem než žádost! Výsledek vrací jako návratovou hodnotu (0 – žádost nenalezena, $\neq 0$ nalezena a vymazána).

Datové úložiště

Program používá pro ukládání informací o přijatých žádostech datové úložiště, jehož rozhraní je definováno v hlavičkovém souboru `storage.h`. Sestává ze tří funkcí:

`save_pkt()` – uloží informace o zprávě

`find_n_erase_pkt()` – najde (a případně ihned smaže) informace o zprávě

`init_structures()` – inicializuje datové úložiště se zvolenými parametry velikosti, hranic naplnění a doby platnosti záznamů

Volitelně je možné použít funkci `dump_stats()`, která v případě, že byl program přeložen s definovaným symbolem `DEBUG`, vypisuje počet žádostí, celkem prohledaných uzlů, využití a velikost tabulky a počet jejích vyčištění.

Implementovaný modul ukládá přijaté zprávy ICMP Echo do hashovací tabulky se separovanými řetězci. Pro uložení údajů o paketu (zdrojová a cílová adresa, identifikační a sekvenční číslo, otisk obsahu a čas vypršení) jsou používány uzly typu struktury `queue_node`. Samotná tabulka slouží pouze pro rychlou adresaci řetězců, všechny uzly s daty jsou ukládány do řetězců – tabulka se tedy skládá jen ze struktur `icmp_info`, které obsahují ukazatele na první a poslední uzel v řetězci (obousměrném spojovém seznamu). Při kolizích jsou nové uzly zařazovány na konec řetězce.

Uzly s přijatými zprávami jsou také zařazovány na konec obousměrného spojového seznamu (hlava `queue_head` a ukazatel na konec `queue_tail`). Tento seznam je využíván k odstraňování starých záznamů při čištění tabulky; záznamy jsou v něm uspořádány podle stáří. Nepoužité uzly jsou udržovány v zásobníku (spojovém seznamu určeném ukazatelem `empty_head`).

Tyto datové struktury byly zvoleny proto, že umožňují rychlé provádění několika operací:

- Přidání nového datového uzlu: Uzel je přepojen ze zásobníku prázdných uzlů na konec seznamu uzlů použitých a na konec řetězce na vypočtené pozici v hashovací tabulce. Tato operace se provádí při ukládání nového záznamu.
- Odebrání nalezeného datového uzlu: Uzel je odebrán z řetězce hashovací tabulky (libovolné pozice v něm) a ze seznamu použitých uzlů (také z libovolné pozice). Je zařazen na zásobník uzlů prázdných. K odebrání uzlu stačí znát jeho adresu. Tato operace se provádí při nalezení záznamu o žádosti ke zpracovávané odpovědi.

- Odebrání nejstaršího datového uzlu: Nejstarší uzel je vždy na začátku spojového seznamu za hlavou `queue_head`. Tím je známa jeho adresa a je možné použít rychlé odebrání z předchozího bodu. Tato operace se provádí při čištění tabulky.

Všechny tyto operace vyžadují konstantní čas. Jedinou operací, která může vyžadovat více práce, je hledání záznamu ke zpracovávané odpovědi. V nejhorším případě může být časová složitost této operace $O(n)$, kde n je počet uložených záznamů, ale v praxi bývá menší. Nejen nebývají všechny záznamy uloženy v jednom řetězci, ale odpovědi obvykle přicházejí ve stejném pořadí, v jakém byly zaslány žádosti – a protože jsou uzly v řetězci uspořádány podle stáří, jsou hledané záznamy většinou na začátku řetězce.

Příklad: Klient odešle postupně žádosti A, B, C, které budou v tabulce uloženy do stejného řetězce. Budou tam tedy uloženy v pořadí A, B, C. Pokud budou postupně zpět přicházet odpovědi A', B' a C', budou jednotlivé hledané záznamy vždy jako první v řetězci. Ten se bude zkracovat: A, B, C $\xrightarrow{A'}$ B, C $\xrightarrow{B'}$ C $\xrightarrow{C'}$ prázdný. Počet prohledávaných prvků tak bude vždy 1.

Při dosažení stanoveného limitu zaplnění tabulky dojde k jejím vyčištění – je odebráno tolik nejstarších záznamů, aby její naplnění kleslo pod nastavenou úroveň. Také jsou odstraněny případné další záznamy, kterým vypršela platnost. Takové záznamy jsou odstraňovány i při prohledávání řetězce v tabulce – zabraňuje to situacím, kdy není tabulka naplněná dostatečně pro automatické vyčištění, ale zastaralé záznamy zdržují vyhledávání aktuálních žádostí. Navíc odstraňování záznamů při jejich procházení nezpůsobuje zhoršení celkové asymptotické časové složitosti, protože se záznamem se beztak pracuje.

Další podrobnosti

Program je psán pro OS Linux a využívá knihovnu `libnetfilter_queue`, která nahrazuje starší rozhraní `libipq` (původní verze filtru byla napsána pro toto rozhraní). Program byl testován v systému Debian Sid (Lenny) s jádrem 2.6.25 a balíky knihoven libnetfilter-queue1 verze 0.0.16 a libnfnetlink0 verze 0.0.39.

Program sestává ze souborů `icmp-f.c`, `storage.h` a `storage.c`. K programu je dodáván také ukázkový soubor `Makefile`, který program přeloží. Program má licenci GNU GPL v2.

Literatura

[1] Firedrill.
URL <http://www.fire-drill.com/>

[2] FreeCap.
URL <http://www.freecap.ru/eng/>

[3] Hamachi.
URL <https://secure.logmein.com/products/hamachi/vpn.asp>

[4] iptables.
URL <http://www.netfilter.org/projects/iptables/index.html>

[5] libnetfilter-queue.
URL <http://www.netfilter.org/projects/libnetfilter_queue/index.html>

[6] Net:Bridge.
URL <http://www.linuxfoundation.org/en/Net:Bridge>

[7] Snort.
URL <http://www.snort.org>

[8] Wireshark.
URL <http://www.wireshark.org/>

[9] Borders, K.; Prakash, A.: Web Tap: Detecting Covert Web Traffic. 2004.
URL <http://www.sigcomm.org/ccr/drupal/files/p7-v37n1b-crotti.pdf>

[10] Dusi, M.; Crotti, M.; Gringoli, F.; aj.: Detection of Encrypted Tunnels across Network.
URL <http://www.ing.unibs.it/~gringoli/pub/PID578397b.pdf>

[11] Fielding, R.; Gettys, J.; Mogul, J.; aj.: RFC 2616: Hypertext Transfer Protocol – HTTP/1.1. 1999.
URL <http://www.ietf.org/rfc/rfc2616.txt>

[12] Force, I. E. T.: RFC 1122: Requirements for Internet Hosts – Communication Layers. 1989.
URL <http://www.ietf.org/rfc/rfc1122.txt>

[13] IANA: Port Numbers. 2008.
URL <http://www.iana.org/assignments/port-numbers>

[14] Information Sciences Institute, U. o. S. C.: RFC 791: Internet Protocol. 1981.
URL <http://www.ietf.org/rfc/rfc791.txt>

[15] Leech, M.; Ganis, M.; Lee, Y.; aj.: RFC 1928: SOCKS Protocol Version 5. 1996.
URL <http://www.ietf.org/rfc/rfc1928.txt>

[16] Luotonen, A.: Web Proxy Servers. 1997.
URL <http://tools.ietf.org/id/draft-luotonen-web-proxy-tunneling-01.txt>

[17] Mockapetris, P.: RFC 1034: Domain Names – Concepts and Facilities. 1987.
URL <http://www.ietf.org/rfc/rfc1034.txt>

[18] Mockapetris, P.: RFC 1035: Domain Names – Implementation and Specification. 1987.
URL <http://www.ietf.org/rfc/rfc1035.txt>

[19] Postel, J.: RFC 792: Internet Control Message Protocol. 1981.
URL <http://www.ietf.org/rfc/rfc792.txt>

[20] Vixie, P.: RFC 2671: Extension Mechanisms for DNS (EDNS0). 1999.
URL <http://www.ietf.org/rfc/rfc2671.txt>