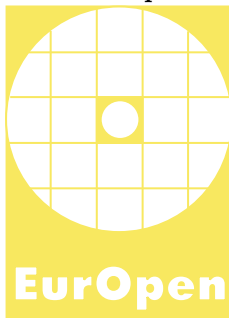


Česká společnost uživatelů otevřených systémů EurOpen.CZ
Czech Open System Users' Group
www.europen.cz



42. konference
Sborník příspěvků



Zámecký hotel Třešť
12.–15. května 2013

Programový výbor:

Pavel Šimerda, Netinstall

Vladimír Rudolf, ZČU Plzeň

Pavel Růžicka, Sunsec

Sborník příspěvků z 42. konference EurOpen.CZ, 12.–15. května 2013

© EurOpen.CZ, Univerzitní 8, 306 14 Plzeň

Plzeň 2013. První vydání.

Editor: Vladimír Rudolf

Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis, Plzeň

e-mail: servis@vydavatelskyservis.cz

Tisk: Typos, tiskařské závody, s. r. o.

Podnikatelská 1160/14, Plzeň

Upozornění:

Všechna práva vyhrazena. Rozmnožování a šíření této publikace jakýmkoliv způsobem bez výslovného písemného svolení vydavatele je trestné.

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-25-9

Obsah

Jiří Eischmann Kam se podělo GNOME 2?	5
Anička Marešová Linux jako Real Time operační systém	11
Luboš Doležel Darling – spouštění aplikací pro OS X na Linuxu	15
Jaroslav Řezník Fedora project management	23
Marek Žehra, Miro Hrončok 3D tisk na FIT ČVUT	29
Michal Zima WrapSix, aneb nebojme se NAT64	35
Pavel Šimerda NetworkManager pro servery a virtualizaci	39
Radek Nováček Správa sítí pomocí OpenLMI	43
Jan Hrach Pasivní útoky na GSM	45
Radoslav Bodó, Daniel Kouřil, Jiří Sitera, Miloš Mulač, Pavel Vondruška Harvesting Logs and Events Using MetaCentrum Virtualization Services	55
Ondrej Mikle Dane (SSL v DNS) (CZ.NIC)	69

KAM SE PODĚLO GNOME 2?

Jiří Eischmann

E-MAIL: EISCHMANN@REDHAT.COM

GNOME

Ač to nemuselo být z pohledu uživatele příliš patrné, prostředí GNOME a celá jeho platforma se nenacházely před pár lety v dobrém stavu. Vývoj se výrazně zpomalil, počet vývojářů upadal, platforma a její klíčové částky jako knihovna GTK+ nebyly dostatečně atraktivní pro vývojáře jiných projektů. Vzhled prostředí byl taky čím dál více považovaný za nemoderní.

První diskuse nad novou generací GNOME započaly na User Experience Hackfest 2008 a následně vyústily v oznámení GNOME 3.0 v dubnu 2009. Tímto datem počínaje se dá říct, že GNOME mělo konkrétní plán, jak novou generaci GNOME vytvořit, i když byl samozřejmě ještě dále přepracováván a rozšiřovaný.

Prvním krokem mělo být vyčištění platformy. Nacházelo se v ní totiž velké množství starých a těžko udržitelných knihoven a i z hlavních knihoven GTK+ a Glib se stal kvůli lpění na zpětné kompatibilitě špatně udržitelný moloch obsahující celou řadu zastaralých funkcí. Cílem tedy bylo zastaralé knihovny úplně odstranit a GTK+ a Glib zbavit zastaralých funkcí. Zvolena byla metoda postupného přechodu, kdy byly dopředu označeny knihovny, které budou odstraněny, a v kódu GTK+ a Glib označeny funkce, které v příštích verzích zmizí. Vývojáři tak dostali nějaký čas, aby se závislosti na těchto knihovnách a funkcích ve svých programech zbavili. Tyto změny probíhaly ještě v rámci vydání druhé generace. V GNOME 2.28 byly odstraněny závislosti na esound, libgnomevfs, libgnomeprint a libgnomeprintui. V GNOME 2.30 pak zmizely závislosti na libart_lgpl, libbonobo(ui), libglade, libgnome, libgnomecanvas, libgnomeprint(ui), libgnomeui, libgnomevfs.

Zatímco práce na knihovnách probíhaly už v rámci posledních vydání GNOME 2, tak trochu mimo zrak uživatelů, započaly práce také na zcela novém rozhraní GNOME. Rozhraní dosavadní generace bylo víceméně definováno dvěma moduly: GNOME Panelem, který zprostředkoval panely v prostředí, a Metacity, správcem oken GNOME. S Panelem se už nepočítalo, ač v rámci

GNOME zůstal až do vydání 3.6. Metacity potom bylo přepracováno tak, aby využívalo grafickou knihovnu Clutter. Spojením těchto dvou slov vznikl také nový název – Mutter. Clutter Metacity přinesl celou řadu nových možností, ale také jedno v té době stále výrazné omezení. Je postaven na OpenGL, takže Mutter nebyl schopen fungovat na počítačích, pro něž nebyly k dispozici grafické ovladače s podporou hardwarové akcelerace.

GNOME Panel byl nahrazen prostředím, které dostalo název GNOME Shell. Odlišuje se nejen vzhledově, ale i technologicky. Jedná se o plugin pro Mutter napsaný v JavaScriptu. Správce oken a rozhraní desktopu tak byly fakticky sjednocené do jednoho procesu. GNOME Shell se vyvíjel vedle standardních vydání GNOME a několikrát výrazněji změnil vzhled. Od začátku si však držel základní myšlenku – mít dva režimy, jeden, v kterém bude věnován maximální prostor práci v aktuálnímu pracovnímu oknu, a druhý, v kterém zase bude celkový přehled prostředí (spuštěná okna, spuštěné aplikace, plochy, ...). Druhý režim dostal od začátku název Activities (do češtiny překládané jako Činnosti). Změny v ovládání byly oproti GNOME 2 fundamentální, což byl a stále také je nejzásadnější problém u přijetí uživateli. Byl odstraněn panel úloh a mezi okny se nyní přepíná pomocí náhledu oken v Činnostech, případně pomocí Alt+Tab. Lišty oken ztratily tlačítka pro minimalizaci a maximalizaci. Minimalizace v novém systému přepínání mezi úlohami ztratila smysl a maximalizace lze nyní dosáhnout gestem. Zmizela oznamovací oblast z horního panelu a částečně ji nahradil skrývací panel v dolní části obrazovky. Počet ploch už není statický, ale číslo se mění dynamicky podle toho, kolik jich uživatel používá, a seřazené jsou pouze vertikálně. V režimu Činnosti se také objevil panel s oblíbenými a spuštěnými aplikacemi, známý z OS X jako Dock, v GNOME však nazývaný Dash.

Zásadní změna oproti GNOME Panelu je také v rozšiřitelnosti. Panel měl systém pluginů, ale ty měly jen velmi omezené možnosti, prakticky fungovaly jen jako prvky na panelu. GNOME Shell umožnil rozšíření, která se aplikují přímo do javascriptového kódu Shellu. To umožnilo prakticky neomezenou rozšiřitelnost a upravitelnost. Posléze vznikla také stránka <https://extensions.gnome.org/>, která nabízí již několik set prověřených rozšíření, které lze instalovat přímo z prohlížeče. Přineslo to ale i jednu nevýhodu. Tím, že rozšíření modifikují přímo kód Shellu, můžou mít také zásadní vliv na jeho stabilitu a uživatel často zjišťuje, že za častými pády Shellu je jedno z nainstalovaných rozšíření. Další výraznou změnou v oblasti rozhraní je také to, že jak Shell, tak motivy GTK přešly na kaskádové styly, takže definování vzhledu je prakticky totožné s tím, jak se definuje vzhled u webových stránek.

Vývoj nové generace se nakonec protáhl o rok. GNOME 3.0 se mělo stát vydání GNOME 2.30, ale nakonec to bylo až vydání GNOME 2.34. Ač si myslím, že se vývojáři vyvarovali mnoha chyb KDE 4.0 a z pohledu stability se jednalo o docela úspěšné vydání, přesto se nesetkala nová generace s příliš pozitivním ohlasem. Nový Shell byla hodně radikální změna a mnoho nových prvků

se ukázalo jako nedomyšlených a problematických, např. nový systém upozornění. Vývoj nové generace tedy s vydáním verze 3.0 zdaleka neskončil.

GNOME 3.2 přineslo hlubší integraci instant messagingu do Shellu, umožňující vyhledávat kontakty přímo v něm. Přibyl také správce vyměnitelných disků ve skrývacím panelu. A GTK+ získalo možnost alternativního tmavého vzhledu, který můžou volitelně používat multimedialní programy. Objevil se taky systém online účtů, který umožňuje přihlášení k webovým službám pouze jednou a ostatní aplikace mohou toto přihlášení používat. Jedná se o řešení, které je dnes běžně hlavně na mobilních platformách.

GNOME 3.4 přineslo zásadní změnu v oblasti aplikací. V plánech pro novou generaci GNOME totiž nebyly pouze změny knihoven a prostředí, ale také změny v aplikacích a vybudování široké nabídky desktopových aplikací. I designéři GNOME se rozhodli, že se chtějí zbavit aplikační nabídky v okně. Některé systémy ji celou přesunuli do horního panelu (OS X, Unity). GNOME pod ikonu aplikace v horním panelu přesunulo pouze položky týkající se aplikace jako celku. Položky, týkající se konkrétního okna, se zapracovaly do rozhraní aplikace. Nové aplikace se také naučily v maximalizovaném stavu skrýt lištu okna tak, aby byl prostor pro aplikaci maximální. Objevily se také nové aplikace jako Dokumenty a Kontakty.

Byla také částečně odstraněna jedna ze zásadních nevýhod nového GNOME a to, že Mutter vyžaduje hardwarovou akceleraci. Ovladač llvmpipe totiž umožnil vykreslovat akcelerované akce i pomocí procesoru, takže GNOME nově běželo i na špatných ovladačích. Nevyřešilo to problém se slabými počítači, protože takové řešení samozřejmě vyžaduje výkonný procesor. Umožnilo to ale vývojářům zbavit se GNOME Panelu, který byl stále součástí GNOME jako záložní řešení, ale už se velmi špatně udržoval.

Další zásadní milník ve vývoji GNOME nastal na konferenci GUADEC 2012, kde se vývojáři shodli na tom, že většinu zásadních úkolů, spojených s přechodem na novou generaci už překonali, a je nutné nastavit, jakým směrem se bude projekt dále ubírat. Dlouho se třeba spekulovalo o tom, jestli je GNOME 3 určené na mobilní zařízení. Na posledním GUADECu se ale dohodlo, že primárními cílovými zařízeními i nadále zůstávají desktopová zařízení, přesto ale bude v designu GNOME myšleno i na dotykové ovládání, protože čím dál více desktopů a notebooků už má i dotykové obrazovky.

Poslední dvě vydání (3.6 a 3.8) přinesly změnu v posledních do té doby téměř netknutých modulech – GNOME Screensaver a GDM. GDM převzalo od Screensaveru také uzamykání obrazovky, o které se nyní stará. Uzamčená obrazovka potom zcela vizuálně proměnila. Přibylly také nové aplikace: Bijiben (poznámový blok), Počasí, Fotky, Boxes (virtualizace). Začal se taky řešit mechanismus distribuce aplikací v GNOME. Vývojáři momentálně navrhuji řešení, které by přineslo sandboxy pro aplikace, což by výrazně zvýšilo bezpečnost, a také to, jak vývojářům třetích stran zajistit stabilní prostředí pro aplikace minimálně po dobu

několika vydání. Vývojářům by také mělo usnadnit život ucelené SDK. Dosud totiž platilo, že GNOME nabízelo příliš mnoho způsobů, jak napsat aplikaci, ale ani jeden nebyl dotažený do konce. Proto se rozhodlo, že se vybere jeden výchozí a tomu se bude věnovat nejvíce pozornosti, ať už v podobě dokumentace nebo IDE. Pro grafická rozhraní byl vybrán JavaScript, pro nízkourovňové věci potom C. Pracuje se také na zcela novém nástroji pro instalaci softwaru a jde vidět, že pozornost vývojářů se postupně přesunuje od samotného prostředí k aplikacím.

Unity

V době, kdy začaly vznikat první návrhy GNOME Shellu, přišla společnost Canonical, stojící za Ubuntu, s vlastním rozhraním Unity. Do té doby Ubuntu používalo GNOME 2. Původně bylo Unity určeno pouze pro netbooky s malou obrazovou a bylo stejně jako GNOME Shell pluginem pro Mutter. V té době měl ale Mutter docela velké výkonnostní problémy a tak se vývojáři rozhodli přepsat Unity jako plugin pro Compiz, který byl v té době nejpoužívanějším kompozitním správcem oken a používala ho i celá řada distribucí s GNOME 2. Podle slov vývojářů Unity, uvažovali o tom, že jej navrhnou jako výchozí prostředí pro novou generaci GNOME místo Shellu. Nikdy k tomu ale nedošlo. Místo toho se Canonical rozhodl, že Unity bude i nadále vyvíjet sám a bude jej používat jako výchozí prostředí i v desktopové verzi. Nejednalo se však o náhlou změnu kurzu, ale spíše o vyvrcholení několikaletého procesu, kdy se Ubuntu postupnými změnami oproti upstreamu začalo od GNOME oddalovat.

V dnešní době Unity i nadále funguje formou pluginu do Compizu, ale co se zdálo jako výhoda na začátku, ukázalo se dlouhodobě jako koule na noze. Mutter se během několika vydání výrazně zrychlil a naopak Compiz se ukázal jako pomalý a dnes již špatně udržovatelný kompozitor, který už používá prakticky jen Unity. Vývojáři tedy oznámili, že Unity v dohledné době přepíší do QML, což už si zkusili i u verze Unity 2D (Compiz má totiž stejný problém jako Mutter). Canonical také vyvinul verze Unity pro tablety a mobily, taktéž v QML. S Qt+QML také počítá jako hlavním frameworkem pro aplikace a také si vyvíjí vlastní display manager Mir jako náhradu za X11, zatímco GNOME deklarovalo, že bude podporovat Wayland. Oddalování GNOME a Unity bude tedy s největší pravděpodobností i nadále pokračovat.

Cinnamon

Prostředí Cinnamon je dílem vývojářů distribuce Linux Mint, která dříve sázela na GNOME 2. Cinnamon nejdříve začal jako Mint GNOME Shell Extensions, tedy jako sada rozšíření pro GNOME Shell. Autoři Mintu totiž souhlasili s tech-

nickými změnami v GNOME 3, ale už ne s rozhraním. Postupem času se ale ukázalo, že udržovat vlastní rozhraní pouze jako sadu rozšíření je neudržitelné, a tak vznikl plnohodnotný fork. Od verze 1.2 pak došlo kvůli změnám nekompatibilním s upstreamem také k forku Mutter, který dostal název Muffin. Cinnamon je prostředí, které se odlišuje především tím, že se drží paradigmatu Windows 95, a také má pár vlastních aplikací a nástrojů (např. fork Nautilu Nemo), jinak se ale drží mnohem blíže GNOME než Unity.

MATE

MATE je desktopem, který také vznikl v dílně vývojářů Linux Mintu. Na rozdíl od Cinnamonu není forkem GNOME 3, ale GNOME 2 a je forkem kompletním. To znamená, že vývojáři forknuli všechny části prostředí včetně aplikací. Technologicky se MATE víceméně rovná GNOME 2. Pouze jednotlivé části byly přejmenovány, aby mohly v distribucích koexistovat s původními aplikacemi z GNOME. V MATE tedy nalezneme Marco místo Metacity, MATE Panel místo GNOME Panelu, Caja místo Nautilu atd. Vývojáři MATE se také snaží o určitý vývoj. Přesunuli se třeba z mate-conf (fork Gconfu) k GSettings, nebo z mate-corba (fork zastaralé knihovny Bonobo) k D-Bus. Uvažují také, že by výhledově portovali MATE na GTK+ 3, čímž by se znovu částečně přiblížili GNOME a Cinnamonu. Otázkou také je, jestli projekt dlouhodobě přežije. Věnuje se mu totiž pouze několik vývojářů, což je docela málo na relativně rozsáhlou platformu knihoven a aplikací.

LINUX JAKO REAL TIME OPERAČNÍ SYSTÉM

Anička Marešová

E-MAIL: ANICKA@ANICKA.NET

Když se zajímáme o kvality operačního systému, přejeme si také, aby byl rychlý. Aby mezi okamžikem, kdy se něco má stát (tedy například klikneme myší a očekáváme odezvu) a okamžikem, kdy se skutečně něco stane (třeba se otevře okno), uplynula co nejkratší doba. Této době říkáme latence a pokud je ve většině případů kratičká, tak se při tom, jak obvykle používáme počítač, dovedeme smířit s tím, že je někdy výrazně delší.

U některých typů úloh si ale příliš dlouhé prodlevy nemůžeme dovolit. Urychlovač v nemocnici musí paprsek vypnout hned, a ne až když doběhne jiná úloha na pozadí a pacient je mezitím spálen na troud. Airbag v autě musí vystřelit hned, a ne až když se palubní CD přehrávač vyrovná s vadou na disku. Zpoždění je zkrátka úplně stejný neúspěch, jako když k akci nedojde vůbec.

Operační systémy, která nám slibují konkrétní maximální latenci (samozřejmě jen pro jeden proces s nejvyšší prioritou) nazýváme real time. Pokud to dělají deterministicky a latence jsou zaručené (tohle samozřejmě musí podporovat i hardware a těžko toho dosáhneme mimo specializovaná zařízení), hovoříme o hard real time. Pokud jen slibují, že budou dělat, co bude v jejich silách, jde o soft real time.

Různých implementací real time založených na Linuxu existuje několik, my se ale budeme věnovat jen jedné, té nejzajímavější: Sadě patchů pro kernel pocházející od Inga Molnara, která Linux mění v real-time operační systém a postupně se dostává do standardního jádra.

Linux na PC může být při nejlepším soft real time, ale ani dosáhnout toho není vůbec jednoduché. Obvyčejný víceuživatelský systém může mít latence zcela neomezené. Klasický jednoduchý příklad založený na inverzi priorit (viz [1]): Máme úlohu A s nejvyšší prioritou, B s prostřední prioritou a C s nejnižší prioritou. Úloha C zamkne nějaký zámek a než se dostane na konec kritické sekce, probudí se úloha A a úlohu C předběhne. A se pokusí zamknout tentýž zámek a zablokuje se. C pokračuje, ale než se jí podaří odemknout zámek, předběhne jej B. B má větší prioritu než C, ale nižší než A. Přesto ve skutečnosti B předběhne A a může běžet libovolně dlouho.

Takových problémů je celá řada. Real-time patch se s nimi vyrovnává tak, že v současné době není problém pro produkční nasazení na běžném hardware zaručit latenci 100 mikrosekund. (Zaručit v tomto případě ovšem znamená to, že když bude latence horší, bude se to řešit jako bug, ne to, že to nikdy nenastane.) Průměrné latence jsou přitom horší než na standardním kernelu, zaručení maximální latence má nezanedbatelnou režii. Situace se ale postupně zlepšuje.

Jedním ze zdrojů velkých latencí je obsluha přerušení. Pokud například úloha s nízkou prioritou čte nebo zapisuje velký objem dat na pevný disk, tak tím zdrží i všechny ostatní úlohy. Real-time patch tento problém řeší tak, že rutiny zajišťující obsluhu přerušení (alespoň těch, u kterých to lze) ukrývá do vláken. Úloha s vysokou prioritou potom nemusí čekat, než úloha s nízkou prioritou vyřídí své IO, latence se podstatně sníží: dojde k vyvolání přerušení, zamaskování přerušení, probuzení vlákna obsluhujícího přerušení a návrat zpět do přerušenoého kódu. To celé na dnešních počítačích zabere jen jednotky mikrosekund.

Další problém způsobují situace, kdy úloha s nízkou prioritou dostane do kritické sekce, tedy do kódu, který je třeba vykonat atomicky, aby nedošlo k nestabilitě systému nebo poškození dat. U takového kódu musí systém zajistit, aby jej nevykonávalo víc procesů najednou. Na jednoprocesorovém systému se to řeší obvykle tak, že dokud proces kód nevykoná, nemůže být přeplánován, na více procesorech je ale třeba přidat zamykání. Linuxové jádro používá spinlocky, zámky, které čekají aktivně. Pokud proces narazí na zamčený zámek, zkouší jej ve smyčce odemknout tak dlouho, dokud se mu to nepodaří. Pokud je kritická sekce krátká a její vykonání trvá kratší dobu, než by trvalo uspat a zase probudit čekající proces, tak se to vyplatí, v Linuxu ale spinlocky chrání i rozsáhlé kritické sekce, dokonce bývají vnořené. Spousta dlouho se točících zámků je v systému zdrojem velkých latencí. Real-time patch tento problém řeší tak, že dělá tam, kde to lze, ze spinlocků spící zámky.

Výše zmíněný problém s inverzí priorit je vyřešen tím, že tyto spící zámky implementují dědičnost priorit. Pokud úloha A s nízkou prioritou drží zámek, který chce zamknout úloha B s vysokou prioritou, priorita A je zvýšena na úroveň B až do okamžiku, kdy A zámek odemkne. Algoritmus, kterým je tento problém řešen, je poměrně komplikovaný (viz [2]), výsledek toho příkladu je ale následující: Úloha C z nyní není předběhnuta úlohou B, protože v okamžiku, kdy se úloha A pokusila zamknout zámek držený C, úloha C získala stejnou prioritu jako A, která je vyšší, než priorita B.

Toto byla jen ochutnávka problémů spojených s implementací podpory pro real time pro Linux. Další podrobnosti lze najít například v článcích [2] nebo [3].

Literatura

- [1] Steven Rosted, RT-mutex implementation design,
[/usr/src/linux/doc/Documentation/rt-mutex-design.txt](#)
- [2] Sven-Thorsten Dietrich, Daniel Walker, The Evolution of Real-Time Linux
<http://www.cse.nd.edu/courses/cse60463/www/amatta2.pdf>
- [3] Paul McKenney, A realtime preemption overview
<http://lwn.net/Articles/146861/>

DARLING – SPOUŠTĚNÍ APLIKACÍ PRO OS X NA LINUXU

Luboš Doležel

E-MAIL: DOLEZEL@ARGONIT.CZ

Darling je velmi mladý projekt, který se pokouší odpovědět na to, o čem se napříč internetem vedly různé debaty: je možné na Linuxu spouštět aplikace pro OS X, když už můžeme spouštět aplikace pro Windows díky Wine?

Odpověď na tuto otázku prozradím hned zkraje. Možné to je, ale podobně jako Wine je nutné značné množství úsilí vzhledem k odlišnostem obou operačních systémů. A právě těmito odlišnostem bude věnována první část tohoto textu.

Odlišnosti mezi OS X a Linuxem

Jádro

Okolo toho, na jakém jádře (kernelu) vlastně OS X stojí, panuje velký zmatek. Mnoho lidí někdy něco slyšelo, pak to někdo někomu pověděl, někdo to zaslechl, až z původní informace příliš nezbylo. Někdo slyšel něco o FreeBSD, někdo jiný zase něco o mikrokernelu. Tak jak to vlastně je?

Pravdou je, že OS X běží na jádře zvaném XNU. XNU je hybridní jádro složené z jádra Mach (na kterém stojí i GNU Mach, neboli jádro systému GNU Hurd) a z části převzaté právě z jádra FreeBSD. Výsledek je zajímavý tím, že jádro má dvě sady systémových volání, které se navíc vzájemně funkčně překrývají.

V místech překryvu funkčnosti se dá obecně říci, že hlavní roli hraje Mach; „duplicitní“ volání BSD tedy skončí interně přeložené na volání Mach (ačkoliv nejde o překlad 1 : 1). Některé bezpečnostní techniky na BSD je možné dokonce přes volání Mach obcházet, jiná volání BSD dokonce nefungují vůbec (příklad: ptrace).

Pěknou ukázkou překryvu funkčnosti je programátorům na Linuxu známé volání clone() (pomocí něhož se implementuje fork()). Jeho nejbližším příbuzným voláním Mach je task_create()/thread_create(), které rovněž vytvoří nový proces/vláknko. Ukázka:

```

ledger_t ledgers[1] = { 0 };
task_t child;
thread_t thread;

/* Vytvoříme nový proces */
task_create(mach_task_self(),
            /* Rodičem je aktuální proces (task) */
            ledgers, 1,
            TRUE, /* sdílet paměťový prostor */
            &child);

/* Vytvoříme první vlákno v novém procesu */
thread_create(child, &thread);

/* Nastavíme entry point */
thread_load(thread, entry_procedure, NULL);

/* Spustíme vlákno */
thread_resume(thread);

```

Toto by mohlo *volně* odpovídat následujícímu kódu:

```

void* stack = malloc(1024*1024);
int thread;

thread = clone(entry_procedure, stack,
              CLONE_NEWPID|CLONE_VM|SIGCHLD, NULL);

```

Aniž bych chtěl zacházet do podrobností, možná jste už zaznamenali odlišnost přístupu API mezi systémovými volání na Linuxu/BSD a na jádře Mach. Zatímco ta linuxová se zpravidla vždy týká aktuálního procesu (nebo vlákna), v případě Machu se vždy určuje, čeho se má operace týkat. Jeden proces – za předpokladu, že má potřebná oprávnění – tedy může vytvářet vlákna v cizím procesu, snadno zapisovat do jeho paměťového prostoru a dělat další vylomeniny.

Mach nabízí i spoustu jiných volání než jen pro práci s procesy. Najdeme tam volání pro práci s locksety (pole mutexů), porty (IPC mechanismus), výjimkami (podobné SEH na Windows, případně signálům na Linuxu) a tak podobně.

Důležité je ale to, že Mach není mezi vývojáři pro OS X zrovna v oblibě. Ani sám Apple tato API programátorům „netlačí“, obecně se doporučuje využívat volání BSD, respektive funkce knihovny Libc.

Na závěr už jen doplněk: jádro XNU není navzdory svému původu v jádře Mach mikrokernelem.

Uživatelský prostor

Základní knihovny

Všechny „obvyklé“ programy na OS X používají knihovnu Libc, která podobně jako glibc, uClibc a jiné na Linuxu zpřístupňuje systémová volání a další základní užitečnou funkčnost. V případě OS X je Libc vypůjčeno od FreeBSD s tím, že obsah byl pochopitelně přizpůsoben odlišností jádra XNU a byly doplněny některé dodatečné funkce.

Obecně řečeno panuje mezi OS X a Linuxem na úrovni (g)libc stejná kompatibilita API jako mezi FreeBSD a Linuxem. A to je pro implementaci „emulační“ vrstvy věru užitečné. Hlavním zádrhelem je to, že kompatibilita API je znamená bezproblémové přenesení mezi systémy jen pokud máme k dispozici zdrojové kódy pro rekompilaci. Tím skutečným klíčem je až kompatibilita na úrovni ABI, ke které se dostaneme později.

Frameworky

OS X má opravdu obrovské množství různorodých frameworků pro stavbu GUI aplikací, práci s videosoubory, přístup ke zvukovému systému, přístup k hardwaru a tak dále a tak dále. Názvy těchto frameworků nejsou uživatelům Linuxu příliš známé – uveďme AppKit, Foundation, CoreFoundation nebo třeba CoreAudio. Přesto jsou první tři jmenované díky projektu GNUstep k dispozici i na Linuxu s určitou mírou kompatibility s původními API od Apple. A co teprve takový WebKit, ten vám jistě něco říká!

Situaci je tedy možné zhodnotit tak, že některé frameworky považované za specifické pro OS X jsou v různé míře rozpracovanosti dostupné i pro Linux. Jde ale o práci komunity, neboť Apple zdrojové kódy těchto vývojově nejnáročnějších součástí nezveřejňuje. Zde se dostáváme k další pověře o OS X.

OS X není v žádném případě open source

Příznivci OS X by mě jistě urychleně odkázali na stránky opensource.apple.com, tam ale najdeme především zdrojové kódy cizího open source kódu (tedy různý GPLv2 kód od Free Software Foundation a komunity obecně) a do toho zdrojové kódy jen vybraných nízkoúrovňových komponent systému. Kompilace těchto „projektů“ je obtížná, často mají vypreparované vnitřnosti a někdy dle mých zjištění ani neodpovídají tomu, co je na OS X skutečně dodáváno.

Abychom to shrnuli, ze zdrojových kódů od Apple si OS X ani při velkém úsilí nesestavíte a vzhledem k dosti kolísavé kvalitě tohoto kódu a ne úplně dobré licenci APSL (Apple Public Source License) mohou posloužit spíše jen jako pomůcka, aby programátoři mohli nahlédnout pod kapotu.

Objective-C

Díky popularitě embedded zařízení s operačním systémem iOS (embedded bratříček OS X) – tedy iPod, iPhone, iPad a jiné – nabývá na popularitě i jazyk Objective-C (neplést s C++). Dá se říci, že zatímco jedna část frameworků od Apple nabízí API v jazyce C, pro přístup k druhé části je nutné používat Objective-C. Zjednodušeně řečeno, bez Objective-C moderní GUI aplikaci pro OS X nebo iOS nenapíšete.

Podpora Objective-C runtime na Linuxu je pro Darling další z kritických komponent. I zde je možné se obrátit na projekt GNUstep, jehož součástí je knihovna libobjc2 nabízející velmi podobnou množinu API jako knihovna objc4 na OS X.

Jazyk Objective-C pro svůj běh na runtimu silně závisí. Cílem tohoto textu není vás zasvětit do tajů tohoto jazyka, přesto uvedu nejvýraznější specifika.

- Možnost reflexe – všechny třídy, metody, členské proměnné a další věci jsou za běhu registrovány v runtimu. Je možné je volně procházet. Aby to nebylo málo, všechna volání metod probíhají dynamicky – na rozdíl od jiných jazyků zde není reflexe jako „bonus“, naopak je pro tento jazyk klíčová.
- Snadná rozšiřitelnost – knihovna může kupříkladu doplnit stávající třídu o další metody.
- Možnost uskutečňovat různé triky – za běhu je například možné nahradit implementaci metody jinou metodou.
- Lehce nezvyklá syntaxe – obdobou `object->addValue(5)` z C++ by byl tento kód v Objective-C:

```
[object addValue: 5];
```

- Jednoduchost – přes zmiňovanou nezvyklou syntaxi jde stále o rozšíření jazyka C. Nenajdete v něm žádné komplikované věci jako v C++: žádné šablony, žádné přetěžování a tak podobně.

Výše uvedené vlastnosti například znamenají, že volání metod neprobíhá přímo (tak jako volání běžných funkcí nebo nevirtuálních metod v C++), nýbrž se k tomu používá runtime. Ukázka kódu Objective-C `[object addValue: 5]` by na úrovni jazyka C byla zapsána takto:

```
objc_msgSend(object, sel_registerName("addValue:"), 5);
```

A pokud byste dekompilovali program napsaný v Objective-C, skutečně byste tam podobný kód našli.

Píšeme „emulátor“, nikoliv emulátor

Dynamický zavaděč

Srdcem každého „emulátoru“, jakým je Wine, je (dynamický) zavaděč, tedy kód, který dokáže přečíst souborový formát spustitelných programů používaných na zdrojové platformě a zopakovat kroky původního zavaděče na platformě cílové.

Na Linuxu se dynamický zavaděč programů nachází obvykle v souborech `/lib/ld*`:

```
$ ls -l /lib/ld*  
/lib/ld-linux.so.2  
/lib/ld-linux-x86-64.so.2
```

Na OS X se pro zajímavost nachází v souboru `/usr/lib/dyld`.

Zde je postup spouštění programů na Linuxu (a na OS X obdobně):

1. Program použije systémové volání pro spuštění jiného programu. Předpokládejme, že jde o program ve formátu (např. ELF), který jádro podporuje.
2. Jádro přečte hlavičky programu a zjistí, zda jde o program staticky, nebo dynamicky slinkovaný.
3. Pokud jde o staticky slinkovaný program, zpracuje všechny povely v programu (především mapování různých segmentů programu do paměti) a spustí jej.
4. Pokud jde o dynamický program, přečte si název dynamického zavaděče (například `/lib/ld-linux.so.2`). Takový dynamický zavaděč byl měl být statickým programem. Namísto původního programu nechá spustit zavaděč (krok 3) a pole argumentů posune doprava – cesta k původnímu programu tedy bude prvním argumentem pro program zavaděče.

Jaké úkoly navíc oproti jadernému zavaděči má zavaděč dynamický?

- Rekurzivně načítá dynamické knihovny.
- Relokace – přesun programu v paměti na jiné místo, než pro které byl připraven.
- Vyhledávání symbolů z knihoven a zápis jejich adresy do programu.
- Podpora různých speciálních voleb:
`LD_PRELOAD`, `LD_LIBRARY_PATH`, ...

Pokud tedy do systému přidáme další dynamický zavaděč (jaký je součástí Wine nebo Darling), rozhodně se nedá hovořit o emulaci. Takový zavaděč může být naprosto rovnocenný dynamickému zavaděči ELF.

To, že je dynamický zavaděč plně implementován v uživatelském prostoru, si může každý snadno ověřit. Jakýkoliv dynamický program je kromě obvyklého postupu možné spustit i takto:

```
$ readelf -l /bin/uname | grep interpreter
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
$ /lib64/ld-linux-x86-64.so.2 /bin/uname -o
GNU/Linux
```

Implementace funkcí

Načtení programu do paměti je jen začátek. Tou nejnáročnější prací je poskytnout programu všechny funkce (API), které pro své fungování potřebuje. Ty musejí kopírovat funkce z původního systému nejen svým chováním, svým API, ale také svým ABI.

Funkce používané aplikacemi pro OS X se dají při spouštění na Linuxu rozdělit do několika skupin:

1. Plně kompatibilní i na úrovni ABI (příklad: `opendir()`).
2. Plně kompatibilní i na úrovni ABI, ale pod jiným názvem (příklad: na co se expanduje použití `errno`).
3. Plně kompatibilní jen na úrovni API (příklad: `uname()`).
4. Chybějící funkce (úplně nebo chybějící vlastnosti funkce).
5. Chybějící celé frameworky.

Ke skupině 3 si ukážeme dva příklady. Na ukázkovém systému A i B se shoduje API (to znamená, že je možné program bez problémů zkompileovat na obou platformách), nicméně ABI jsou odlišná.

Příklad 1

Systém A

```
#define FLAG_READONLY 0x2
int doSomething(int flags);
```

Systém B

```
#define FLAG_READONLY 0x1
int doSomething(int flags);
```

Pokud budeme v programu mít volání `doSomething(FLAG_READONLY)`, při kompilaci na systému A bude použito číslo 2, kdežto na systému B bude použito číslo 1.

Příklad 2

Systém A

```
struct data
{
    int unit;
    int value;
};
void getData(struct data* pData);
```

Systém B

```
struct data
{
    int value;
    int unit;
};
void getData(struct data* pData);
```

Zde se ABI liší v umístění polí ve struktuře `struct data`. Pokud by program zkompileovaný pro systém A byl spuštěn na systému B, kde by ale funkce `getData()` očekávala, že prvním polem ve struktuře je „value“, pak program obdrží hodnoty polí prohozené.

Jak si stojí Darling

Dynamický zavaděč

Dynamický zavaděč Darlingu již více či méně plně podporuje formát programů Mach-O pro platformy i386 a x86-64 ve všech podvariantách – Apple formát v průběhu vydání OS X různé rozšiřoval a upravoval.

x86 je systémem OS X podporováno až od řady 10.4. Od verze 10.0 až po 10.5 včetně byl tento operační systém dostupný pro platformu IBM PowerPC. Jelikož Apple svým přechodem k Intelu ukončil podporu PowerPC, zájem o tuto platformu již prakticky vymizel i mezi vývojáři. Případná budoucí podpora PowerPC v Darlingu je tedy zajímavá jen z historického hlediska a není prioritou. Tento postoj je posílen faktem, že jediný PowerPC hardware, který se běžně nachází mezi uživateli, je vyjma původního hardwaru od Apple jediné Sony PlayStation (3). PlayStation 4 už bude používat taktéž procesory x86.

Poslední platformou, kterou by mohlo být zajímavé podporovat, je ARM. Šlo by tedy o podporu aplikací pro iOS. Podpora ARM (a s tím související podpora API dostupných na iOS) by přinesla možnost spouštět tyto aplikace i na Androidu nebo ARM počítačích s Linuxem obecně.

Běžná API libc a systémová volání

Hlavní branou k funkcím libc a systémovým voláním (ať už BSD nebo Mach) je na OS X knihovna zvaná libSystem.dylib.

Darling již drtivou většinu funkcí libc poskytuje a to samé lze říci o voláních BSD. Oproti tomu volání Mach jsou buď podporována jen omezeně nebo vůbec.

Pro plnohodnotnou implementaci volání Mach by bylo nutné napsat jaderný modul pro Linux, neboť specifika těchto volání není možné napodobit z uživatelského prostoru. Překážkou je to, že takový modul by se musel pomoci různých „háčeků“ napojit na různá místa v systému, což není bez úpravy jádra jako takového úplně jednoduché.

Do budoucna se jeví jako východisko tyto háčky rozmístit pomocí jaderných API pro LSM (Linux Security Modules). Od určité verze Linuxu ale nemohou žádná LSM být přítomna v modulech. To by se do budoucna naštěstí mohlo změnit.

Historická API

Historická API jsou nejnepříjemnější částí práce. Typicky jde o API pocházející ještě z dob Mac OS (vyvíjeného do roku 2001), která sice Apple postupně označuje jako zastaralá, ale vývojáři se jich přesto stále drží.

Demotivující je jednak pocit práce na čemsi „mrtvém“, podivnost těchto API (složitě funkce se složitým fungováním) a často i obtížná implementace (některé vlastnosti nedávají v kontextu Linuxu moc smysl, ale aplikace na nich závisí).

Další frameworky

Díky projektu GNUPstep (vedeného pod Free Software Foundation) jsou základní frameworky pro stavbu GUI aplikací dostupné už teď. Mým cílem je do tohoto projektu dále přispívat a soustřeďovat tak práci na jediném místě.

Ostatní frameworky (jako CoreAudio pro přístup ke zvukovému systému) jsou během na dlouhou trať a je proto obtížné odhadovat, kdy která konkrétní aplikace pro OS X bude fungovat na Linuxu.

Aktuální stav

V současnosti je možné skrze Darling spouštět nejrozličnější konzolové aplikace. Fungují také jednodušší grafické aplikace.

Rychlost rozvoje je omezena velmi nízkým zájmem o projekt mezi vývojáři v komunitě.

FEDORA PROJECT MANAGEMENT

Jaroslav Řezník

E-MAIL: JREZNIK@REDHAT.COM

Úvod

Fedora je jedna z nejrozšířenějších a nejoblíbenější linuxových distribucí současnosti (do jaké míry nechám raději na čtenáři) a mezi její základní hodnoty patří být první (v zavádění novinek), přátelství v komunitě, nové vlastnosti a hlavně svoboda ve smyslu svobodného software a svobodného obsahu.



Vznik Fedory se datuje do roku 2003, kdy vyšla první verze distribuce s názvem Fedora Core 1. Ta byla založena na Red Hat Linuxu jako reakce na vznik Red Hat Enterprise Linuxu. První skutečně komunitní Fedora však spatřila světlo světa až s vydáním sedmé verze Fedory – již pod současným názvem, kdy došlo ke sloučení Fedora Core (platformy kontrolované Red Hatem) a Fedora Extras (komunitou spravovaných rozšiřujících repozitářů). Letos tedy Fedora oslaví své desáté narozeniny.

Kdo řídí Fedoru?

Vývoj velké distribuce, jakou Fedora je, kde na jednotlivých vydání pracují desítky lidí, je potřeba nějakým způsobem koordinovat a vést. V rámci komunity je tak ustanoveno několik základních orgánů (FESCo, Board, FPC atd.) a další

skupiny vznikají časem dle potřeby s různou formou formálnosti. Od malých týmů pracujících na určité části systému (často ve formě SIGu viz dále), až po velké týmy, které mají svoje pole působnosti přesně popsané. Vše se ale děje svobodně právě v rámci komunity vývojářů a uživatelů, každý se může do již existující skupiny zapojit, případně založit vlastní, pro doposud nepokryté oblasti. Jednou z posledních takových skupin je Big Data SIG.

Fedora Engineering and Steering Committee (FESCo)

FESCo je komunitní orgán, skládající se z devíti volených členů a to vždy na jedno volební období, které odpovídá dvěma vydáním Fedory (tedy cca na 1 rok). Jak již z názvu vyplývá, jedná se o skupinu, která se stará především o řešení technických aspektů vývoje distribuce a směřování distribuce po technické stránce. FESCo tak schvaluje a koordinuje novou funkcionalitu obsaženou v nadcházejících vydáních, schvaluje nové sponzory skupiny „balíčkováčů“ a uděluje vyšší práva v rámci skupiny (takzvané proven packagery, kteří mohou zasahovat do většiny balíčků v distribuci) a případně řeší i technické spory.

Pro zajímavost v současné době drží tři místa zástupci brněnské pobočky Red Hatu a Fedora se tak dá díky zastoupení českých vývojářů (nejen v řídicích orgánech) považovat i za naši distribuci.

Fedora Board

Board je pak naopak orgán, který řídí směřování projektu po stránce strategické. Řeší směřování projektu jako Fedory jako například cílovou skupinu uživatelů. Schvaluje využití ochranné známky, moderuje konflikty osobní povahy (nově je k tomuto účelu zavedena Community Working Group).

Fedora Board se skládá z pěti volených členů (opět jako u FESCo na jeden rok) a čtyř dosazených členů. Jejich výběr má na starosti Fedora Project Lead (tzv. FPL), zastávající nejvyšší pozici ve Fedoře a reprezentující projekt navenek. Dle současných pravidel musí být FPL zaměstnancem společnosti Red Hat. Častým omylem je, že všichni dosazení členové jsou dosazení Red Hatem a pracují pro jeho zájmy a volení zástupci jsou zástupci komunity. Ne vždy tomu tak musí být (a většinou ani není, snahou je mít vyvážené zastoupení).

V současné době funkci FPL zastává Robyn Bergeron a autor příspěvku je voleným členem Fedora Board.

Special Interest Groups a ostatní týmy

Special Interest Groups, takzvané SIGy, jsou subkomunity v rámci projektu, které se zaměřují na určitý aspekt distribuce. Většinou se jedná o skupiny lidí

starajících se o desktopová prostředí (Desktop SIG, KDE SIG), ale také například o cloud atd.

- Design tým
- Release Engineering
- Infrastruktura
- Dokumentační tým
- Lokalizační tým
- Fedora QA (Quality Assistance)
- a další

Životní cyklus distribuce

Různé distribuce volí různý model vydávání, ale poslední dobou je běžným standardem vydávat nové verze jednou za cca šest měsíců. Tento model vychází z modelu vydávání hlavních open source desktopových prostředí. Distribuce tak mohou být synchronizované s upstreamy jejich výchozích prostředí. A Fedora využívá právě tohoto modelu.

K vydání nové verze Fedory tedy dochází cca dvakrát ročně s tím, že každé vydání je podporované po dobu jednoho roku (a přechodného období o délce jednoho měsíce). Současně jsou tak podporované dvě verze Fedory, označované jako F_n a F_{n-1} . Specialitou je pak tzv. Rawhide, neboli rolling release verze určená především k vývoji následujících verzí Fedory.

Po ukončení podpory (End Of Life) dané verze Fedory přestanou být vydávány veškeré aktualizace včetně bezpečnostních a nevyřešené chyby jsou v Bugzille uzavřené jako neopravené. Repozitáře jsou však nadále k dispozici.



Plánování vydání

Vývoj nové verze Fedory probíhá většinou paralelně s právě dokončovanou verzí a to vyhlášením termínu přijímání nových vlastností a funkcí, na kterých vývojáři pracují a budou pracovat. Proces má na starosti tzv. Feature Wrangler a ve spolupráci s FESCo jsou tyto návrhy validovány a schváleny pro danou verzi Fedory.

Pro schválené vlastnosti a novou funkcionalitu se pak aplikují dva milníky – jejich „zmražení“ (Feature Freeze), kdy musí být z velké části hotové a připravené pro testování (časově odpovídá větvení nové verze z Rawhide) a stoprocentní dokončení (100% Feature Complete). Pokud některá z navrhovaných vlastností nestihne jeden z těchto milníků, tak je vyřazena ze z jejich seznamu. Vývojáři na nich ovšem stále mohou pracovat.

Pro plánování následujících verzí (od Fedory 20) dochází k několika výrazným změnám. Celý proces se více zaměří více na komunikaci mezi vývojáři než na externí marketing. Dojde taky k rozdělení nových vlastností do více kategorií – tzv. self contained změn, tedy změn, které nezasahují do zbytku systému a celosystémových komplexních změn. Na ty se pak bude moci více zaměřit jak FESCo, tak vývojáři.

Důležité milníky vývoje

Fedora, jako většina softwarových projektů, vydává během vývojového cyklu Alpha a Beta vydání, které jsou následované finálním vydáním. Nejdříve se oddělí vývoj následující Fedory z Rawhide a dochází k takzvanému větvení (branching, probíhá v rámci interního verzovacího systému). Od téhle chvíle si již právě vyvíjená distribuce žije vlastním životem, jak v GITu pro správu balíčků, tak build systému atd. Před každým klíčovým milníkem dochází k zmražení změn (freeze), které půjdou do vydání a nahlášené chyby a jejich opravy musí projít přes review v rámci blocker bug procesu. Pouze schválené chyby tak mohou blokovat vydání nebo v případě neblokující chyby být začleněné do sestavení v rámci „freeze exception“. Správnost oprav a vůbec kvalita daného vydání se ověřuje pomocí tzv. Test Compose (testovacích sestavení) a ve chvíli, kdy jsou všechny blokující chyby opravené, tak se připravuje Release Candidate sestavení. O vydání distribuce se pak pro všechny tři milníky rozhoduje na tzv. Go/No-Go mítingu, kde se sejdou zástupci Fedora QA, release engineeringu, vývojářů a management – Fedora Project Lead a Fedora Program Manager. V případě, že dojde k rozhodnutí, že RC splňuje kritéria pro vydání, tak dojde k zrcadlení obrazů vydání na spolupracující servery. To trvá cca pět dní a poté je vydání ohlášeno jako veřejně dostupné (obvykle se plánuje na úterý). Pokud ne, dochází k posunutí vydání a většinou se posunuje o jeden týden.

Jak se zapojit do vývoje distribuce?

Jak již bylo zmíněné výše, ve Fedoře je spousta možností jak se zapojit do jejího vývoje. A to jak pro vývojáře a technicky orientované přispěvatele, tak i pro netechniky například v design týmu, lokalizačním týmu a marketingu. Komunikace většinou probíhá pomocí Internetu a to na IRC (většinou síť FreeNode)

a pomocí mailing listů. Pro nové přispěvatele je k dispozici praktický rozcestník na Wiki projektu.

Lidé pracující na Fedoře se taky pravidelně setkávají na různých komunitních akcích. Specifické pro Fedoru jsou FAD (Fedora Activity Day) a pak hlavní konference FUDCon (Fedora Users and Developers Conference). Ta se koná čtyřikrát ročně v různých částech světa. V současné době se plánuje nahrazení FUDConů jednou velkou konferencí Flock. Skupina ambasadůrů pak reprezentuje Fedoru na ostatních open source akcích, release párty a podobně.

3D TISK NA FIT ČVUT

Marek Žehra, Miro Hrončok

E-MAIL: MIRO@HRONCOK.CZ

Termín 3D tisk se pomalu, ale jistě stává součástí open-source světa. Jestli si představujete 3D brýle z kina a obyčejnou tiskárnu, rychle na to zapomeňte, protože 3D tiskárny RepRap tisknou opravdové trojrozměrné objekty.

Představte si stroj, do kterého pošlete počítačový 3D model navržený například v CADu, v Blenderu, nebo v jiném modelovacím nástroji, chvíli počkáte a dostanete skutečný objekt, který si můžete osahat v ruce. Představte si, že tento stroj se vám vejde na stůl, že si ho můžete sami postavit za méně než dvacet tisíc korun a že je kompletně open source – od firmwaru přes ovládací software až k hardwaru. Navíc tento stroj umí vyrobit části sama sebe. Není to sci-fi, je to realita.

3D tiskáren je celá řada, jsou postaveny na různých principech a technologiích. Některé stojí miliony, některé si postavíte v pracovním po večerech za zlomek této částky. Tento článek bude o těch druhých – o tiskárnách RepRap.

Jak to funguje

Modely vznikají aditivní cestou, takže tiskárna nevyřezává ani nic podobného. Nevzniká tak kromě zmetků žádný odpad a veškerý spotřebovaný materiál se nachází ve výsledném objektu. Určitě znáte tavnou lepicí pistoli. . .

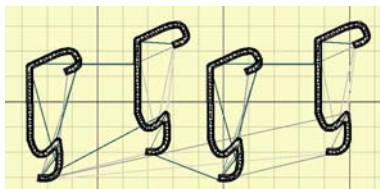
. . . přesně na jejím principu funguje 3D tisk na tiskárně RepRap. Místo lepicího silikonu se používá mnohem užší plastový drát namotaný na roli, který je nasouván do tiskové hlavy, kde se zahřeje a roztaví, nanese na podložku a zase ztuhne. To samozřejmě neděláte rukama, tiskovou hlavou a tiskovou plochou hýbe sada krokových motorů, která umožňuje pohyb hlavy vůči podložce po třech osách.

Tiskne se většinou z plastu, konkrétně z ABS, nebo z plastu podobného materiálu z kukuřičného škrobu, PLA – můžete se tak rozhodnout, že budete tisknout ekologicky. Vzhledem k principu tisku ale není problém tisknout z čokolady, co dokážete roztavit a nechat zase zatuhnout, existuje například tisk z čokolády (samozřejmě jde spíš o „tukoládu“, opravdová čokoláda by se spálila).

Máme tedy samohybnou tavnou pistoli, ze které vylézá místo lepidla plast – jak z toho ale udělat objekt podle počítačové reprezentace? Jednoduše, objekt



Obr. 1: Tavná lepicí pistole

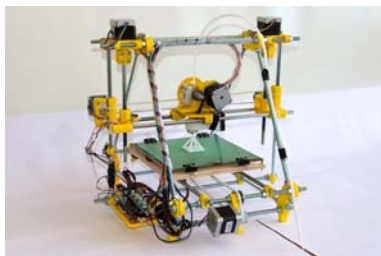


Obr. 2: Schéma průběhu tiskové hlavy jednou vrstvou

se před tiskem pomocí speciálního programu nařeže na dvourozměrné pláty. Tisková hlava pak nakreslí vždy jednu vrstvu a posune se o kousek výš, kde začne kreslit další. Výhoda je, že řezací program pozná, která část je vnitřní, a podle požadavku uživatele vyplní objekt jen částečně, například sítí nebo včelími pláty. Není tak dutý a jednoduše zničitelný, není ale ani plný a ušetří se tak materiál.

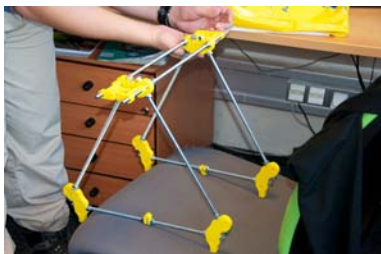
Jak to vypadá a kde to koupit

RepRap je open-source projekt, takže podobně jako u linuxových distribucí je jeho konkrétních variant mnoho. Většinou se skládají z nějaké konstrukce (závitové tyče pospojované plastovými součástkami vytisknutými na tiskárně), případně je celá tiskárna zasazena do nějaké bedýnky. Jednou z rozšířených variant tiskárny je například model Prusa Mendel, který má tvar na bok položeného hranolu s trojúhelníkovou podstavou. Tisková plocha se pohybuje po jedné ose, tisková hlava po zbývajících dvou.



Obr. 3: Tiskárna Prusa Mendel

Tento model tiskárny si postavíte za pár dní sami, budete k tomu potřebovat vytištěné součástky, kovové tyče (závitové a hladké), krokové motory, vyhřívanou desku a skleněnou desku na ni, trysku, odpory, řemeničky a řemeny, šrouby a matky, ložiska, dráty, stahovací pásy, termoizolační pásku, počítačový zdroj a Arduino nebo Sanguino(lolu) s USB kabelem. Většina materiálu na tiskárnu se dá koupit v Bauhausu, ale některé speciální věci jako materiál a tryska se rozhodně lépe shánějí v zahraničních e-shopech. Z nářadí kromě šroubováku a klíčů zmíním ještě pilku na železo, ať už ruční, nebo elektrickou, pilník a páječku, svěrák taky není na škodu.



Obr. 4: Stavba tiskárny



Obr. 5: Připravené díly na tiskárnu

Celkem se materiál na tiskárnu dá sehnat za necelých patnáct tisíc, ale počítejte, že pokud něco zničíte, může se cena vyšplhat i výš. Prodávají se i předpřipravené kity nebo dokonce hotové tiskárny, ale zde cena roste exponenciálně. Vytiskněte díly seženete od někoho, kdo tiskárnu už má, většinou se s ním stačí dohodnout například na tom, že až ji postavíte, vytisknete na oplátku něco vy pro něj.

Samo sebou existuje manuál na stavbu, přečtěte si ho dopředu, ať víte, jestli to zvládnete. Určitě je lepší pracovat ve skupině.

K čemu to je dobré

Mnoho lidí, kterým vysvětlím princip 3D tisku, se mě ale stejně ptá, k čemu to je dobré. 3D tisk se vyplatí při výrobě malého počtu objektů a návrhu prototypů nebo atypických součástí. Ne náhodou RepRap znamená replicating rapid prototyper (replikující se rychlý výrobce prototypů). Zde popíšu tři modelové situace, které ukážou výhodu 3D tisku nad běžnou výrobou.

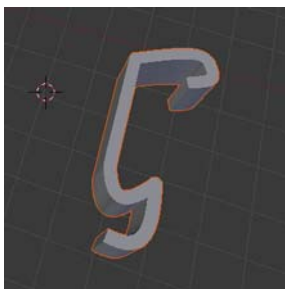
Modelová situace číslo jedna: Návrh a výroba píšťalek

Pokud máte v úmyslu vyvinout určitý typ plastové píšťalky, jistě si nevystačíte s počítačovou reprezentací objektu, těžko by se vám zjišťovalo, jestli dostatečně

píská. Svůj prototyp tedy musíte nechat vyrobit, pokud si ale objednáte plastový díl na míru, například z Číny, nedostanete dobrou cenu, pokud jich nechcete tisíce. 3D tiskem zde tedy velmi ušetříte. A nejen peníze, ale i čas, píšťalka se bude tisknout asi čtvrt hodiny, ale z továrny by píšťalky přijely nejdříve za týden. Když budete se svým produktem spokojeni, rozhodně se už 3D tiskárna nevyplatí, na velkovýrobu není určena.

Modelová situace číslo dvě: Atypická součástka

Chováme doma potkany, o kterých můžu s jistotou říct, že sežerou všechno, nedávno například okousaly plastové díly, které spojují kovovou část klece s plastovým dnem. Jedná se o poměrně atypický díl, který se nedá koupit normálně v obchodě, za předpokladu, že nechcete kupovat kompletně novou klec, řešení z drátu by ale vypadalo ohyzdně. Součástku jsem tedy namaloval v Inkscapu, prodloužil v Blenderu a vytiskl na tiskárně.



Obr. 6: Atypická součástka v Blenderu Obr. 7: Nový díl a rozkousaný originál

Modelová situace číslo tři: Personalizované objekty

Pokud máte v úmyslu vytvořit větší množství skoro stejných objektů s malou úpravou, je opět 3D tisk jasnou volbou. Ať už to jsou píšťalky s plastickými iniciálami návštěvníka konference, nebo klíny pod dveře s vyraženým číslem kanceláře. Stačí pak napsat skript, který z databáze a jednoho objektu vytvoří celé sady.

Vytisknout toho lze mnohem více: šperky, krabičky na různou elektroniku, hlavolamy, dekorativní předměty či trofeje atp.

Open source a open hardware

Koncept open-source softwaru čtenářům LinuxEXPRESu asi nemusím představovat. Pokud jste zde jen náhodu a nevíte, co to je, tak jen ve zkratce:



Obr. 8: Klíny pod dveře s číslem místnosti



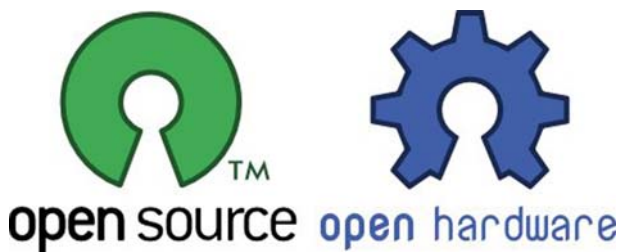
Obr. 9: Bowlingová trofej

Proprietární (closed-source) software patří nějakému jednotlivci nebo firmě. Vlastník ho zdarma nebo za úplatu poskytuje uživatelům ve spustitelné formě. Uživatel musí souhlasit s dodržováním podmínek, které jsou mnohdy omezující. Pokud chcete daný software upravit, nezbyvá vám nic jiného než kontaktovat dodavatele a doufat, že podobné služby poskytuje a že si je budete moci dovolit. Pokud chcete mít kontrolu nad tím, co vlastně program s vašimi daty dělá, musíte používat různé metody reverzního inženýrství, což často zakazují právě podmínky programu.

Na rozdíl od toho svobodný (open-source) software může používat každý k jakémukoli účelu. Ke spustitelné podobě dostane také zdrojové kódy, pokud by chtěl zjistit, jestli je program bezpečný, nebo kdyby ho chtěl upravit. Jako tvůrce softwaru můžete open-source software klidně prodávat, ačkoli se většinou nabízí zdarma a za úplatu jsou poskytovány související služby. Příkladem open-source softwaru je operační systém Linux, kancelářský balík OpenOffice.org, redakční systém WordPress, internetový prohlížeč Firefox nebo multimediální přehrávač VLC.

Podobně jako open-source software existuje i open-source hardware (nebo jen open hardware). U hardwaru tento pojem značí, že je k produktu dodávána kompletní dokumentace, veškerá schémata, seznam materiálů, často i návod na sestavení vlastního zařízení. Bývá zvykem, že v open-source zařízení je navíc také open-source software, ačkoli to není nutnou podmínkou.

Otevřeným hardwarem se zabývá například web Openmobility.cz. Známými kousky jsou kromě tiskárny RepRap například mobil Neo FreeRunner nebo programovatelná platforma Arduino.



Obr. 10: Loga open source a open hardwaru

3D tiskárna na FIT ČVUT

Na Fakultě informačních technologií Českého vysokého učení v Praze existuje laboratoř 3D tisku, kde mohou studenti ve volném čase tiskárnu stavět, tisknout na ní nebo ji vylepšovat. V současné době je plánován i předmět zabývající se 3D tiskem. Je to ideální místo, kde si můžete nechat vytisknout součástky. Studenti z Fakulty architektury si zde například nechávají tisknout modely pro své diplomové práce.

V rámci PR aktivit fakulty tiskárna často vyjíždí nejen na linuxové konference (InstallFest, Software Freedom Day), ale také na hudební festivaly (Majáles, Sázavafest) nebo technické výstavy (Vynálezci a vynálezy, Muzejní noc). S tiskárnou se tak mohou setkat i „běžní smrtelníci“, kteří by o její existenci jinak neměli ani ponětí.



Obr. 11: 3D tiskárna na výstavě Vynálezci a vynálezy

Článek je publikován na LinuxEXPRESu.

<http://www.linuxexpres.cz/hardware/3d-tisk>

WRAPSIX, ANEB NEBOJME SE NAT64

Michal Zima

E-MAIL: ZIMA@WRAPSIX.CZ

NAT64 je přechodový mechanismus, který navazuje na zavržený NAT-PT. Základní myšlenkou bylo doplnit do sítě chybějící prvek, který by mezi protokoly IPv4 a IPv6 položil most přes jejich vzájemnou nekompatibilitu tím, že bude pakety mezi těmito dvěma protokoly překládat.

Z rozdílných velikostí jejich adresních prostorů plyne zřejmé omezení: nemůže libovolně komunikovat každý s každým – například uzel z IPv4 sítě nebude moci komunikovat s libovolným IPv6 uzlem jednoduše proto, že nemá dostatek IPv4 adres, do kterých by mohl jejich IPv6 adresy mapovat. Rozlišujeme proto dva typy NAT64 – stavový a bezstavový –, jejichž účel se mírně liší.

Stavový NAT64 vytváří most mezi uzly v IPv6 síti, ze které chtějí chtět komunikovat s IPv4 uzly „tam venku“. Stejně jako v případě maškarády u IPv4 se používá pro skupinu šestkových uzlů jen jedna čtyřková adresa. Oproti tomu bezstavový NAT64 vyžaduje pro každý komunikující šestkový uzel samostatnou čtyřkovou adresu pro mapování 1 : 1. Na druhé straně to umožňuje bezproblémovou iniciaci komunikace ve směru z IPv4 sítě. Lze tak třeba mít servery umístěné na ryze IPv6 infrastruktuře, a přesto umožňovat IPv4 uzlům, aby s nimi zcela libovolně navazovaly komunikaci.

Problém, který klient za NAT64 řeší jako první, je, kde vzít IPv6 adresu uzlu, který ji nemá. IPv4 adresy získává z DNS nebo je zná. Druhá situace se v rámci NAT64 běžně neřeší, avšak té první se dá velmi pěkně využít. Konkrétně se o to stará koncept zvaný DNS64. Jakmile se klient zeptá na záznam s IPv6 adresou nějakého doménového jména, DNS64 server vrátí buď nativní IPv6 adresu, kterou dané jméno skutečně má ve svých DNS záznamech, nebo pro něj jednu vyrobí z jeho IPv4 adresy.

Tato adresa se syntetizuje z vyhrazeného IPv6 prefixu pro NAT64 – standard pro tento účel doporučuje oficiální vyhrazený prefix 64:ff9b::/96 – a samotné IPv4 adresy. Například pro jméno europa.cz s IPv4 adresou 147.228.60.10 bude mapovanou IPv6 adresou adresa 64:ff9b::93e4:3c0a.

Pro uzly v IPv6 síti tak budou dostupná všechna jména, která by byla dostupná i z IPv4 sítě.

Prefix 64:ff9b::/96 má ještě jednu pěknou vlastnost, a to že je neutrální vzhledem ke kontrolním součtům, takže bezstavový NAT64 může díky němu ušetřit výkon při překladu. Stavový ne, protože ten navíc ještě mění čísla portů v hlavičkách protokolů čtvrté vrstvy.

NAT64 si umí poradit s veškerými specifiky v rozdílnosti obou protokolů. Zajímavá je zejména fragmentace, protože každá verze implementuje odlišný přístup. V IPv6 smí provádět fragmentaci pouze odesílatel paketu. Zda fragmentovat či ne rozhoduje na základě průzkumu cesty k cíli – pokud se mu některý velký paket vrátí s chybou, že je moc velký, pošle ho znovu, ale již rozdělený na více menších, a další pakety již bude na danou adresu posílat rovnou malé. Oproti tomu v IPv4 může nastat fragmentace prakticky na každém uzlu, kterým daný paket prochází, a každý fragment může dorazit k cíli úplně jinou cestou. Pokud se tomu bude chtít odesílatel vyhnout, může si vynutit nefragmentování paketu nastavením bitu „Don't fragment“ na jedničku. Zacházení s paketem se pak prakticky nijak neliší od chování v IPv6.

NAT64 se snaží chovat pro obě strany co nejpřirozeněji. Ve směru z IPv6 do IPv4 proto automaticky nastavuje „Don't fragment“ bit, takže i vně IPv6 sítě se na paket aplikuje stejná fragmentační politika – tedy nefragmentuje se, a pokud je paket větší než MTU některé linky, tak se vrátí patřičná chybová zpráva. V opačném směru se NAT64 chová jako poslední IPv4 uzel a jako první IPv6 uzel v cestě – z obou pozic může paket fragmentovat, což také udělá, pokud je MTU v IPv6 síti menší než velikost paketu. Pokud není explicitně řečeno jinak, konzervativně se za MTU považuje 1280 B, což je podle standardu nejmenší hodnota MTU v IPv6, takže by paket měl vždy dojít až k cíli.

Podobně transparentně se chová k veškerému provozu na třetí a čtvrté vrstvě. Překládají se i vnitřní pakety v chybových ICMP zprávách, takže příjemce ani odesílatel netuší, že druhá strana používá jiný protokol, a mohou chyby korektně zpracovat.

Takto tedy NAT64 funguje. Důvodů, proč ho použít například místo rozšířenějšího dual-stacku, je řada. Tím hlavním je definitivní přechod na IPv6. Zatímco dual-stack představuje dvoufázový přechod – první fází je zavedení IPv6 vedle IPv4, druhou pak odstranění IPv4 v okamžiku jeho zanedbatelného využití –, tak NAT64 zahrnuje pouze jednu fázi, čímž přechod na IPv6 zjednodušuje – situace se v síti vyřeší jednou a nebude potřeba znovu a znovu řešit případné problémy plynoucí ze existence a správy dvou sítí v jedné. Zjednodušení infrastruktury sítě také znamená méně míst, kde se může něco porouchat. NAT64 navíc nevyžaduje kromě podpory IPv6 žádné jiné investice, na rozdíl od dual-stack lite, který potřebuje podporu na směrovačích na hranici koncových podsítí.

Částečnou nevýhodou může být, že pro kompletní přechod z IPv4 na IPv6 v rámci nasazování NAT64 bude nutné, aby všechny uzly v síti IPv6 podporovaly. Především tedy koncoví klienti – zde povede přítomnost velmi starého zařízení či operačního systému k úplnému odříznutí uzlu od sítě. Je sice možné

IPv4 zachovat, ztrácí se pak však hlavní výhoda nasazení NAT64. Další stinnou stránkou je v současnosti malá prověřenost praxí – je známých jen několik ISP, kteří NAT64 do svých sítí nasadili a používají ho.

NAT64 prakticky

Provoz NAT64 brány není nikterak náročný. Pro menší sítě se špičkovým provozem do několika desítek tisíc paketů za vteřinu stačí i naprosto běžný PC hardware a jedno síťové rozhraní. Pro provoz kolem sta tisíc paketů za vteřinu (v jednom směru) je již nutné mít výkonnější server se dvěma síťovými rozhraními. Větší nároky pak splní dedikované stroje, které jsou pro NAT64 speciálně optimalizované.

DNS64 je již několik let dostupné v rozšířeném DNS serveru BIND, a to od verze 9.8.0, takže pokud BIND používáte a alespoň čas od času aktualizujete, s největší pravděpodobností bude stačit, když přidáte do jeho konfiguračního souboru jeden nový řádek „dns64 64:ff9b::/96 {}“;“ do sekce „options“. Více není potřeba.

V praxi provozují WrapSix, což jedna z dostupných open-source NAT64 bran. Za necelý rok provozu obstál co do stability, výkonu i hardwarové nenáročnosti. Zprovoznění je velmi jednoduché, protože o většinu práce se postará konzervativní výchozí nastavení – nutné je pouze nastavit vyhrazenou IPv4 adresu (tedy takovou, kterou nemá přiřazenu ani systém, na kterém WrapSix běží) a zařídit, aby síť směrovala používaný NAT64 rozsah (standardně 64:ff9b::/96) na server s bránou. Hotovo.

Jakmile však odstraníte ze sítě IPv4, dříve či později zjistíte, že ne všechny aplikace fungují – problém budete mít například s překládaným FTP či s online hrami, u kterých se nebudete moci připojit na žádný herní server v IPv4. Druhý zmíněný problém vyplývá z faktu, že překládán je jen provoz, který byl mířen na nějaké jméno, tedy DNS64 pro ně mohl vytvořit mapovanou IPv6 adresu.

Tento problém jsem ve WrapSixu vyřešil vytvořením speciálního klientského programu – „WrapSix klienta“ –, který na koncovém počítači simuluje přítomnost IPv4, ve skutečnosti však takový provoz překládá do IPv6 a posílá přímo na NAT64 bránu. Chová se tedy v podstatě jako malý inverzní NAT64. Doplnuje chybějící kousek pro kompatibilitu IPv4 s IPv6, takže nyní mohou v IPv6 síti fungovat i ty aplikace, které IPv6 ještě vůbec nepodporují. Bez tohoto elementu se NAT64 bez IPv4 nedá univerzálně nasadit, což zbytečně ztenčuje spektrum přechodových mechanismů, mezi kterými si může správce sítě vybírat. WrapSix klient je samozřejmě nezávislý na konkrétní implementaci NAT64 a je možné ho použít zcela univerzálně.

K úplné dokonalosti tedy schází již jen podpora překladu aplikačních protokolů – FTP, SIP apod. To znamená těch protokolů, které přenášejí IP adresy

komunikujících uzlů v těle TCP a UDP paketů. Pro překlad FTP je dostupné i RFC, které specifikuje, jak překlad provádět. Implementací však je minimum – zpravidla se autoři NAT64 bran brání tomu, aby zasahovali někam hlouběji než jen do základních hlaviček protokolů TCP, UDP a ICMP. Osobně se však domnívám, že bez této úrovně překladu se omezuje univerzálnost nasazení NAT64. Ve WrapSixu se proto jeho podpora brzy objeví.

Uživatele nezajímá, proč mu to či ono nefunguje – on chce funkční službu a pokud ji nemá, odchází ke konkurenci. Pokud nebude NAT64 tuto podmínku splňovat, nelze logicky počítat s jeho širokým uplatněním. Základní ideou WrapSixu je být prostředkem k bezproblémovému nasazení NAT64 – aby uživatel pokud možno nezpozoroval, že se v síti něco změnilo. Teprve pak totiž plní NAT64 beze zbytku svůj účel.

NETWORKMANAGER PRO SERVERY A VIRTUALIZACI

Pavel Šimerda

E-MAIL: PAVEL.SIMERDA@NETINSTALL.CZ

Síťová konfigurace se na Linuxu řeší různými způsoby. Jádro nabízí několik co do funkcionality překrývajících se rozhraní. Netlink nabízí správu síťových rozhraní, síťových adres a směrovacích tabulek. Některé specifické informace jako podpora VLAN nebo podpora detekce L1 spojení jsou dostupné přes `ioctl`. Detailnější konfigurace specifických typů rozhraní jako bridge nebo bond je možná pomocí souborů v `/sys/devices`. Pro některá obecnější nastavení se musí do `/proc/sys`.

Konfigurační nástroje

Nad celým tímto nepořádkem lze používat kupu specializovaných nástrojů, které se ovládají na příkazové řádce nebo ve skriptech. Skripty si může člověk psát sám nebo může použít nějaké hotové řešení, typicky svázané s konkrétní rodinou distribucí a postavené na oddělené konfiguraci jednotlivých síťových připojení. Řešení se síťovými skripty je celá řada a nejvíce se liší právě konfiguračním formátem.

V mnoha případech se uživatelé a provozovatelé nespokojí se statickou konfigurací a je potřeba dynamicky reagovat na to, co se děje na síti. Může se jednat o reakci na vypojení či zapojení kabelu. Nebo to může být konfigurace adres pomocí dynamického protokolu DHCP či NDP. Dalším podobným případem je výměna šifrovacích klíčů u zabezpečených tunelů a bezdrátových sítí.

Každý z těchto dynamických protokolů musí obstarávat nějaký démon, který je schopný plánovat akce a reagovat na vnější události jako jsou příchozí pakety nebo informace z fyzických síťových rozhraní. I kdyby skutečně každé jedné takové úloze odpovídal jeden jediný démon, někde je potřeba agregovat výsledky těchto démonů a skládat výslednou konfiguraci. Zářným příkladem je výběr výchozí brány a konfigurace DNS.

Síťové skripty dokážou ve spojení s dílčími démony vyrobit jednotlivé fragmenty konfigurace. Ty se pak musí pomocí nějaké politiky poskládat do výsledné systémové konfigurace. V případě DNS v tom pomáhají nástroje typu `resolvconf`, směrování se řeší o něco hůře. Jako jednodušší se dle mého názoru jeví centrální démon, který řeší jak směrování, tak DNS.

Takový centrální démon umí s výhodou použít přístup k uživatelské konfiguraci sítě a komunikaci se specializovanými démony nebo knihovnami. Navíc má možnost upozorňovat na zásahy způsobené jinými nástroji a pomoci tak ladit neočekávané síťové problémy.

Pořád se můžete ptát, kde je ta výhoda pro servery, které nevyžadují žádnou formu dynamické konfigurace kromě toho, co dokáže obstarat sám kernel. Žádné zázraky nečekejte. Pokud jste zvyklí na nějaký způsob práce a vyhovuje vám, těžko vás přesvědčím k výkřikům nadšení. Nicméně koncept centrálního síťového démona skutečně nabízí velmi návykové vlastnosti. Když si na ně jednou zvyknete, nebudete už chtít udělat krok zpět.

Pokud jste už alespoň částečně přijali myšlenku centrálního démona, má smysl se zabývat tím, jaké kompetence se mu přidělí. Já osobně inklinuju k tomu, aby se staral o veškerou běhovou i diskovou konfiguraci a zbytku systému poskytoval konfigurační API. Nad ním se pak dobře staví další projekty jako vzdálená správa nebo virtualizace.

NetworkManager

Mně osobně zaujaly dva projekty, které jsou postavené podle výše uvedených kritérií, a to NetworkManager a OpenWRT netifd. Na prvním z nich necelý rok aktivně pracuju a docela jsem se zorientoval v jeho návrhu. Během jeho přepracovávání NetworkManageru směrem k serverovým systémům s virtualizací jsem došel k překvapivému závěru. Ač je systém od začátku navržený spíše pro laptopy a desktopové systémy, jeho architektura tím není zdaleka tak poznamenaná, jak by člověk očekával.

Nedávno vydaný NetworkManager 0.9.8 je takovým nesmělým krokem do serverového světa. Toto vydání obsahuje kupu vylepšení týkajících se softwarových rozhraní. Nově podporuje rozhraní typu bridge. Spolu s bondy a vlany tvoří trojici softwarových rozhraní používaných virtualizačními nástroji. Už jen to vzbudilo zájem mezi dalšími projekty, které potřebují zasahovat do konfigurace sítě.

Vydáním 0.9.8 to samozřejmě nekončí. Spíše je to taková první vlaštovka, která má za cíl ukázat, že se něco děje. Další vývoj se zaměřuje na pročištění klíčových částí architektury a zajištění spousty vlastností, které lidé od serverů považují za samozřejmost. Jako příklad poslouží oddělení běhové konfigurace od té uložené na disku.

Integrace

Aby mohl NetworkManager konfiguraci správě poskládat, potřebuje znát všechny informace, které mají mít na výslednou konfiguraci vliv. A naopak potřebuje,

aby mu ostatní nástroje co nejméně zasahovaly do výsledné konfigurace. Jsou věci, které se dají zvládnout, jako například přidání statické adresy na rozhraní.

Do budoucna by měl NetworkManager umět automaticky zahrnovat změny adres a směrovacích záznamů do běhové konfigurace. O něco horší případ je prioritizace směrovacích záznamů a konfigurace DNS. V mnoha případech je klasický `/etc/resolv.conf` nepoužitelný. Neumožňuje svázat záznamy s konkrétním rozhraním a především je omezit na konkrétní DNS zóny, které jsou přes dané rozhraní dostupné (split DNS).

Obecně jsou tak dvě cesty jak dosáhnout komplikovanějších síťových konfigurací. Buď musí každý nástroj umožňovat detailní nastavení priorit a různých speciálních výjimek v konfiguraci, nebo lze delegovat rozhodnutí na centrálního démona. Ač jsou obě možnosti validními řešeními, mně stále přijde čistší, jednodušší a přehlednější ta druhá.

SPRÁVA SÍTÍ POMOCÍ OPENLMI

Radek Nováček

E-MAIL: RNOVACEK@REDHAT.COM

Úvod

S rozvojem informačních technologií je stále vyšší poptávka po správcích počítačových systémů, ať už serverů nebo jednotlivých pracovních stanic. Nakonfigurování systému je však nelehký úkol, který vyžaduje, aby administrátor měl detailní znalosti o systému i aplikacích, které na něm běží. Mezi jeho úkoly patří zajistit vysokou dostupnost poskytovaných služeb, zabezpečení proti útokům i nastavení jednotlivých programů. Po odbornících, kteří tyto úkoly zvládnou, je velká poptávka a jsou i odpovídajícím způsobem nároční finančně. Mnoho firem proto vyžaduje, aby systémy byly jednodušší, snadněji konfigurovatelné a umožnily i méně zkušeným administrátorům systém správně nastavit a zabezpečit. Z tohoto důvodu je zde neustálý tlak na vytváření nových konfiguračních nástrojů. Pro usnadnění vytváření těchto nástrojů je vhodné mít k dispozici rozhraní, které umožní jednotným způsobem přistoupit k nastavení systému a upravit jej.

Projekt OpenLMI

Projekt OpenLMI si klade za cíl vytvořit jednotné rozhraní a další infrastrukturu pro správu linuxových systémů za použití otevřených standardů. Základním stavebním kamenem je sada technologií nazvaná WBEM (Web-Based Enterprise Management), která umožňuje vzdáleně zjistit spravovatelné prvky na vzdálených počítačích, číst jejich stav a nastavovat je. WBEM definuje všechny části správy počítačových systémů, od komunikačního protokolu až po model spravovaného systému a dostupné operace. Tento model je popsán standardem CIM (Common Information Model). Vývoj všech těchto standardů zajišťuje organizace DMTF (Distributed Management Task Force).

CIM se skládá ze dvou částí, specifikace a samotného modelu. Specifikace určuje, jakým způsobem bude model definován. Objektový model popisuje jednotlivé spravované prvky, které reprezentovány třídami. Vztahy mezi prvky se definují pomocí asociací.

Model samotný není dostačující pro popis chování daných objektů, proto jsou k dispozici také profily. Profil definuje, které vlastnosti objektů musí být definované, jak se dané objekty mají používat, atd. Obvykle tyto profily také obsahují UML schéma s popisem objektů a jejich vztahů pro dané případy použití.

V praxi tento systém funguje následovně: na spravovaném systému běží tzv. CIMOM (Common Information Model Object Manager), což je síťový server, který zpracovává požadavky od klientů. Tyto požadavky jsou předány Providerům, což jsou softwarové moduly zaměřené správu určité části (třídy) zdrojů daného počítače.

V rámci projektu OpenLMI se vyvíjí několik Providerů pro základní oblasti správy systému, například pro správu sítí, diskových úložišť, popis hardwaru, instalaci softwaru či správu systémových služeb.

Síťová část projektu OpenLMI

Správa a monitorování síťových rozhraní je nedílnou součástí mnoha nástrojů pro správu počítačových systémů. Součástí projektu OpenLMI je modul openlmi-networking, což je Provider, který tuto funkcionalitu zajišťuje. Pro interakci se spravovaným systémem je využit NetworkManager, což je systémový démon (spolu s dalšími nástroji) pro správu sítí. Provider tedy tvoří vrstvu mezi rozhraním NetworkManageru a CIM modelem. Provider využívá standardních tříd CIM modelu a s mírnými odchylkami i DMTF profilu DSP1116 (IP Configuration Profile).

V rámci projektu je v plánu implementovat mimo základní funkcionality (Ethernet, IPv4, IPv6) také některé pokročilejší funkce jako například bridging a bonding. V současné době (duben 2013) je funkční možnost zobrazení aktuální síťové konfigurace, nastavení síťových profilů a základní podpora pro bonding.

PASIVNÍ ÚTOKY NA GSM

Jan Hrach

E-MAIL: JENDA@HRACH.EU

GSM je síť široce používaná po celém světě. Povaha přenášených dat, hovorů a SMS vedla již od 90. let minulého století k výzkumu a vývoji mnoha útoků. První byly relativně primitivní a především snadno odhalitelné, v posledních letech se však rozšiřují sofistikované, plně pasivní útoky.

Útoky typu DoS

Pokud útočníkovi stačí způsobit odepření služby, má to celkem jednoduché. GSM je rádiová síť, takže z principu lze zarušit. Levné rušičky lze objednat z Číny, jejich stavba ze šuplíkových součástek také není příliš složitá. Nevýhodou je snadné odhalení.

Mnohem zajímavější je způsobení DoS odesláním malého množství „vhodných“ dat někam do sítě. Softwarová stránka implementace GSM je všelijaká a když spustíte na mobil nějaký běžný fuzzer (věc, která zkouší posílat různými způsoby rozbitá data a čeká, jestli na druhé straně spadne parser), najdete spoustu chyb. Přečtěte si například o projektu SMS-o-death¹ (záznam přednášky²), který zjistil, že spousta mobilů lze zablokovat SMSkou obsahující binární balast.

Je zřejmé, že na bezdrátovém principu není proto vhodné stavět kritické technologie, jako například zabezpečení budov posílající SMS notifikaci a volající policii přes GSM. Leč mnozí tak činí.

Dále se již budeme věnovat bezpečnosti přenášených dat.

Prostě si to poslechni

Po telefonu si lidé říkají všelijaké citlivé informace. Bohužel u většiny dnes provozovaných sítí tečou někde u provozovatele data nešifrovaně. U GSM je šifrováno spojení mezi mobilem a BTS, ale infrastruktura operátora jsou již data dostupná.

¹<http://events.ccc.de/congress/2010/Fahrplan/events/4060.en.html>

²<http://www.youtube.com/watch?v=J-IUL3E-uPc>

Odposlechnout je může neposlušný zaměstnanec nebo složka státní moci (samozřejmě pouze na soudní příkaz:-). Již z tohoto důvodu bychom si měli dávat pozor, co telefonu svěřujeme.

IMSI catcher je vlastně Man-In-the-Middle

Pokud útočník nemá přístup do infrastruktury operátora a přesto by chtěl poslouchat, musí nějakým způsobem obejít či zlomit šifru, kterou je komunikace vzduchem chráněna.

Jedním z prvních útoků, které se objevily, byl tzv. IMSI catcher. Využívá toho, že mobilní telefon kryptograficky neověřuje identitu sítě, ke které se přihlašuje. Každý operátor v každé zemi má unikátní ID skládající se z MCC (Mobile Country Code, pro ČR 230) a MNC (Mobile Network Code, v ČR 01-03 pro známou trojku a 98 pro jednoho méně známého³) a tyto informace jsou všemi BTS pravidelně ohlašovány do sítě. Mobil si ze SIM karty zjistí, kterému operátorovi patří, a k jeho nejsilnější BTS se přihlásí. Pokud žádnou „rodinnou“ BTS nenajde (typicky v zahraničí), zkusí si domluvit roaming.

Co se ale stane, pokud si spustíme vlastní BTS a nastavíme dvojici MCC a MNC na tu, kterou už nějaký operátor používá? Pokud zařídíme, aby obě našeho útoku měla dostatečně silný signál (například směrovou anténou), ochotně se její mobil přihlásí na naši BTS. A protože šifrování přenosu iniciuje BTS, můžeme mobilu říct, aby šifrování vypnul, a pak si hovor v klidu poslechnout.

Problémy jsou dva: za prvé nefungují příchozí hovory a za druhé lze takový útok poměrně snadno odhalit. Překvapující je, že to nikdo nedělá. Totiž – indicií je několik:

- Vypnulo se mi šifrování. Ke sjednání symetrického šifrovacího klíče Kc (session key) je potřeba sdílené tajemství mezi operátorem a SIM kartou. IMSI catcher toto tajemství nezná, šifrování proto musí vypnout. Telefon má podle specifikace na absenci šifrování upozornit, ovšem v přednášce GSM: SRSLY?⁴ se tvrdí, že toto varování lze potlačit nastavením jednoho konfiguračního bitu na SIM kartě a světe div se, většina operátorů to má potlačeno.
- V okolí se objevila nová mimořádně silná BTS.
- Nefungují mi příchozí hovory (to už prý mají nejnovější implementace IMSI catcherů nějak vyřešené, ale vůbec netuším, jak).

Existuje projekt IMSI catcher catcher⁵, který se snaží podle těchto indicií útok odhalit. Shodou okolností je implementován nad telefony Osmocom⁶.

³<http://cs.wikipedia.org/wiki/GSM-R>

⁴<http://events.ccc.de/congress/2009/Fahrplan/events/3654.en.html>

⁵<https://opensource.srlabs.de/projects/catcher>

⁶[/clanky/na-gsm-s-osmocombb](http://clanky/na-gsm-s-osmocombb)

Zařízení vlastní například různé složky státní moci⁷ (zajímavý dodavatel⁸). Ostatně i ta⁹ slavná¹⁰ Agáta¹¹ je „jenom“ takový naspeedovaný IMSI catcher.

Opensource IMSI catcher se dá postavit nad OpenBTS¹². Je k tomu potřeba USRP a spousta času.

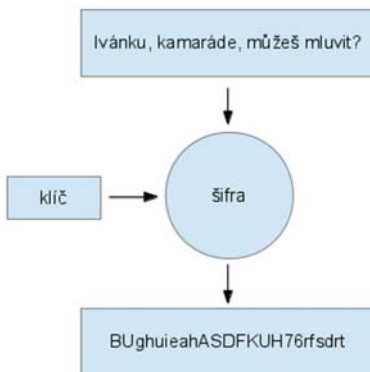
Pasivní sniffing je nejsofistikovanější typ útoku

Především hrozba odhalení vedla k vývoji dalších typů útoků. Pro jejich pochopení ale budeme muset tušit něco málo o symetrických šifrách.

Intermezzo o blokových a proudových šifrách

Symetrická šifra je funkce, která vezme nějaký tajný klíč a data, která chceme zašifrovat, vyrobí z toho nějaký nečitelný blob a pošle to. Na druhé straně se vezme *ten stejný* klíč a blob se jím dešifruje. Příklady symetrických šifer jsou AES, Blowfish, DES, A5/1, RC-4.

Některé symetrické šifry jsou blokové. Fungují tak, že vezmou blok dat (řádově stovky bitů) a klíč a vyprodukovat zašifrovaný blok dat. Blokové šifry jsou například AES, Blowfish, DES.



Obr. 1

⁷<http://vz.statnisprava.cz/?sid=0&pg=vz&ecvz=60021365>

⁸<http://www.bull.cz/about-bull/index.html?page=special>

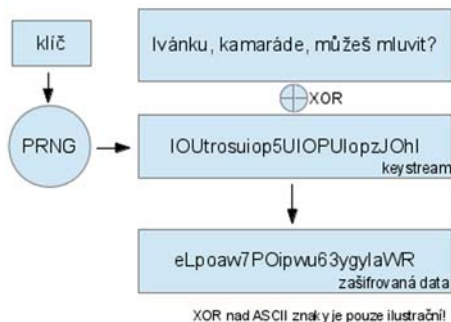
⁹http://mobil.idnes.cz/mame-manual-k-odposlouchavaci-agate-jmenuje-se-armada-a-je-z-polska-p98-/mob_tech.aspx?c=A120521_172742_mob_tech_vok

¹⁰<http://www.irucz.ru/cz/zpravy/1-/102000000000-ceska-republika/10206-/-/107-korupce-krimi-spioni/4896-agata-tajna-superzbran-odposlechy-telefonu-jsou-technicky-nenarodne/>

¹¹<http://www.dfens-cz.com/view.php?cisloclanku=2008060805>

¹²<http://wush.net/trac/rangepublic>

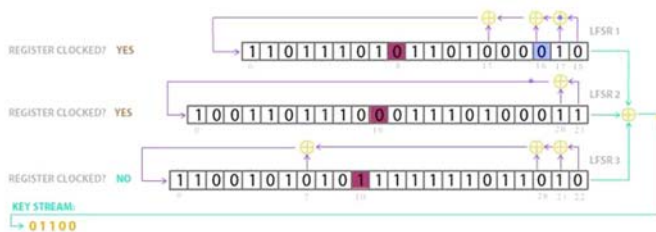
Jiné symetrické šifry jsou proudové. Pomocí klíče se inicializuje *generátor pseudonáhodných čísel* a tímto generátorem se vygeneruje potřebné množství (proud) pseudonáhodných dat, kterým se říká keystream. Keystream se potom zXORuje s daty, která chceme zašifrovat. Příjemce si také inicializuje svůj pseudonáhodný generátor, nechá ho vygenerovat keystream a zXORuje s ním přijatá data. Vypadne na něj původní text. Proudové šifry (stream ciphers) jsou například RC-4 nebo A5/1 používaná v GSM.



Obr. 2

Generátor keystreamu v A5/1 jsou tři posuvné registry, do kterých je na začátku nasypan klíč, a pak se začínou různě posouvat a míchat, výstupy z nich se vzájemně XORují a někudy z toho začínou padat pseudonáhodná data, keystream. Můžete se podívat na povedenou animaci principu na YouTube¹³.

STEP 5
CYCLE 6 / 228



Obr. 3

¹³<https://www.youtube.com/watch?v=LgZAI3DdUA4>

Šifra má tedy cosi jako vnitřní stav definovaný hodnotami proměnných v generátoru. Když ho získáte, můžete ho nasypat do svého generátoru, vygenerovat si stejný keystream a dešifrovat přenášenou zprávu. A5/1 lze navíc „backclockovat“¹⁴ – s o něco vyššími nároky na výkon lze obrátit směr, kterým se posouvají bity v posuvných registrech, a po chvíli počítání získat i počáteční stav šifry a z něj klíč. Tedy zjištěním vnitřního stavu v jakémkoli okamžiku používání šifry jste schopni dešifrovat celou komunikaci.

Můžete si spočítat tabulku:

Vnitřní stav	Keystream
0x0000000000000000	0xabcdef12345678
0x0000000000000001	0x54632221afed03
0x0000000000000002	0x456dcd562b980e

Když uvidíte ve vzduchu nějaká data a dokážete z nich extrahovat keystream, podíváte se do tabulky, zjistíte vnitřní stav šifry a máte vyhráno. Bohužel taková tabulka je poněkud velká. Triviálně 2^{64} možných stavů $\times 64$ bitů keystreamu na stav, přičemž se ukazuje¹⁵, že keyspace je jenom nějakých 2^{61} , ale i tak je to v řádech EB (exabajt, tedy milion terabajtů).

Naštěstí lze použít time-memory tradeoff. Pomocí jistého triku si spočítáte menší tabulku, která bude obsahovat pouze vybrané vnitřní stavy šifry, ale zároveň bude obsahovat informace, které vám umožní chybějící data rychle dopočítat v okamžiku, kdy je potřebujete.

Velmi hrubě řečeno to funguje tak, že si zvolíte nějaký počáteční bod, použijete ho jako tajný stav pro A5/1 a vyrobíte keystream. Ten vhodně upravíte a použijete jako tajný stav pro další A5/1 a tak dále, až se dostanete do *distinguished pointu*. Ten může být definován třeba tak, že na konci je určitý počet nul. Počáteční bod a distinguished point uložíte do tabulky. Opakujete tak dlouho, až vám dojde místo na disku.

Počáteční tajný stav	Distinguished point (po mnoha iteracích)
0x0000000000000000	0xcd547853000000
0x0000000000000001	0x4512ebca000000
0x0000000000000002	0x3249bdcc000000

Když potom získáte nějaký keystream, použijete ho jako tajný stav a počítáte tak dlouho, až se dostanete do distinguished pointu. Ten si najdete v tabulce, podíváte se, jaký počáteční tajný stav na něj vedl, a dopočítáte zbytek řetězu. Víte, že článek řetězu těsně před vaším keystreamem je tajný stav, který vás zajímá.

¹⁴<http://web.archive.org/web/20100710081708/http://reflexor.com/trac/a51/wiki/BackclockA51>

¹⁵<http://web.archive.org/web/20100710081708/http://reflexor.com/trac/a51/wiki/BackclockA51>

Pomocí dalších optimalizací (nad rámec článku) se dá zmenšit velikost tabulek zabráněním kolizí (merge), kdy máte nějaká data zbytečně víckrát. Hezky je to popsáno v přednášce GSM: SRSly?¹⁶ Podívejte se také na projektovou wiki¹⁷. Útok momentálně implementuje projekt Kraken¹⁸. Velmi cenným zdrojem při studiu samotné kryptografické fáze útoku mi byly zdrojáky Krakena (bohužel nejsou příliš komentované) a archiv mailinglistu vývojářů projektu¹⁹.

Na čtyřjádrovém Xeonu s 8 GiB RAM, grafikou Radeon HD 5900 a 2 TB polem z rychlých 15kRPM SCSI disků lze cracknout A5/1 v řádu minut. Se SSD disky a čtyřmi grafikami se dá dostat na sekundy.

Skákání po kanálech

Když přenášíte nějaká data vzduchem a přijímač není v ideálních podmínkách, například je mezi železobetonovými domy, vlny se různě lámou a odrážejí, v některých místech se sečtou a v některých vyruší, na některých frekvencích se objevuje rušení atd. Proto je výhodné používat channel hopping – při přenosu se velmi rychle mění frekvence, na které se vysílá, a doufá se, že i když jeden kanál bude zarušený, ostatní budou fungovat dobře a dojdou alespoň nějaká data (z těch potom lze pomocí rozličných error correction a parit dopočítat zbytek). Případnému útočníkovi, který se snaží komunikaci odposlechnout, tohle ale může způsobit komplikace, protože se musí přeladovat přesně stejně jako vysílač.

Poslech signálních dat

Můžeme se zaposlouchat, co se kolem nás ve vzduchu děje. Naladíme se na BEACON kanál nějaké blízké BTS a koukáme:

BTS: Jsem nejmenovaný operátor v České republice!

BTS: Jsem nejmenovaný operátor v České republice!

BTS: Jsem nejmenovaný operátor v České republice!

BTS: Telefon s tmsi 123456 nechť skočí na kanál 8, timeslot 2!

BTS: Jsem nejmenovaný operátor v České republice!

BTS: Jsem nejmenovaný operátor v České republice!

TMSI (Temporary Mobile Subscriber Identity) je identifikátor, pod kterým mobil komunikuje s BTS.

Chtěli bychom slyšet i něco víc, tak se na výzvu k přeladění (v řeči GSM Immediate Assignment) naladíme na ohlášený kanál a poslechneme si ohlášený timeslot.

¹⁶<http://events.ccc.de/congress/2009/Fahrplan/events/3654.en.html>

¹⁷<http://web.archive.org/web/20110715170344/http://reflexor.com/trac/a51>

¹⁸https://srlabs.de/decrypting_gsm/

¹⁹<http://www.mail-archive.com/a51@lists.reflexor.com/>

BTS: Telefon, mám tu pro tebe zprávu.

Telefon: OK.

BTS: Síla tvého signálu je 5 a tvoje time-advance je 1.

Telefon: OK.

BTS: Zapni šifrování.

(dál už pokračuje nesrozumitelný balast)

Time advance je „náskok“, o který musí mobil začít vysílat dřív – když je daleko od BTS, než se k ní signál dostane, chvíli to trvá a netrefil by se tak přesně do timeslotu, který mu byl přidělen. Změření TA k různým okolním BTS a následná triangulace se dá použít k poměrně přesnému zaměření mobilu.

Získání keystreamu

K použití tabulek, které jsme si spočítali výše, potřebujeme keystream. Vzduchem ale létají jenom zašifrovaná data. Z principu šifry víme, že zašifrovaná data jsou keystream XOR nešifrovaná data. Z toho vyplývá, že nešifrovaná data XOR zašifrovaná data je keystream. Zašifrovaná data jsme právě odposlechli, nešifrovaná data zdánlivě nevíme, ale můžeme se pokusit je uhádnout. V GSM se totiž přenáší spousta vaty, například pokud BTS zrovna nemá co říct, pošle prázdný rámec (nazvěme ho NOP) vyplněný paddingem (0x2b, ASCII znak +). Předání SMS by mohlo vypadat třeba takhle:

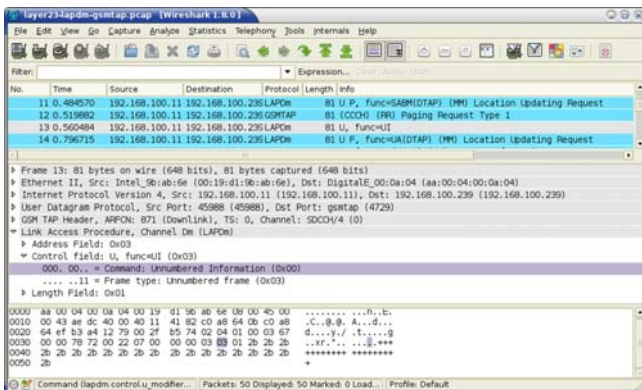
NOP+++++

Síla tvého signálu je 5 a tvoje time-advance je 1.

NOP+++++

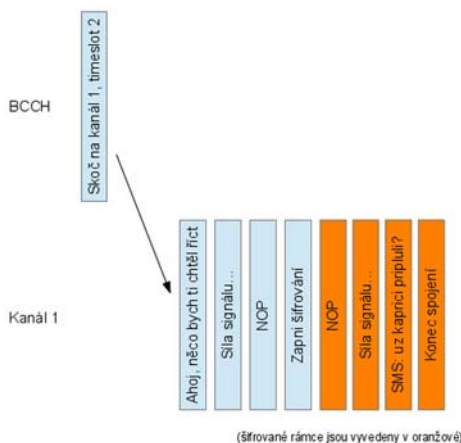
Máš tu SMS s~textem "uz kaprici pripluli?"

NOP+++++



Obr. 4: Prázdný rámec ve Wiresharku (posledních 23 bajtů) – 0x03 0x03 0x01 a pak padding z ASCII +

ZXORujeme předpokládaný NOP s přijatými daty a výsledek se pokusíme cracknout. S trochou štěstí získáme klíč, kterým dešifrujeme zbytek zprávy.



Obr. 5: Ukázka typického přenosu SMS z pohledu jednotlivých kanálů

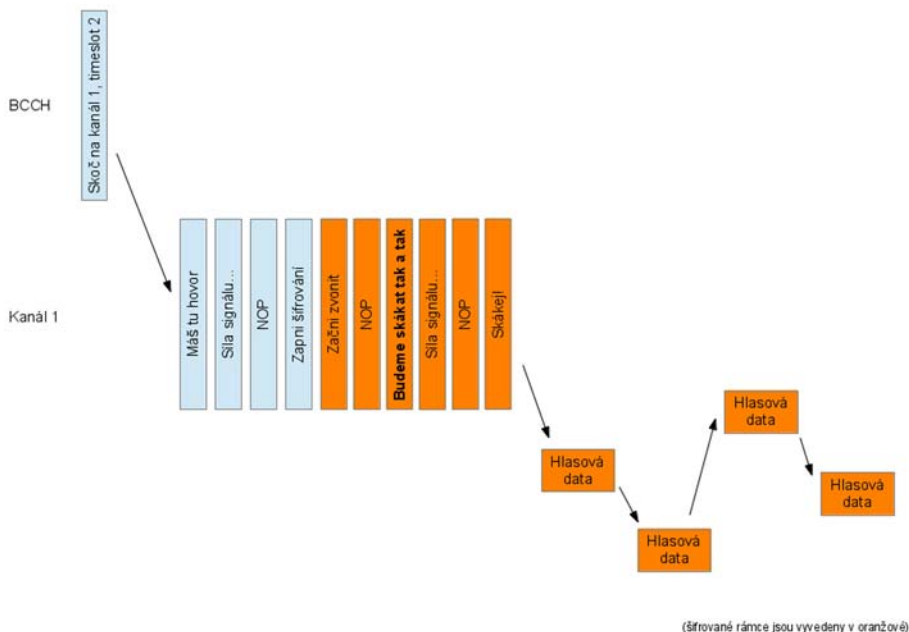
Poslech hovorů

Při hovoru se hopuje. Problém je v tom, že po kterých kanálech se bude skákat, se dohodne až po zapnutí šifrování. Takže:

- Buď nahrajeme celé pásmo a až crackneme klíč, doskáčeme si v tom zpětně. To vyžaduje drahé rádio, které dokáže celé pásmo zachytit.
- Smíříme se s tím, že nám začátek hovoru (než vycrackujeme klíč) unikne.
- Využijeme toho, že mnozí operátoři nemění klíče. Když je mobil pořád na jedné BTS, jeden klíč se používá dostatečně dlouho na to, abychom si ho mohli vycrackovat předem a při navázání hovoru ho už „měli připravený“.

Útok byl předveden na 27c3.²⁰ Velká část použitého softwaru byla uvolněna, ale některé části stále chybí – konkrétně program na uhádnutí keystreamu a následné dešifrování komunikace. Na uvolněné části odkazují na konci článku.

²⁰<https://www.youtube.com/watch?v=IsIriAdbttc>



Obr. 6: Typický hovor je složitější – začátek probíhá stejně, jako u SMS, ale samotný přenos zvuku už skáče po kanálech

Co s tím?

Nejlepší by bylo upgradovat z 25 let starého GSM na nějaký modernější protokol. Bohužel ani 3G, ani UMTS na tom také nejsou s bezpečností zrovna²¹ růžově²². Nejvíc by se mi osobně líbilo, kdyby operátor poskytoval IP konektivitu a já bych si přes to tuneloval šifrovaný VoIP (SIP přes TLS, RTP přes sRTP nebo ještě lépe zRTP), ale něčeho takového se asi hned tak nedočkám. Navíc po světě existují miliardy GSM telefonů a GSM zařízení na straně operátorů a jejich výměna není ani v horizontu 5–10 let reálná.

Šifra použitá v GSM je sice zlomená, i přes to lze ale útočníkovi její lámání všelijak znechutit. Útok, který jsme si popsali, využívá toho, že po síti létá spousta known-plaintextu v paddingu. Doporučení TS44.006 z roku 2008 definuje padding randomizovaný. I potom lze použít rámce, jejichž obsah lze tipnout (například rámec „System Information Type 5“ se silou signálu – telefon se nepohybuje zas tak rychle na to, aby to nešlo zjistit, nebo v krajním pří-

²¹<http://en.wikipedia.org/wiki/A5/3>

²²<http://wiki.thc.org/vodafone/>

padě i zbruteforcovat), ale ty se posílají méně často a náročnost útoku tak velmi stoupá. Nicméně, ačkoli náhodný padding byl doporučen již v roce 2008, například v České republice ho zatím implementuje pouze jeden ze tří operátorů.

Druhým kostlivcem ve skříni je málo časté překlíčování. Extrémem je operátor, u kterého, pokud je mobil stále na stejné BTS, vydrží klíč několik dní. Překlíčování bohužel sežere trochu výpočetního výkonu a přenosové kapacity sítě, ale když se bude dělat dost často, útočníka opět znechutí.

Nicméně všechna tato opatření pouze posouvají náročnost útoku zhruba o desítkový řád. Jak se bude dostupný výpočetní výkon s časem zvyšovat, bude se crackování A5/1 stávat snazším a snazším. Proto je potřeba myslet na to, že s GSM nikdy neslyšíte hovor sami.

Literatura

- [1] Na GSM s OsmocomBB.
<https://www.abclinuxu.cz/clanky/na-gsm-s-osmocombb>
- [2] Notes on sniffing. <http://bb.osmocom.org/trac/wiki/Sniffing>
- [3] Airprobe. <https://svn.berlin.ccc.de/projects/airprobe>
- [4] Eavesdropping on and decrypting of GSM communication using readily available low-cost hardware and free open-source software in practice.
https://www.os3.nl/_media/2011-2012/students/jeffrey_bosma/ot_report.pdf
- [5] Practical exercise on the GSM Encryption A5/1.
http://www.data.ks.uni-freiburg.de/download/misc/practical_exercise_a51.pdf

HARVESTING LOGS AND EVENTS USING METACENTRUM VIRTUALIZATION SERVICES

**Radoslav Bodó, Daniel Kouřil, Jiří Sitera,
Miloš Mulač, Pavel Vondruška**

E-MAIL: BODIK@CIV.ZCU.CZ, KOURIL@ICS.MUNI.CZ, SITERA@CIV.ZCU.CZ,
MULAC@CIV.ZCU.CZ, DEXTOR@CIV.ZCU.CZ

Abstract

Applications and operating systems describe important events using *logs*. The log record contain information that are very important to detect and fix issues in the infrastructure. Logs are especially crucial for security operations. In the paper we describe our experience with deployment of a central logging service. We also describe the solutions we deployed to process large volumes of data collected on the central server. These solutions operate in a cloud, which make it possible to us to tune the performace just according to current needs and loads of the logs.

1 Introduction

Centralized management of logs produced by computer systems is an important piece of operations. Having log records collected at a single point simplifies handling of various incidents since the operator has all the information collected centrally. For example a common task is to detect what steps a particular user has performed recently, which is needed during an investigation of a security incident. Without a centralized log repository it is necessary to connect to every single machine in the infrastructure and check all the logs that may be relevant. The whole process is too long and one risks that an important piece of information is missed. Another example is ongoing checks of the logs, e.g. to detect an issue immediately when it appears. Without centralized log management in place, all relevant machines in the infrastructure need to perform these checks independently, which is hard to establish and maintain. Being able to evaluate logs quickly and work with them on the fly contributes to solid system management and makes it more efficient.

Despite the basics of such deployment is well understood, we identified several areas that either are not covered at all or whose scope exceeds the possibilities of

current tools. In the paper we will present the deployment of a central logging service in the Czech NGI. We will describe the infrastructure to collect logs from machines that are distributed all over the country to a single place. Such an arrangement differs from usual deployments of remote syslog servers and we describe our experience which might be interesting for other NGIs, too.

Another principal problem concerns with the way how data is processed once it is collected and stored on the central service. The traditional way is setting up series of filters that detect some known patterns in the incoming data. This mechanism can only be employed for patterns that are known in advance and can only be applied to a new data. Sometimes it is necessary to analyze the data collected. There are several common tools that can be utilized for that, like `grep` and other standard Unix commands. However, processing a large amount of data with these tools is very intensive, which makes them unusable for interactive work.

Based on these limitations we decided to explore alternative ways how logged data can be processed and examined. We combined together several open source solutions and built up an infrastructure to fast index log records and manipulate them in an easy and quick way. The solution is based on an internal cloud that runs the indexing tools, which continually process received logs. Being provided in a cloud, the indexing service can shrink or enlarge on demand, based on current amount of the data and requirements of the operators. On the top of the pre-processed results we run visualization tools that are used to access the data and manipulate with it via web interfaces.

Having a centralized service collecting data from the whole NGI and tools to efficient manipulation with the data, made it possible for us to have a better control over the infrastructure. The fast access to information collected is appreciated by operators, infrastructure developers and also by the security officers. The possibility to interactively work with all logs also opened new possibilities of how this data can be utilized. For instance, it is much easier to detect mis-configured services or uncommon events, which would cause serious problems if they were not handled.

More details on the deployment and utilization of the centrally managed logs as well as details on particular configurations can be found in our technical report [11] that is an extended version of this paper.

2 Collecting Logs

Applications generate log records describing important events in their processing. The records have to be stored in a permanent storage for further use. In order to log an event the application use either an own mechanisms to store it in a file or database or use a dedicated library. For the latter case the syslog POSIX

calls are often utilized, which usually pass the log records to the syslog daemon running on the system. In the default configuration the syslog daemon stores log messages on a disk but it can also be configured to send them to a remote server over a network. In order to deliver log messages, the syslog messaging protocol [2] is used. If the logging is done over e.g. a private network and do not need any security, it is very easy to establish. Since the log messages may carry sensitive information, it is reasonable to ensure they are protected properly. To provide transport security of the log messages, the TLS protocol can be used [3], which is supported by the main open-source implementations. Remote logging, including its protection using TLS is properly documented and sources exist that describe it in details (e.g. [1]).

However, we were deploying the solution in the environment of project MetaCentrum¹ which relies heavily on the Kerberos security system. Deploying a parallel PKI infrastructure would have been an overkill, so we required to secure the messaging using Kerberos, too. Another requirement stem from the fact that the infrastructure is distributed all over the whole country, which increases chances of network outages. Therefore, a basic requirement for the whole messaging was that it provide sufficient fault-tolerancy so records do not get lost when they cannot be delivered for a while.

Based on these requirements we selected **rsyslog** as the implementation of the syslog messaging and server. Daemons of rsyslog are standard part of major Linux distributions and most of them utilizes it as the default solution to store the logs. It has a plugin-based architecture supporting various ways of authentication and channel protection. One of the plugins supports GSSAPI, which would allow us to leverage the existing security infrastructure of MetaCentrum. Early at the beginning of our project we found out that the module from the stable release of **rsyslog** (5.8 at the moment of writing) was crashing from time to time and initiated a fix, which was introduced in 5.8.x and onwards. However, even after the fix the plugin is not an ideal solution for a huge distributed environment. Having done quite extensive stress testing, we found out that the client side of the plugin is not able to reconnect correctly on server failures, which yields a huge traffic triggered by TCP reconnection attempts. Therefore, we have implemented a new GSSAPI module (**omgssapi**), which is based on the code of output plugins for the 5.8 family which merged the GSSAPI functionality from the existing module. The new module has been contributed to the community and is available from [12].

It is easy to instruct the rsyslog daemon on a node to pass every logs it receives to a remote host. However, in order to provide a robust delivery, a few additional options are needed and must be tuned properly. By default every rsyslog action invoked to process a message is processed directly in the context

¹<http://www.metacentrum.cz>

of main rsyslog message queue and the corresponding thread. If the action is long running or is failing and retrying (possibly forever), it can block processing of the other messages. To address this issue, it's advised to use one of the available action queue implementations to de-couple the processing of the action to a separate queue/thread. A general recommendation is to use a disk-assisted queue, which makes it possible for a queue to buffer its contents on a disk and to process actions asynchronously so the main processing is not blocked.

When using disk assisted queue implementation for TCP or GSSAPI (or other actions, the buffering can lead to filling whole disk space, which is not desired in production environment. In order to prevent this the queue must be restricted to use only defined maximum space. It is also inevitable to configure additional enqueueing timeout on the queue so it starts dropping messages that cannot be inserted and protect rsyslog from blocking indefinitely. Exact details and reasoning are provided in the technical report [11].

The configuration of a server collecting logs is quite straightforward and is also well described in the rsyslog documentation. Since it is not possible to combine multiple authentication mechanisms in a single connection, the server needs to listen to multiple TCP ports and clients needs to connect to appropriate port to perform requested authentication.

Server side is being done by default GSSAPI input module. Logs received can be forwarded to another service for processing or stored on the system in file or databases. When it is stored, the backend must be chosen properly to allow easy processing of the data when needed. We are used to store the the data in a file hierarchy based on the date and the IP address of the client.

3 Log processing

Once all system logs are stored on one place, it is possible to analyze them. A lot of various things can be monitored and extracted out of the log records. For example, the Torque batch system logs information about resource utilization (CPU, memory) of the jobs, which can be used to extract information about the actual utilization of the resources. This information can be communicated to the users whose jobs exceed granted capacities and possibly used to warn them before applying hard enforcement of the limits. The central storage makes it much easier to collect the information about activities of a particular user in the whole environment. Being able to access this information on demand is crucial for efficient function of both user support and security operational staff.

The common way how logged data is processed nowadays is to use general text handling tools like `grep`, `awk` or `perl`. While this approach works for simple cases, the task is getting complicated by the ever increasing size of the logs and also by the fact that subsequent queries requires grepping the whole set all

over again. Additional analytic tasks requires an efficient aggregation queries, scanning through the whole dataset and also some persistent storage for internal state and metadata.

The traditional solution to handle large volumes of data is to utilize relational databases. However, before taking the straightforward approach we decided to consider a few things:

The size In the grid environment it might not be sufficient to rely on a single node database system because amount of logs is waste. A scaling solution must be used by design. Probably an advanced partitioning or sharding features of traditional RDBMS like MySQL or PostgreSQL can be used.

The structure In the traditional RDBMS it is necessary to know in advance and work with the fixed structure of the data stored in the tables. Every change of the structure must be reflected not just by application using that data, but also by the corresponding database model and configuration. This requirement does not quite fit to the way how the logs are processed since it is often not known in advance what kind of data will be processed. For instance logs generated by Apache HTTPD differ a lot from logs produced by Kerberos KDC.

Those characteristics led us further into exploring new emerging technologies like text indexing and NoSQL databases, in particular we focused on ElasticSearch and MongoDB. Both solutions are designed to work in a cloud-like environment and both employ a structureless approach to work with the data — json documents. The first aspect is very likely to fit into grid computing environment possibly to address size and scaling issues. The second is likely to be more than helpful for fast prototyping or research development of the applications working with logs.

3.1 ElasticSearch

ElasticSearch² is a full-text engine to index and search text data, which is built on the top of the Lucene library [7]. ElasticSearch was designed for a cloud, allowing for dynamic adding and removing of nodes based on current requirements. It features several characteristics that makes it easy to use. In particular it supports autodetection of other ElasticSearch nodes on the same network, automatic splitting and distribution of the indices among the available nodes and their replication and relocation when needed. In order to search the indexed data users can utilize available interfaces (REST, native, thrift) or a web GUI provided by Kibana³.

²<http://www.elasticsearch.org/>

³<http://www.kibana.org/>

ElasticSearch can run as a single process but its advantage is its p2p character, which allows to dynamically add and remove its instances on multiple nodes within a local network. Therefore, it is possible to provide resources just according to current needs. More, every instance can play one or more roles of the node configuration (*client*, *master*, *data*) which is used for deploying a cluster with layered architecture. However, current interfaces do not support encryption or authentication so anyone who can talk to the cluster will be able to work with the data stored in the index. To address this limitation it is advised to make sure the cluster of ElasticSearch nodes is private and only available to authorized people or services. In MetaCentrum we leverage the virtualization framework to setup virtual clusters overlaying physical resources. Using the framework it is possible to instantiate virtual workernodes connected to the dedicated private VLAN. The VLAN itself is accessible from all main NGI sites through the CES-NET backbone network so the cluster members can be randomly allocated on all major NGI sites [8, 9]. Proxy servers are used to provide access to the cloud services running inside such network.

Establishing a cloud with multiple ElasticSearch nodes is as easy as starting an ElasticSearch instance on every node of the cloud. Utilizing the autodiscovery feature and automatic sharding, the nodes will establish themselves the topology and are ready to accept data to index and answer queries. No additional configuration of the nodes is necessary, especially no data schema needs to be specified. The data is just inserted and ElasticSearch handles it properly.

3.2 Turning logs into structured documents

Logs are not just text lines, but should rather be considered as structured data. There is at least a timestamp in every log along with some text data, and the data itself has a given structure [5, 6]. The structure of the log messages varies from product to product. ES is able to accept and parse JSON structures and index them so that queries can be made based on the provided fields. Using created index provides much better experience and response time in comparison with the common tools like **grep**.

There are several ways how logs can be passed to ES for processing. The simplest way is to use the ElasticSearch plugins available from the latest development release of rsyslog. However, in order to keep a better control over the parsing process we decided to use Logstash⁴ for several reasons:

Grok — the heart of parsing. A library⁵ which is able to parse text lines with regular expressions and turn them into structured data. Regular

⁴<http://www.logstash.net>

⁵<http://code.google.com/p/semicomplete/wiki/Grok>

expressions are written in enhanced language which allows pattern reuse and provides ability to create nested rules in very convenient way.

Logstash flexible architecture — the concept of event/message processing pipeline consists from the set of input, filtering and output plugins working in the separate threads and interconnected with the sized queue. Logstash supports a large scale of input and outputs which can be used to create flexible indexing services.

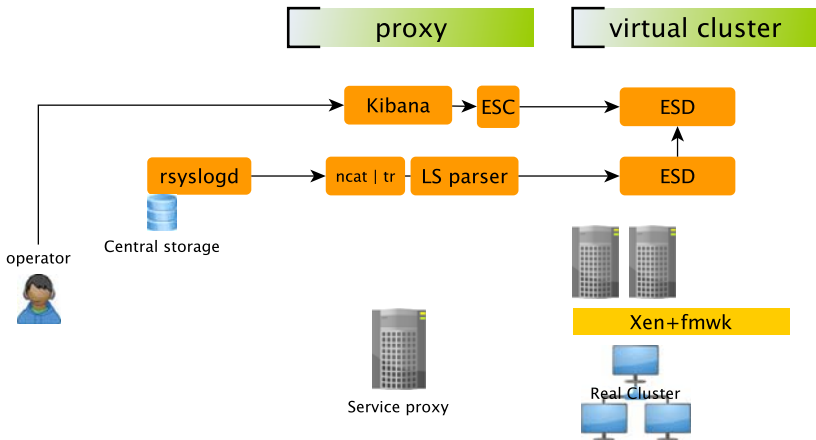


Fig. 1: Proxy parser

Logstash provides an entry point used to insert the syslog data to ElasticSearch. The basic arrangement is depicted in Fig. 1. There are ElasticSearch data nodes running within the virtual cloud and special services running on the proxy host are used to push (Logstash parser) and query (Kibana along with ElasticSearch client) the data into/from the cloud.

3.3 Performance tuning

Establishment of the whole infrastructure is quite easy and can be done quickly. While the infrastructure works well for small datasets, it has performance limits which needs to be addressed before bigger data is processed.

If more power is needed than what the existing deployment provides, a larger ElasticSearch cloud can be instantiated. In a larger cloud, however, the single LS parser becomes a bottleneck which needs to be addressed, ideally by moving the parser from the proxy node to the cloud and running multiple instances to

achieve more parallel processing. For that, some form of data spooling or other messaging queue is required, so indexing services running in the cloud can be configured to pop data from a single place. For example the `redis`⁶ server can be used in this manner. Redis is populated with logs sent from the syslog server and provides the data to parsers running in the cloud.

The default behavior of most logstash input and output plugins is to process only one message/event at a time. That may be fine for a low traffic environments but in cases of bootstrapping the Elasticsearch cluster, indexing historical dataset or recovering after total cluster failure, it is necessary to enhance the uploading speed by introducing support for batch processing at several places.

At `redis` as a message queue, batching can be done using pipelining of redis commands or using scripts in the LUA language. Scripts can result in less network utilization while pipelining leads in the smoother client requests interleaving on the server side. The latest logstash implementation uses LUA scripting.

For the final delivery of structured data from Logstash to ES cluster there are two main options. One can either use the native API plugin or an HTTP output plugin but neither is ideal. The native API plugin does not support any bulk indexing API yet. The HTTP plugin feeds the data by separate POST requests yielding significant overhead for standard syslog messages. In order to improve the performance of bulk indexing we have produced a customized output plugin that implements an enhanced batching system with synchronized internal buffer queue and time based flush.

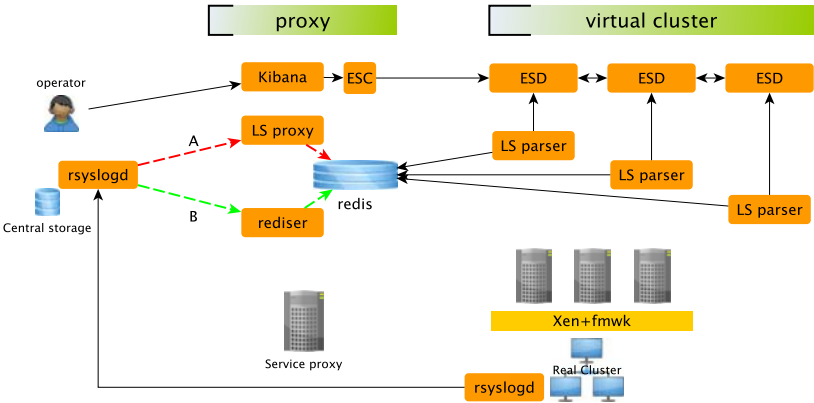


Fig. 2: Cloud parser

⁶<http://redis.io/>

Another bottleneck was revealed in the way how the Redis server is populated. By default, Logstash adds a lot of metadata information to every log record providing more information about the origin of the data. In our case, however, all data is arriving from the syslog server and it all relates to a syslog messages. Therefore the metadata only generates unnecessary overhead that increases the requirements for the processing of the records.

In order to produce a feeder for the Redis server, which does not add the overhead, we have created a separate component – `rediser`. Rediser is built on the Logstash source code and it makes possible to push incoming messages into redis queue without any modification.

Whole arrangement of a cloud tuned to better performance can be seen in Fig. 2.

3.4 The speed of ES upload and search

The speed of upload generally depends on many factors, such as the size of grokked data and final documents, batch/flush size of input and output processing, etc.

A real dataset from January 2013 in the size of 105 GB (~ 800 M events) was used for main testing. In the best found configuration the time for indexing is approximately 4 hours using 8 nodes ESD with 16 shared indexers (LS parser installed on all ESD nodes) and speed averages about 50 000 messages per second (eps). During regular operations an incoming rate 100–5 000 eps was observed. A single or two node cluster should be able to handle those rates.

Search speed was evaluated on basic level and is limited by IO throughput, but even when using small cluster is much faster than conventional tools. While a grep query over the data on the local filesystem required cca 18 minutes, the same query over indexed data in ElasticSearch took only cca 35 seconds.

4 Advanced data analysis

Logs contain a lot of interesting information that can reveal security incidents or attacks very quickly. The logs can also be correlated with other sources of information provided a combined knowledge, which can also point to a particular issue. While ElasticSearch makes it possible to quickly search the logs, it is not quite suitable to provide advanced analysis and data-mining in the data. Therefore we decided to utilize a database to store parts of the logs and provide additional processing of them. We choose the MongoDB⁷ nosql database since we aggregate information from multiple sources, which vary a lot in the way how it is structured.

⁷<http://www.mongodb.org/>

The database is populated from the central syslog server in a way that is similar to feeding data to ElasticSearch. Unlike the feed to ElasticSearch however, only a fraction of data is stored in MongoDB. For a pilot solution we focus mainly on information logged by the SSH servers and we used the logstash and grok to parse authentication logs.

Statistical analysis and other analytic tasks can be done on the index/collection produced by such a grok filter. For example, the aggregation framework of MongoDB and its map-reduce techniques can be used to generate time-based hierarchical aggregations. Using these techniques it is simple to group events originated in the same period of time. Other similar aggregated views can be produced too, see Fig. 3 for the whole schema.

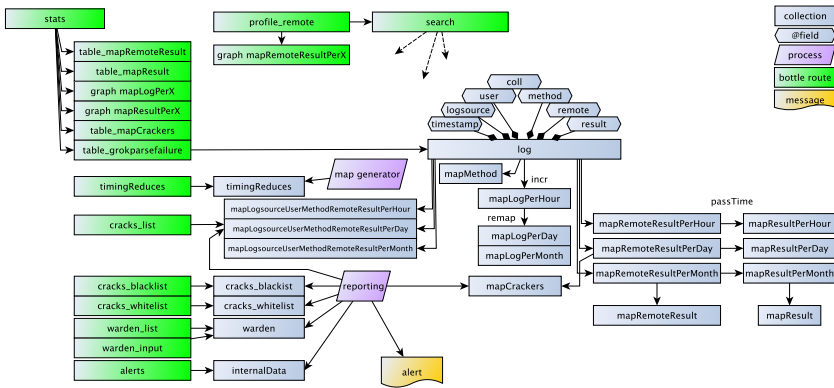


Fig. 3: Mongomine schema

4.1 The speed of MongoDB upload

The refined data can be further used for reporting or help with investigation of security events. All users and/or services interacting with the monitored domain can be profiled with the collected data. For instance it is easy to detect SSH attacks by correlating the number of unsuccessful login attempts.

It is also possible to include information from other sources. Our solution currently consumes data from IDS Warden⁸ operated by CESNET-CERTS within the CESNET2 network. Warden itself is a messaging platform used for sharing data about security incidents and attacks that have been detected by the CESNET member institutions and also by external partners. Data received from Warden is stored in MongoDB and used to detect attempts to access from malicious addresses, etc.

⁸<http://warden.cesnet.cz/>

Because the volume of the data fetched through Warden can be large and installation of the client requires some non trivial software dependencies it was decided to host the client in the virtual cluster as well as MongoDB service itself.

Data collected in MongoDB can be processed directly using the MongoDB interfaces. However, in order to simplify the operations and automate the most common queries we prepared a set of scripts to mine information from the database. These scripts can be run either manually on demand or as cron jobs. In the latter case the scripts emit alerts for security issues, like access from an attacking IP, etc.

Logstash mongod output does not use bulk inserts, neither we tried to develop this feature. Also only subset of sharding/clustering capabilities⁹ of MongoDB was used during our experimentations. Mainly because MongoDB does not yet include any autodiscovery or autoconfiguration features.

A real dataset from January 2013 in the size of 20GB (~ 33 M events) was used for testing. Time for inserting all data is approximately 8 hours using 1 MongoDB server with 20 shards and 1 dedicated indexer with 6 LS instances. The average time of all incremental aggregations during normal operations is 10 seconds.

5 Deployment in MetaCentrum

The described system has been in production for about a year and in our current deployment we have 90 % coverage of worker and service nodes connected to the central logging service, yielding cca 750 machines. The system is also receiving logs from other site grid site in the Czech NGI through a single syslog proxy supporting GSS-API security. The amount of stored logs is approximately 100GB per month.

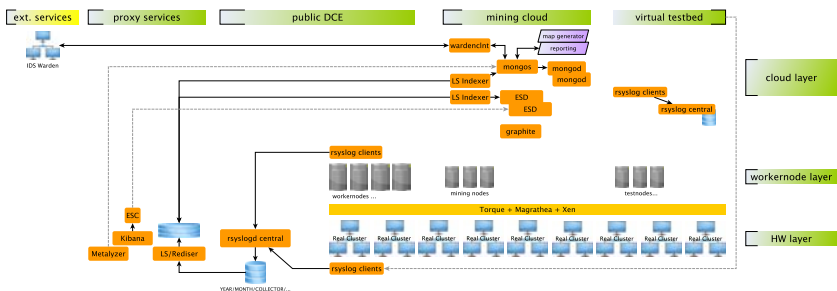


Fig. 4: Overall schema of mining architecture

⁹single host, 1 shard per CPU, 0 replica set

The overall schema as depicted in Fig. 4 consists of several subsystems. The logs produced within the infrastructure are collected by a central service. All communication is secured properly to avoid any leak of information in the logs and any information distortion. Attention is also paid to sufficient resiliency so that the system does not stop working in outages of the network, etc.

The indexing and searching part is primarily mediated by Elasticsearch and makes it possible to query the logs in a quick and easy manner. Elasticsearch consumes data passed from the central syslog server and the system has been tuned to provide sufficient performance and to utilize the power of the distributed environment. A proxy service used to accept the data and spool it in a queue before it is retrieved for parsing/tokenizing and subsequent indexing. The indexing services running in a cloud retrieves the workload from the queue and process it. The proxy also provides access to the searching interfaces as well as some auxiliary internal functions (DHCP, ...).

The important data is stored in a nosql database for further analysis and correlation with other information, e.g. results of intrusion detection systems. Custom scripts developed over this databases made it possible to quickly detect common security incidents that are hard to detect nowadays. For instance, we are able to detect successful SSH attacks, even if it is performed in a distributed manner, i.e. it does not target a single node. Also investigation of security incidents became quicker since the information is easy to obtain.

The environment is orchestrated from proxy server through set of shell and cron scripts. All components are installed from custom self-contained software packages provided to the cloud by the proxy.

In order to develop all the components needed we use custom clouds and virtualized environment provided by the MetaCentrum project, but system is able to run in any other cloud solution.

6 Conclusions

In this paper we have presented a system designed for collection and analysis of a logs from distributed environment. The system is based on open source projects and is backed up by cloud solutions provided within the Czech NGI MetaCentrum. Whole solution is designed to ease daily operations and security tasks and due to its cloud nature it is capable of scaling up and also down depending on actual needs.

All custom components and enhancements developed on the top of the used tools are published and available to the public domain in [12].

The system was developed mainly towards intrusion detection and incident response usecases. Experience gathered so far shows that it is much easier to collect all information with the new set of services. Often, the operator could

learned all information in 10 minutes using presented indexing and searching services in contrast of using traditional log handling processors like grep or perl, which would take seriously much more time.

Acknowledgments

The work has been supported by the research programme „Projects of Large Infrastructure for Research, Development, and Innovations“ (LM2010005). We highly appreciate the access to computing and storage facilities provided by the National Grid Infrastructure MetaCentrum, co-funded by the programme.

We also want to thank to the user groups of rsyslog¹⁰ and logstash¹¹ for their consultancy support provided during our project.

References

- [1] Peter Matulis: Centralised logging with rsyslog. *Canonical Technical White Paper*, September 2009. <http://www.canonical.com/about-canonical/resources/white-papers/centralised-logging-rsyslog>
- [2] R. Gerhards. „The Syslog Protocol“. IETF RFC 5424. 2009
- [3] F. Miao, Y. Ma, J. Salowey. „Transport Layer Security (TLS) Transport Mapping for Syslog“. IETF RFC 5425. 2009.
- [4] Rainer Gerhards: Reliable Forwarding of syslog Messages with Rsyslog http://www.rsyslog.com/doc/rsyslog_reliable_forwarding.html
- [5] Common Event Expression: Unified Event language for Interoperability. <http://cee.mitre.org/>
- [6] Jordan Sissel. „logging, logstash and other things“. *PuppetConf 2012*. <http://www.youtube.com/watch?v=RuUFnog29M4>
- [7] Shay Banon. „Road to a Distributed Search Engine“. <http://www.elasticsearch.org/videos/2011/08/09/road-to-a-distributed-searchengine-berlinbuzzwords.html>
- [8] Václav Novák, Pavel Šmrha, Josef Verich. „Deployment of CESNET2+ E2E Services in 2007“. *Networking Studies II, Selected technical Reports*. 2008. <http://www.cesnet.cz/vyzkum-a-vyvoj/dosazene-vysledky/networking-studies/networking-studies-2008/>

¹⁰rsyslog@lists.adiscon.com

¹¹irc.freenode.org#logstash

- [9] David Antoš, Luděk Matyska, Petr Holub, Jiří Sitera. „VirtCloud: Virtualizing Network for Grid Environments“. In *Advanced Information Networking and Applications — AINA 2009*. 2009.
- [10] Miroslav Ruda, Jiří Denemark, Luděk Matyska. „Scheduling Virtual Grids: the Magrathea System“. *Second International Workshop on Virtualization Technology in Distributed Computing*. 2007.
- [11] Radoslav Bodó, Daniel Kouřil, Jiří Sitera, Miloš Mulač, Pavel Vondruška. „Crunching Logs for Fun and Profit“. *CESNET Technical Report*.
<http://home.zcu.cz/~bodik/metasw/doc-harvesting-logs/harvesting-logs.pdf>
- [12] Bodó Radoslav: ESB repository
<http://home.zcu.cz/~bodik/metasw/doc-harvesting-logs/pubdata>

DANE (SSL v DNS) (CZ.NIC)

Ondrej Mikle

E-MAIL: ONDREJ.MIKLE@NIC.CZ

Úvod

Model certifikačních autorit (CA) má jednu zásadní slabinu: každá „důvěryhodná“ autorita může prakticky vydat certifikát na jakékoli doménové jméno. To se již několikrát stalo, například v roce 2011 v neslavně známém incidentu s íránským útočníkem zvaným ComodoHacker, který napadnul několik CA. Vydal si certifikáty například pro gmail.com a addons.mozilla.org. DigiNotar, jedna z napadených CA, v důsledku napadení zkrachovala.

Pro certifikáty existuje sice například X509v3 rozšíření „name constraints“, které omezuje doménové jména, pro které může CA vydat certifikát, ale tato extension je pro jakokoli větší CA prakticky nepoužitelná.

Předchůdci DANE a podobné protokoly

První nápad byl jednoduchý – podívat se na certifikáty a jejich fingerprinty z více míst na síti, čímž by se většina MitM útoků odhalila. Jedinou výjimkou je „eclipse attack“, při kterém je útočník „příliš blízko“ cílovému serveru (například na routeru jeden hop od cílového serveru). Navzdory tomu, jak je tato technika stará, první známější implementace byl až Firefoxí doplněk Perspectives. Perspectives se jednoduše zeptá několika notářských serverů, jaké vidí fingerprinty certifikátů pro danou dvojici FQDN/port, porovná fingerprinty. Notářské servery odpovědi podpisují, veřejné klíče se distribuují spolu s Perspectives.

Metoda Perspectives ovšem nefunguje moc dobře s velkými CDN (content-delivery network) a load-balancery jako je třeba farma serverů Googlu. Důvodem je, že každý notářský server se často trefí do jiného uzlu z „CDN farmy“ a tudíž vidí jiný certifikát. Rozlišování skutečného útoku od normálního fungování by bylo v tomhle případě složité.

Další metodou je tzv. „CA pinning“, kde se k FQDN přiřadí certifikát nebo veřejný klíč CA, od které má být certifikát pro FQDN podepsán. Tento přístup implementoval Google Chrome, ale jenom pro služby Googlu a několik

dalších. Seznam „pinovaných FQDN“ je zadrátován přímo v prohlížeči, z toho důvodu není metoda škálovatelná. Nicméně, shodou okolností to byl právě tento mechanismus v Google Chrome, který pomohl odhalit uvedenou kompromitaci DigiNotar CA.

DANE

Protokol DANE byl standardizován jako RFC 6698 a implementuje obecnější „pinning“ protokol, než je ad-hoc implementován v Chrome. Je možné pinovat CA certifikáty, certifikáty koncových serverů i „holé klíče“. I když DANE protokol je v zásadě celkem jednoduchý, trvalo asi dva roky, než se z draftu dostal do fáze standardu. Většinou se jednalo o různé formality PKIX, okrajové případy a občasné „politické hádky“.

DANE umožňuje provozovatelům TLS služeb k doménovému jménu vytvořit v DNS TLSA záznam určující certifikát služby (pro trojici FQDN, protokol a port). Přiřadit certifikát lze jednou z následovných možností, tzv. certificate usage:

- usage 0: řetězec musí projít PKIX validací u klienta a musí jít přes certifikát v TLSA
- usage 1: řetězec musí projít PKIX validací u klienta a serverový certifikát musí souhlasit s certifikátem v TLSA
- usage 2: řetězec musí projít PKIX validací když certifikát v TLSA použijeme jako trust anchor
- usage 3: serverový certifikát musí odpovídat certifikátu v TLSA

První dvě možnosti lze volně popsat jako restrikcí aktuálního stavu s „předinstalovanými CA“: klient udělá validaci, ale navíc zamítne jakýkoliv řetězec neodpovídající TLSA záznamu. Usage 2 umožňuje udělat si „vlastní CA“ a poslední například použít self-signed certifikát. V praxi se poslední dvě možnosti až tak neliší, šlo o äkompatibilitu s RFC 5280 a definicí PKIX validace.

Druhý parametr TLSA záznamu je selector – říká která část certifikátu se kontroluje:

- selector 0: celý certifikát
- selector 1: veřejný klíč (struktura SubjectPublicKeyInfo z certifikátu)

Posledním parametrem je matching type, jak se porovnává certifikát z řetězce proti datům z TLSA záznamu:

- matching type 0: kontrola bajt po bajtu na vybrané části

- matching type 1: SHA-256 vybrané části
- matching type 2: SHA-512 vybrané části

Důležitým požadavkem je, že TLSA záznam musí být podepsán DNSSEC-em (jinak by to příliš nemělo smysl).

Příklad TLSA záznamu

Podíváme se na existující TLSA záznam pro HTTPS server ‚torproject.org‘ na portu 443. Když se prohlížeč připojí, vyžádá si od DNS TLSA záznamy jmennující se ‚_443._tcp.torproject.org‘. Ten by vypadal následovně:

```
_443._tcp.torproject.org. 3600 IN TLSA 3 1 2
4DF6667294FC1DFD6A95DA7BD8959B717B68213E2539DDECAEB09DE0B755A1B9
72707EE45FAA4D4891CCF22B1103C729105695E28AE891331E0C56E635FCFF57
```

Pořadí parametrů v TLSA je usage, selector, matching, data. Tudíž klient ověří, jestli veřejný klíč v serverovém certifikátu serveru torproject.org má SHA-512 hash identický s hashem deklarovaným v posledním poli TLSA záznamu nahoře: ‚4DF6...FF57‘. Prakticky lze s TLSA záznamy můžete experimentovat použitím nástroje [swede](<https://github.com/pieterlexis/swede>).

I když základní idea je velmi jednoduchá a jasná, existují „patologické případy“ např. s wildcard doménami nebo SRV záznamy. Na draftu pro kombinaci DANE s SRV záznamy se pracuje. Další trochou nedeterminismu přispívá způsob vytváření řetězců v různých klientech – můžou používat cachované certifikáty a jiné triky z RFC 4158.

Existuje i DNSSEC stapling implementován jako experimentální featura Chrome – DNSSEC řetězec se „přimíchá“ do TLS handshake. Výhodou DNSSEC staplingu (podobně jako u OCSP staplingu) je lepší soukromí a rychlost (nemusí se dělat extra request na DNS servery). Nevýhoda je složitější implementace na straně klienta i serveru. Vypadá to, že tento experiment se dál nevyvíjí.

Rozšíření DANE

DANE je protokol dost nový, proto nepřekvapí, že jeho rozšíření nebude moc velké. Skenováním kompletních zón .com, .net, .org ke konci roku 2012 jsme našli cca 50 TLSA záznamů na doménách druhé úrovně. GnuTLS je jedna ze známějších knihoven, kam byl zatím DANE protokol implementován. Autoři Postfixu pracují na implementaci i do Postfix serveru, ale tam ještě není úplně konečný draft, jak má DANE spolupracovat s SMTP protokolem.

