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

XXVIII. konference – XXVIIIth conference

Sborník příspěvků
Conference proceedings

Hotel Zámek Lázeň
Chudenice
21.–24. května 2006
May 21–24, 2006

Sborník příspěvků z XXVIII. konference EurOpen.CZ, 21.–24.května 2006

Plzeň 2006. První vydání.
Editor: Vladimír Rudolf, Jiří Felbáb
Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis
Vytiskl: IMPROMAT CZ, spol. s r. o., Kopírovací centrum RICOH
Smetanovy sady 6, 301 37 Plzeň

ISBN 80-86583-

Příspěvky neprošly redakční ani jazykovou úpravou.

Obsah

Botnety – sítě infikovaných počítačů

Helena Nykodýmová

E-mail: nykodymova@bis.cz

Vincent Gulotto[1]*, viceprezident bezpečnostního týmu McAfee: „Práce hackerů je čím dál jednodušší: v internetu existuje takové množství nezabezpečených počítačů, že hackeři nemusí sami hledat bezpečností díry, ale stačí jim počkat na vydání bezpečnostní záplaty a reverzním inženýringem z ní zjistit, kde je díra. Tu pak snadno využijí, protože uživatelé záplatu nenainstalovali.“*

Svět botů

Odborníci otevřeně hovoří o tom, že až 70 % dnešních zákeřných kódů vzniká čistě za účelem zisku. Škodlivé kódy již nejsou destruktivní a na infikovaném systému způsobují minimální viditelné škody. Jejich cílem je šířit se tiše, usídlit se v co nejvíce systémech a ty následně zneužít pro nekalou činnost. Zrodil se bot – program, který umí plnit příkazy útočníka zadávané z jiného počítače.

Termín bot pochází z českého výrazu „robot“ a stejně jako robot je využíván pro různé „práce“. Bot představuje klientský program, který proměňuje počítač v „terminál“ umožňující jeho vzdálené ovládání.

Je-li počítač infikován botem, stává se z něho tzv. „zombie“ (živá mrtvola) - stroj ovládaný útočníkem bez vědomí uživatele.

Bot – automatický program, který obsahuje komunikační a kontrolní modul. Bot lze vzdáleně ovládat tak, aby plnil požadované příkazy. Bot je velmi flexibilní, díky možnosti vzdáleného ovládání dokáže plnit nové příkazy autora a měnit svoji funkčnost.

Zombie – počítač, který byl nakažen a je ovládán jinou osobou.

Botnet (bot network) – sít předem infikovaných počítačů mnoha různých uživatelů, které lze kontrolovat a řídit prostřednictvím nástrojů vzdálené správy.

[1] http://www.crn.com/sections/breakingnews/

Různých bot programů se vyskytuje celá řada. Podle odhadů se počet botů blíží k tisíci v rozmezí od velmi jednoduchých až po velmi složité. Trendem dnešní doby je skutečnost, že dochází ke stírání hranic mezi jednotlivými typy malwaru. Některé boty mohou být trojskými koni, jiné vykazují vlastnosti virů a jsou schopny se samy o sobě šířit nebo se může jednat o kombinovanou hrozbu např. spojení červa s botem.

Přeměna počítače v zombie

Co umožňuje úspěšné šíření botů? Stačí k tomu známé triky malwaru, který využívá známých chyb v systému. Jednoznačným cílem jsou zejména domácí počítače, které nemají velmi často kritické záplaty systému, aktualizovaný antivirový program či nepoužívají firewall. Takovéto systémy se stávají lehkou obětí škodlivých kódů.

Oblíbeným terčem hackerů je samozřejmě systém Windows. Boty ke svému šíření využívají celou řadu bezpečnostních nedostatků v produktech Microsoftu a šíří se pomocí exploitů, které známou chybu zneužívají.

Boty se mohou také šířit za přímé účasti oběti, pokud nejsou dodržovány základní zásady bezpečnostního chování na internetu. Riziko představuje navštěvování nedůvěryhodných webových stránek, stahování programů z neověřených zdrojů či otevíraní podezřelé přílohy v e-mailu.

Ideální prostředí pro hackery představují počítače, které jsou trvale připojeny k Internetu trvale prostřednictvím širokopásmového připojení. Takových počítačů neustále přibývá.

Síla botnetu

Kompromitované počítače stejným botem jsou sdružovány do automatické sítě botnet někdy nazývané jako armáda zombie. Botnet lze řídit z jednoho vzdáleného počítače tak, aby kompromitované počítače prováděly na povel jednotné příkazy. Přitom o skrytých činnostech napadeného počítače nemá jeho majitel kolikrát ani tušení.

Aby bot mohl provádět požadovanou činnost, musí komunikovat s autorem. Boty mohou být ovládány několika různými způsoby přes P2P (Peer to Peer) sítě, diskusní skupiny a nejvíce přes chatovací kanály IRC (Internet Relay Chat). Prostředí IRC sítí představuje pro botnety ideální prostředí: především protokol IRC nebyl navržen s ohledem na bezpečnost, IRC serverů existuje velké množství a jsou propojeny do různých sítí, většina sítí používá slabou autentizaci, je zde zajištěna anonymita uživatelů.

Jakmile je bot aktivován, připojí se na předem dohodnutý IRC server a čeká na příkazy, které hacker zapisuje do chatovacího kanálu. Takto ovládané červi mohou být vybaveny následujícími funkcemi:

- rozesílání spamu;
- účast na distribuovaných DoS (Denial of Service) útocích;
- sběr informací z hostitele (např. uživatelská jména, hesla, čísla platebních karet, produktové klíče);
- šíření zákeřného kódu;
- zneužívání pay-per-click reklamních programů automatizovaným otevíráním webových stránek s reklamou.

Tyto příkazy pak může vykonávat více počítačů najednou (celá botnetová síť). Následkem může být provedení nárazového útoku, kde kritické úrovně je pak dosaženo během několika minut.

Spojení kriminality s byznysem

Vytváření botnetů může být velmi výnosný byznys: zisk může být z prodeje nebo pronajmutí botnetu či prodeje získaných důvěrných informací, pokud nejsou přímo zneužity. Zločinci také vydírají obchodní uskupení hrozbou masivních DoS útoků a vyžadují po svých obětích výpalné. Pokud společnost nezaplatí, hrozí, že útok provedou.

Ochrana domácích počítačů

Co můžeme udělat pro to, aby naše počítače nebyly využívány ve prospěch někoho jiného?

Zopakujme si známé bezpečnostní zásady:

- pravidelně aktualizujte operační systém;
- používejte antivirový a antispywarový program a pravidelně jej aktualizujte;
- chraňte své připojení firewallem;
- přistupujte pouze k důvěryhodným webovým stránkám;
- neinstalujte neověřené nebo neznámé aplikace;

- před potvrzením volby prostudujte, co potvrzujete;
- nereagujte na nevyžádané emailové zprávy;
- neotevírejte nevyžádané nebo neznámé přílohy e-mailů;
- v případě potíží kontaktujte odbornou pomoc.

Některé druhy infekce (backdoor) lze jen velmi těžko odhalit a tak mohou být neustálou hrozbou. Pokud je počítač botem již infikován, není jednoduché ho odstranit. Navíc vyčištěním od infikovaných souborů nemusí nic skončit. V obou případech stoprocentní jistotu přinese pouze reinstalace systému. K internetu je pak možné se připojit až poté, co byly odstraněny všechny komponenty botu a byly nainstalovány všechny bezpečnostní záplaty.

Ochrana systémů

V informačních systémech společností je nutné jasně stanovit pravidla komunikace chráněných systémů s nedůvěryhodným okolím. Jde především o stanovení způsobů komunikace, které budou z chráněných systémů přípustné. Nejčastěji jde o elektronickou poštu, prohlížení webu a služby spojené s provozem informačního systému organizace (např. messaging aplikací, klienti přistupující k aplikačním či databázovým serverům).

V boji proti malwaru je vhodné dodržovat tato doporučení:

- restriktivní pravidla elektronické komunikace;
- kontrola elektronické komunikace;
- informovanost zainteresovaných pracovníků o aktuálních hrozbách;
- definice politiky záplatování;
- restriktivně nastavené firewally;
- kde nepomůže prevence, je potřebná detekce (IDS/IPS).

Závěr

V prostředí internetu stále existuje velké množství nezabezpečených počítačů, které mohou být snadno infikovány. Boj proti počítačové kriminalitě musí spočívat i v prevenci a ve zlepšování bezpečnostního povědomí uživatelů.

Literatura

[1] *Data Security Management.* č. 6/2005, Botnety – nová hrozba?

[2] http://en.wikipedia.org/wiki/Botnet

[3] http://honeynet.org/papers/bots

[4] http://swatit.org/bots

[5] http://www.cert-in.org.in/knowledgebase/ whitepapers/ciwp-2005-05.pdf

[6] http://www.securityfocus.com/columnists/398

[7] http://www.symantec.com/enterprise/threatreport/index.jsp

Rootkity

Zdeněk Říha

E-mail: zriha@fi.muni.cz

Klíčová slova: Rootkit, malware, skrývání, operační systémy, detekce rootkitů

Abstrakt

Rootkit je sada nástrojů používaná hackerem po napadení počítačového systému pro udržení přístupu k systému a jeho využití ke škodlivým aktivitám. Spočívá především ve skrývání nežádoucích aktivit útočníka před ostatními uživateli systému (včetně administrátora).

Zatímco první rootkity byly v podstatě jen upravenými verzemi bežných systémových utilit, modernější verze rootkitů se již soustředí přímo na změnu funkčnosti jádra, takže rootkitem upravená realita je pak „servírována“ všem uživatelským aplikacím (ve smyslu jejich běhu v uživatelském prostoru, tj. včetně aplikací spustěných administrátorem).

Skutečnost, že primární činností rootkitů je něco skrývat (ať už to jsou procesy, soubory, položky registrů či síťová spojení), je často využita při detekci rootkitů a to srovnáním pohledu na tyto prostředky OS přes rootkit a bez něj (buďto nabootováním z čistého média nebo přímějším přístupem k HW či interním strukturám OS).

Se vzrůstající rafinovaností rootkitů se i detektory rootkitů stávají sofistikovanějšími (a naopak :-)). Řada iterací tohoto neustálého boje je stále ještě před námi.

1 Úvod

Jakým způsobem útočník (dnes označovaný jako hacker) získal přístup k administrátorských právům nás příliš nezajímá (příkladem může být chybějící patch, špatná konfigurace, 0-day exploit nebo přístup ke konzoli administrátora). Co nás teď zajímá je to, co útočník udělá po získání takových práv. V každém případě útočník usiluje o to, aby po sobě v napadeném počítači zanechal co nejméně stop, typicky si chce si také ponechat možnost opakovaného budoucího přístupu do něj. A k tomu mu poslouží právě rootkit. Rootkit sám o sobě ale obvykle nevykovává přímo škodlivou činnost, jen ukrývá kód tuto činnost

provádějící. Pod takovýmto kódem si typicky můžeme představit keyloggery či jiný odposlouchávající software, klienty DDoS útoků a programy pro rozesílání spamu.

Wikipedia [34] definuje *rootkit* jako sadu nástrojů používaných hackerem po napadení počítačového systému pro udržení přístupu k systému a jeho využití ke škodlivým aktivitám.

Rootkity jsou dnes k dispozici jako hotové softwarové balíky připravené k rychlé a snadné instalaci. Rootkity umožní provést všechny operace „na jedno kliknutí“ a v kombinaci s automatickým zkoušením exploitů může postup útočníka vypadat tak, že večer spustí program a ráno si prohlédne seznam ovládnutých strojů.

Skrývací vlastnosti rootkitů dnes používá také řada jiných typů programů. Viry a červy začínaje a konče softwarem pro digitální správu práv (viz známý případ firmy Sony [35]). K ovládání počítačů na dálku slouží také tzv. boty. Boty se typicky připojují k určitému serveru a od něj očekávají příkazy, k jednomu serveru se připojuje velké množství (např. tisíce, někdy se však mluví i o podstatně vyšších číslech) botů a všechny provádějí stejné příkazy (ideální např. na DDoS útoky). U rootkitů spíše očekáváme připojení skutečného člověka (hackera). Ovšem principy obou jsou podobné a rozdíly se mohou stírat.

2 Unixové systémy

Útok, při kterém útočník získá administrátorská práva, je obvykle doprovázen řadou záznamů v logovacích souborech pořízených příslušnými systémovými mechanismy. Prvním krokem útočníka proto bude likvidace těchto usvědčujících záznamů. V unixových systémech se jedná především o záznamy syslogu, wtmp, utmp a lastlog (stále častěji je ale vstupním bodem webserver, zajímavé jsou proto i jeho logy). Ke smazaní lze použít jednu ze známých utilitek wted, zapper nebo z2, které mažou nebo alespoň nulují záznamy. Vynulované záznamy sice nejsou standardními utilitami operačního systému zobrazovány, specializované programy (pro detekci manipulace se záznamy) si ale takových záznamů všímají a informují o nich správce. Proto je (pro útočníka) bezpečnější problematické záznamy úplně smazat. Pro administrátora to pak znamená, že detekce „děr“ v záznamech není úplně spolehlivá. Ačkoliv úprava logovacích záznamů není hlavní náplní činnosti rootkitů, jsou tyto obvykle vybaveny nástroji na mazání záznamů, a detektory rootkitů naopak utilitami na vyhledávání „děr“ v logovaných datech.

Další krokem útočníka bývá umožnění následného snadného přístupu k napadenému počítači (tj. instalace zadních vrátek, tzv. backdoor). Zde má útočník řadu možností od jednoduchého přidaní uživatele s UID 0 nebo nabindování rootovského shellu na určitý port, přes úpravu přihlašovacích procedur (změna

programu login, konfigurace r* služeb apod.) až po speciality typu zadní vrátka přístupná pouze z určitých IP adres a komunikující přes běžně používaný port (např. 80) bez vlivu na funkčnost aplikace normálně navázané k tomuto portu, tj. http [16]. Kromě přihlášení ze sítě je obvykle umožněna i eskalace privilegií běžného uživatele systému, kdy po zadání speciálního parametru (a případně hesla) některému běžnému SUID (např. chsh) programu (který je takto útočníkem změněn) je například možné přímo získat rootovský shell. Instalace zadních vrátek bývá standardní součást rootkitů, často jde o úpravu ssh serveru spočívající v umožnění přístupu útočníka po zadání speciálního hesla. Z hlediska útočníka jsou samozřejmě atraktivní takové možnosti, které jsou nenápadné i z hlediska analýzy síťového provozu. Na napadeném stroji totiž může skrývat vše jak se nám líbí, sítový provoz je ale viditelný i z jiných (nenapadených) strojů. Nově bindované porty také snadno odhalí nmap nebo podobný nástroj. Nenápadné alternativy zahrnují již zmínění využití běžného portu, na kterém komunikuje standardní program a dodatečný provoz rootkitu je odfiltrován na úrovni jádra ještě než se dostane k naslouchající aplikaci (takový provoz je samozřejmě možné v síti také zachytit, není však jinak nápadný, protože k těmto běžným službám typu http přistupuje obrovské množství klientů), porty „otevřené“ jen pro spojení z určitých IP adres (je to obrana proti nmapu, ale analýza provozu sítě může útočníka prozradit) nebo tzv. pasivní komunikace, kdy se neotvírá nové spojení, ale využívá se metadat existujících spojení (zde je nutné mít přístup k routerům). Pojďme se však podívat na hlavní část rootkitů, jíž je kód (program), který se snaží zakrýt všechny kompromitující aktivity útočníka a nechat administrátora v domnění, že všechno je v pořádku, server nebyl úspěšně napaden a žádná podezřelá aktivita není nebo nebyla vyvíjena.

2.1 Trojský kůň

První verze rookitů se snažily zakrýt svoji přítomnost nahrazením základních utilit administrátora jinými programy – trojskými koni. Například program ls pro zobrazení obsahu adresáře byl nahrazen upravenou verzí, která se chovala zdánlivě stejně jako originál, ale nezobrazovala některé soubory útočníka (a rootkitu samotného). Obdobně program `ps` pro získání seznamu spuštěných procesů skrýval procesy útočníka. Tímto způsobem upravených utilit bývá celá řada; aby iluze, že všechno je v pořádku, byla co nejdokonalejší. Například Rootkit IV obsahuje upravené verze utilit `ls`, `find`, `du`, `ps`, `top`, `pidof`, `netstat`, `killall`, `ifconfig`, `crontab`, `tcpd` a `syslogd`. Administrátor, který používá tyto podvržené utility, nemusí registrovat aktivity útočníka po dlouhou dobu. Podezření obvykle pojme, až když něco přestane pracovat tak, jak byl zvyklý. Ačkoliv výhodou rootkitů ve formě trojských koní je platformní nezávislost v tom smyslu, že jsou k dispozici ve zdrojovém kódu a útočník je kompiluje až na cílovém systému, je toto zároveň i nevýhodou. Rozdíly mezi parametry příkazové řádky jsou

nejen mezi jednotlivými unixovými systémy, ale i mezi jednotlivými distribucemi jednoho systému a to i u tak běžných programů jako je ls nebo ps.

Pokud útočník udělá chybu a použije nesprávnou verzi trojské utility pro daný systém, může se stát, že chování utility bude odlišné, což upozorní administrátora, eventuálně i obyčejné uživatele. Existuje celá řada dalších možností odhalení útočníka, neboť útočník nemůže upravit všechny potenciálně používané nástroje a dříve či později může administrátor na nesrovnalosti přijít. Oblíbeným trikem může být použití programu dd (který obvykle nebývá nahrazován trojskou verzí) pro zobrazení obsahu souboru, rozdíl mezi výsledkem „`ls`“ a „`echo *`“ či použití souborového manažeru mc.

2.2 Jádro

O řád lepší úroveň ukrytí mají rootkity nové generace, které přímo upravují jádro operačního systému. Princip spočívá v tom, že místo úpravy uživatelských programů typu ls ke skrytí souborů, procesů apod. se rovnou upraví volání jádra tak, že všechny uživatelské programy (včetně těch spuštěných administrátorem) se již dostanou k „cenzorovaným“ informacím. V žádném z moderních operačních systémů nemají uživatelské programy (programy běžící v uživatelském prostoru) přímý přístup k hardwaru a musí žádat operační systém o zprostředkování formou definovaného systémového volání. Změnou funkcí tohoto API (např. volání open – otevření souboru, read – čtení ze souboru, getdents – čtení adresáře) je možné lehce ukrýt soubory, procesy, síťová spojení apod. před všemi procesy v systému. Náročnost změny chování jádra závisí na konkrétním systému (administrátorská práva jsou vyžadována v každém případě).

V případě operačního systému Linux do verze jeho jádra 2.4. s aktivovaným modulárním systémem je úprava hračkou [4, 10] (jádra do verze 2.4. exportují symbol s tabulkou systémových volání `sys_call_table` a vložení nového modulu je nejjednodušší forma modifikace jádra), od verze 2.5. již tabulka systémových volání není exportována (zřejmě právě z důvodu jejího zneužívání), lze ji však rychle najít v kódu obsluhy přerušení 0x80 (systémového volání) [11, 12]. Linuxové jádro bez modulárního systému lze modifikovat přímým zápisem do paměti (přes speciální zařízení /dev/kmem) [11, 12]. Rootkity modifikující jádro existují i pro operační systémy Solaris (Sparc i Intel), OpenBSD a FreeBSD, linuxové rootkity pro 32bitovou architekturu Intel jsou však zdaleka nejrozšířenější.

2.3 Obrana

Předně je potřeba zdůraznit, že úloha rootkitu začíná až po získání administrátorských oprávnění, takže je třeba úsilí zaměřit především takovým směrem, aby útočník tato práva nezískal. 100% bezpečnost však neexistuje, a proto je nutné prevenci doplňovat detekcí.

Detekce starších trojských rootkitů je relativně snadná. Protože útočník zaměňuje binární verze některých utilit za svoje verze, stačí kontrolovat změny těchto klíčových utilit programy typu tripwire nebo „rpm -V“. Samozřejmě pak musí být jak samotný program pro výpočet hashů tak i vypočítané hashe programů zabezpečeny z hlediska integrity. Detekovat rootkity modifikující jádro operačního systému je možné například pomocí testování vedlejších efektů upravených systémových volání [17], kontrolou neobvyklých příznaků procesů (rootkity je používají pro interní potřeby), kontrolou integrity tabulky systémových volání či procházením paměti a hledáním známých vzorů daného rootkitu (např. podpis autora), struktur zavedených modulů (pro moduly, které se později skryly vypojením z lineárního seznamu), struktur procesů (obdobně pro skryté procesy) apod. Řada rootkitů například zakrývá promiskuitní režim síťového rozhraní (protože jej používá k odposlouchávání síťového provozu na odchytávání nešifrovaných hesel); jednoduchý test spočívá v uvedení rozhraní do promiskuitního režimu (např. spuštěním programu tcpdump) a zkontrolováním, zda příkaz `ifconfig` tento režim korektně zobrazí či nikoliv. Existují rovněž specializované programy pro vyhledávání rootkitů. Mezi takové programy patří například rkhunter [25] a chkrootkit [18], které jsou zaměřeny především na trojské koně. Protože se své činnosti spoléhají na jádro, jsou oba dva na moderní rootkity krátké. Paměť jádra přímo analyzují například detektory kstat [26] a rdetect [27], které k paměti přistupují pomocí čtení zařízení /dev/kmem a jsou tak závislé na správné činnosti systémových volání open a read s tímto zařízením – zde se rootkity zatím neangažují, to se však v budoucnosti může změnit. Pro FreeBSD existuje podobný LKM Detector (sec_lkm) [28]. Jakýkoliv detektor je však více či méně závislý na spolupráci jádra. Pokud rootkit ví, co detektor při své práci sleduje, může mu vždy podstrčit nesprávné hodnoty nebo jej zcela deaktivovat (jde tedy v principu o to, jestli máte aktuálnější rootkit či detektor rootkitů). Můžeme sledovat jasný trend, že i detekce a obrana vůči modifikaci jádra rootkitem se stěhuje do kódu jádra (například speciálních modulů). Pro opravdu spolehlivou detekci je však nutné opustit (potenciálně) modifikované jádro, nabootovat jádro čisté a zjistit, jaký je pravý stav věcí.

Vraťme se ale na chvíli opět k prevenci. Protože aktivní modulární systém (LKM) je nejjednodušším způsobem modifikace jádra, znamená jeho vypnutí eliminaci celé řady rootkitů, které jsou na této vlastnosti jádra závislé. Dalším způsobem obrany může být omezení manipulace s pamětí pomocí zařízení /dev/kmem. Existují úpravy jádra, které nepovolují u zařízení /dev/kmem čtení ani zápis. Například Linuxová distribuce Fedora má tuto úpravu ve svých jádrech zahrnutu automaticky. (Potom ale samozřejmě nefungují ani detektory rootkitu závislé na čtení z /dev/kmem.) Existuje také celá řada drobných modifikací jádra, které ztěžují útočníkům zavedení rootkitu, např. St. Michael[23], tripwire pro tabulku systémových volání [6] apod.

BSD systémy ztěžují práci útočníkům zavedením bezpečnostních úrovní, které neumožňují manipulaci s jádrem bez rebootu systému. I tato ochrana však není neprůstřelná [2].

3 Microsoft Windows

Ačkoliv je slovo rootkit přímo odvozeno od slova root, což je administrátor systému UNIX, není princip rootkitu omezený pouze na operační systémy typu UNIX.

Rootkity pro operační systém Windows také prošly vývojem od jednodušších ke složitějším. První generace rootkitů ve formě trojských koní systémových utilit se pod Windows příliš neprosadila, druhá generace spočívající v modifikaci aplikace v paměti nebo některých částí operačního systému (např. přilinkování jiných funkcí či úprava tabulky systémových volání) je již rozšířena zcela běžně. Třetí generace rootkitů jde ještě hlouběji do operačního systému a modifikuje přímo dynamické struktury jádra bez volání funkcí jádra (Direct Kernel Object Manipulation – DKOM). Čtvrtá generace rootkitů navíc umí díky trikům se správou paměti skrýt svůj kód v paměti před detektory rootkitů [13].

Pojďme se podívat, jakým způsobem může rootkit pod Windows například skrývat soubory [14]: po spuštění příkazu „`dir`“ se využijí funkce FindFirstFile a FindNextFile v knihovně kernel32.dll. Při linkování knihovny se používá importní adresní tabulka, jejíž manipulací je možné docílit změny funkčnosti daných funkcí (takto skrývá soubory např. rootkit Mersting [29]). Jiné rootkity (např. Vanquish [20]) přímo mění volání API v paměti procesu tak, aby se volal kód rootkitu. Funkce rozhraní Win32 FindFirst(Next)File při své činnosti volají funkci NtQueryDirectoryFile z knihovny ntdll.dll a zde mají rootkity další možnost modifikace funkčnosti (například rootkit Hacker Defender [21]).

NtQueryDirectoryFile slouží ke spuštění systémového volání a zde se tak dostáváme z kódu běžícího v uživatelském režimu ke kódu jádra. V jádře je nejprve použita tabulka systémových volání (System Service Dispatch Table – SSDT); tu ke skrývání souborů modifikuje například ovladač, který je součástí keyloggeru ProBot [22]. Dále jsou využity služby správce vstupně výstupních operací a ovladačů souborových systémů, kam je také možné vložit specializované ovladače, jejichž funkcí bude skrývat některé soubory (zde typicky působí ovladače, které jsou součástí komerčních skrývacích programů jako např. Hide Folders XP [19]). Hledání škodlivého kódu (programů) mají pod Windows na starost především antivirové programy. Ty procházejí procesy v paměti a soubory na disku a hledají v nich známé (a pomocí heuristik i neznámé) vzory. Protože však rootkity skrývají svůj kód v paměti i na disku

před uživatelskými procesy (jakými jsou i antiviry), klasické antivirové programy rootkity obvykle nenajdou. Pro hledání rootkitů v operačním systému Windows existuje celá řada drobných utilitek, které s větší či menší úspěšností hledají známé projevy rootkitů (změněné tabulky systémových volání, seznamy procesů apod.). Mezi takové prográmky patří například Kernel Hidden Process/Module Checker [7], F-secure blacklight [30], Patchfinder [31], VICE [32]. V blízké budoucnosti zřejmě bude tato činnost přímo součástí antivirových programů.

4 Odhalení vlastní zbraní

S rostoucí rafinovaností rootkitů je jejich detekce stále náročnější. Jestliže útočník upraví jádro, jsou možnosti jeho odhalení pomocí programů běžících v uživatelském režimu jen velmi omezené. Skutečnost, že rootkit něco ukrývá (ať už jsou to soubory, adresáře, položky registrů, procesy či jiné struktury operačního systému), je však možné lehce obrátit proti němu a využít toho při detekci [15]. Pro srovnání potřebujeme nestranný pohled na systém. Při srovnávání sledujeme rozdíl v seznamu souborů, adresářů, záznamů registrů apod. z pohledu potenciálně modifikovaného jádra a z pohledu nestranného. Získat nestranný pohled na soubory (včetně souborů s registry) můžeme buďto přímým čtením fyzického zařízení (protože však budeme používat potenciálně nespolehlivé jádro, nemusí být výsledek vždy přesný – zatím využíváme toho, že rootkity toto chování neupravují) nebo zavedením čistého jádra (zde však díky určitému časovému odstupu může dojít k řadě legitimních změn).

Nabootování čistého jádra spočívá v případě Windows ve využití bootovacího CD s Windows PE [24], záchranného (případně i Live) CD v případě unixových systémů nebo vložení disku do jiného počítače s čistou (nemodifikovanou) verzí operačního systému. Pro získání seznamu souborů můžeme použít běžný příkaz `ls` či `dir` a s vytvořením rozdílového seznamu nám pomůže program diff (windiff). Microsoft Research již na výzkumu v této oblasti pracuje, samostatný produkt (nazvaný GhostBuster [14]) však zatím není volně k dispozici a funkčnost analýzy rozdílů bude zřejmě časem přidána do programu Microsoft Windows Defender. Na podobném principu pracuje i program RootKitRevealer (ten porovnává výsledek vysokoúrovňového API s nízkoúrovňovým přístupem k disku) [33].

Pro nestranný pohled do paměti nám však přebootování nepomůže a musíme využít speciální hardware. Potřebujeme přídavné zařízení, které umí přečíst obsah paměti pomocí DMA (direct memory access – přímý přístup do paměti) přímo z paměťových čipů bez pomoci procesoru a operačního systému [15, 9]. V takto získaném výsledku je pak možné vyhledávat vzory rootkitu či jiné anomálie.

5 Budoucnost

Rozvoj rootkitů pokračuje mílovými kroky. Rootkity ve formě trojských koňů patří již dávno do muzea. Nejen rootkity, ale i jejich detekce již dnes patří do jádra operačního systému. Detekce rootkitů je dnes zaměřena především na skutečnost, že rootkit něco skrývá. Tím něčím bývají soubory (včetně položek registrů), procesy a síťová spojení. Ačkoliv skrývání opravdu patří do funkcionality dnešních rootkitů, nemusí tomu tak být navždy. Důvod proč rootkit skrývá soubory a položky registrů typicky souvisí s požadavkem na znovuzavedení rootkitu po restartu (rootkit samotný tak musí být uložen v permanentní paměti tj. na disku a dále musí být zajištěno jeho znovuspuštění po startu tj. ve startovacích skriptech nebo registrech). Pokud však takový požadavek mít nebudeme (například jakmile zjistíme že stroj byl rebootován, tak jej znovu nakazíme), tak ani nepotřebujeme skrývat nic na disku. Co se týká skrývání procesů, není třeba pro činnost rootkitu vytvářet nové procesy, je přece možné danou funkcionalitu vložit rovnou do modifikovaného jádra nebo vnést do existujících legitimních procesů. To vše v kombinaci s pasivní komunikaci umožní rootkitů podstatně vyšší úroveň neviditelnosti, než je tomu dnes [35]. Všechny tyto prvky však již byly jako proof-of-concept představeny, je tedy jen otázkou času, kdy se dostanou do skutečných rootkitů, pokud se tomu tak již nestalo...

Literatura

[1] BRUMLEY, D. Invisible intruders: rootkits in praktice, *;login:,* září 1999, `http://www.usenix.org/publications/login/1999-9/features/rootkits.html`.

[2] EREN, S. Smashing The Kernel Stack For Fun And Profit, *Phrack*, roč. 0x0b, č. 0x3c, `http://www.phrack.org/show.php?p=60&a=6`.

[3] „Holy_Father“ (pseudonym). *How to become unseen on Windows NT*, `http://rootkit.host.sk/knowhow/hidingen.txt`.

[4] HÝSEK, J. Úvod patchovaní systémových volání v Linuxu (LKM), *Blackhole*, `http://www.blackhole.sk/readme.php?id=166`.

[5] JONES, A. R. A Review of Loadable Kernel Modules, *SANS Security Essentials*.

[6] JONES, K. Loadable kernel modules, *;login:*, roč. 26, č. 7, 2001, `http://www.usenix.org/publications/login/2001-11/pdfs/jones2.pdf`.

[7] KEONG, T. CH. *Win2K Kernel Hidden Process/Module Checker*, `http://www.security.org.sg/code/kproccheck.html`.

[8] „OpioN“ (pseudonym), Kernel Rootkits Explained, *IT observer*, 26. března 2003, `http://www.ebcvg.com/articles.php?id=124`.

[9] PETRONI, N. L., ET AL. Copilot – a Coprocessor-based Kernel Runtime Integrity Monitor, In *Proceeding from Usenix Security Symposium*, srpen 2004.

[10] „plaguez“ (pseudonym), Weakening the Linux Kornel, *Phrack*, roč. 8, č. 52, 1998.

[11] „sd“ (pseudonym), patchovani linuxoveho /dev/kmem, prielom #17, 27. 2. 2002, `http://hysteria.sk/prielom/`.

[12] „sd, devik“ (pseudonymy), Linux on-the-fly kernel patching without LKM, *Phrack*, roč. 0xb, č. 0x3a, prosinec 2001.

[13] SPARKS, S., HITLER, J. Raising The Bar For Windows Rootkit Detection, *Phrack*, roč. 0xb, č. 0x3d.

[14] WANG, Y., et al. Detecting Stealth Software with Strider GhostBuster, *Microsoft Research Technical Report*, MSR-TR-2005-25.

[15] WANG, Y., et al. Strider GhostBuster: Why It's A Bad Idea For Stealth Software To Hide Files, *Microsoft Research Technical Report*, MSR-TR-2004-71.

[16] ZHOU, J., QIAO, L. *Backdoor and Linux LKM Rootkit – smashing the kernel at your own risk*, `http://eva.fit.vutbr.cz/~xhysek02/syscalls/020129lkm.htm`.

[17] ŽILKA, R. *Analýza a detekce linuxových rootkitů.* bakalářská práce FI MU, 2005.

[18] `http://www.chkrootkit.org/`.

[19] `http://www.fspro.net/downloads.html`.

[20] `http://www.rootkit.com/project.php?id=9`.

[21] `http://rootkit.host.sk/`.

[22] `http://www.nethunter.cc/index.php?id=14`.

[23] `http://sourceforge.net/projects/stjude`.

[24] http://www.microsoft.com/licensing/programs/sa/support/winpe.mspx.

[25] http://www.rootkit.nl/projects/rootkit_hunter.html.

[26] http://www.s0ftpj.org/en/site.html.

[27] http://www.stud.fit.vutbr.cz/~xhysek02/.

[28] http://www.s0ftpj.org/tools/tools.html.

[29] http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=39113.

[30] http://www.f-secure.com/blacklight/.

[31] http://www.rootkit.com/project.php?id=15.

[32] http://www.rootkit.com/project.php?id=20.

[33] http://www.sysinternals.com/utilities/rootkitrevealer.html.

[34] http://en.wikipedia.org/wiki/Rootkit.

[35] Mark's Sysinternals Blog: Sony Rootkits and Digital Rights Management Gone Too Far, http://www.sysinternals.com/blog/2005/10/sony-rootkits-and--digital-rights.html

[36] Joanna Rutkowska: Rootkit Hunting vs. Compromise Detection http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Rutkowska/BH-Fed-06-Rutkowska-up.pdf

Odolnost kryptografického hardwaru s ohledem na nasazení

Vašek Lorenc, Vašek Matyáš

E-mail: lorenc@fi.muni.cz, matyas@fi.muni.cz

Abstrakt

Příspěvek představí některé zajímavé poznatky a závěry srovnávací analýzy různých typů kryptografických zařízení, a to především s ohledem na bezpečné provádění aplikací a vhodnost jejich nasazení. Analýza pokrývá čipové karty, mikrokontrolery a hardwarové kryptografické moduly a posuzuje schopnosti jednotlivých zařízení odolávat útokům jak po stránce fyzické, tak i logické. Základní typy útoků jsou blíže popsány a je rozebrána jejich náročnost a proveditelnost na uvedených typech zařízení, včetně nejčastěji používaných obran.

1 Úvod

Se vzrůstajícím významem bezpečnosti, šifrování a elektronických podpisů se objevuje velké množství otázek a problémů spojených s bezpečným uložením kryptografického materiálu. Vyrábějí se tedy různé varianty čipových karet, USB tokenů nebo jiné, vysoce specializované moduly, jejichž výrobci deklarují určitou schopnost uchovat tajné informace ve formě nepoužitelné mimo dané zařízení. V soudobých počítačích se začínají používat i tzv. *Trusted Platform* moduly [11]. Všechna tato zařízení a technologie mají za úkol ochránit důvěrná data před neoprávněnou manipulací.

Nabízí se tedy otázka, zda-li tomu tak opravdu je a do jaké míry se jednotlivé typy těchto zařízení od sebe liší právě v míře poskytovaného bezpečí vůči útočníkům. Tento příspěvek se snaží stručně představit používané třídy zařízení, typy existujících útoků proti nim a používané možnosti obrany.

2 Třídy zařízení

Ne všechna zařízení jsou stejně odolná vůči stejnému typu útoku. Aby bylo možné alespoň trochu sjednotit pohled na typy používaných kryptografických zařízení, zavedeme si členění na následující třídy:

- Mikrokontrolery.
- Čipové karty – plastická karta standardizované velikosti obsahující integrovaný obvod (čip), který slouží k zpracování dat a/nebo jejich uložení. Základní dělení čipových karet je na:
  - paměťové – použitelné pro ukládání dat, nemají významné bezpečnostní schopnosti;
  - procesorové (smartcard[13], supercard) – s procesorem, operačním systémem, aplikačním softwarem a paměťovým prostorem pro data;
  - kryptografické – od procesorových mají navíc i dodaný kryptografický koprocesor.
- Hardwarové kryptografické moduly – (dále jen HSM, Hardware Security Module) vytváří prostředí pro bezpečné zpracování dat tím, že data jsou zpracována v důvěryhodném modulu, který je chráněn před externími hrozbami. Obvykle provádí generování, uchování a ochranu kryptografických klíčů. Mezi známé výrobce HSM patří například IBM, Chrysalis-ITS, Eracom nebo nCipher.

Vlastnosti těchto skupin zařízení se často projevují nejen v odlišné schopnosti reagovat nebo odolávat pokusům o narušení, ale také v atributu mnohem jednodušším na rozpoznání – v ceně.

Typ zařízení	Přibližné rozmezí cen/kus
Mikrokontroler	10^1–10^2 Kč
Čipová karta	10^2 Kč
HSM	10^4–10^5 Kč

3 Bezpečnost, druhy útoků

Pro potřeby tohoto příspěvku budeme rozlišovat útoky na základě nutnosti mít kartu fyzicky k dispozici. Podle toho je možné rozčlenit útoky na *vzdálené* a *lokální*.

3.1 Vzdálené útoky

Vzdálené útoky budeme dělit do tříd podle podobnosti typu chyby, které zneužívají. Toto dělení je převzato z [12] – útoky na klíče a jejich integritu, nedostatečnou kontrolu parametrů při práci s PINy a útoky na nevynucení politiky.

- *Útoky na klíče* – využívají vlastností pro zajištění kompatibility se staršími zařízeními nebo znalosti vztahů mezi klíči, kdy při (částečné) znalosti klíče je možné odvodit nebo dopočítat jiný (citlivější) klíč. Dalším problémem může být chybějící (horní) omezení na počet generovaných klíčů – pak je možné zařízení využít kárychlému vygenerování velkého množství klíčů, které použijeme pro útok (tj. nalezení jednoho z klíčů).
- *Nedostatečná kontrola parametrů* – je možné definovat jako neočekávanou posloupnost transakcí, jejímž cílem je oklamat bezpečnostní modul tak, aby zpřístupnil tajemství způsobem, který odporuje bezpečnostní politice zařízení. Mezi zástupce těchto útoků patří útok s decimalizační tabulkou nebo útoky využívající možností odvodit části PINu nebo najít vhodnou kolizi.
- *Nevynucení politiky* – zneužívá API, která neobsahují a nevyhodnocují žádnou bezpečnostní politiku – např. PKCS #11, které je navrženo jako rozhraní mezi aplikacemi a jednouživatelskými bezpečnostními zařízeními, a přesto bývá často používáno jako hlavní API i u mnohých kryptografických modulů.

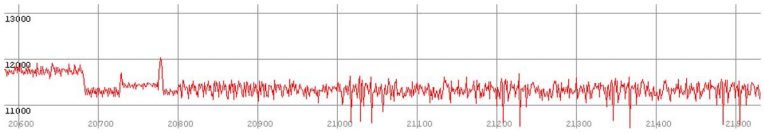
Kryptografická zařízení jsou k chybám v API bohužel náchylná, ať již z důvodu složitosti či rozsáhlosti rozhraní (běžně obsahujícího desítky až stovky funkcí).

3.2 Lokální útoky

Máme-li kartu, mikrokontroler nebo HSM fyzicky v držení a k dispozici k dalším experimentům, nabízí se poněkud širší spektrum možných přístupů k získání klíčového materiálu. Podle finanční náročnosti i obtížnosti provedení je možné tyto přístupy rozčlenit na následující tři skupiny – invazivní techniky, neinvazivní a poměrně nové semi-invazivní útoky.

Invazivní útoky obvykle spočívají v umístění sond přímo na sběrnici napadeného zařízení, aby bylo možné číst důvěrná data přímo uvnitř zařízení. V současné době jsou tyto útoky možné jen ve specializovaných a kvalitně vybavených laboratořích a je možné u nich rozeznávat následující čtyři procesy:

- *Příprava vzorku*, kdy dojde k odstranění ochran a obalu kolem čipu za pomoci kyselin nebo jiných specializovaných roztoků nebo mechanických postupů.



Obr. 1 Příklad odběru proudu při různých operacích na kartě

- *Reverse engineering*, postup používaný při zjišťování vnitřích struktur polovodičového čipu a učení se nebo emulování jeho funkcionality. Od útočníka vyžaduje tutéž technologii, jakou mívají k dispozici výrobci polovodičových prvků a současně mu dává k dispozici tytéž možnosti.

- Techniky *mikrosondáže* mohou být použity k přímému přístupu k povrchu čipu. S jejich pomocí je pak možné sledovat, manipulovat a ovlivňovat chování integrovaného obvodu.

- Po zjištění chování a vlastností čipu je následně možné provést jeho *modifikaci*, aby se choval tak, jak vyžaduje útočník a ne tak, jak zamýšlel výrobce. Častým cílem modifikace bývá možnost přímého čtení paměti zařízení.

Útoky **neinvazivní** využívají skutečnosti, že ne všechna zařízení bývají navržena s dostatečnou odolností vůči extrémním výkyvům či vlivům okolního prostředí. Takovéto stavy pak mohou způsobovat, že se obvody začnou chovat chybně a dochází k únikům kryptografického materiálu. Ačkoliv je tu jisté riziko zničení testovaného obvodu, zpravidla nezanechávají neinvazivní metody žádné stopy po své činnosti.

Mezi zástupce neinvazivních útoků patří:

- chybové útoky (*clock/power glitches*) – útoky využívající nesprávné chování hardwaru v případě neočekávaných hodinových signálů nebo nepravidelností v napájení;

- časové útoky (*timing attacks*);

- útoky hrubou silou (*brute force attacks*);

- výkonová analýza a odvozená výkonová analýza (*simple power analysis*, *differential PA*) [2], [9], [7] jsou techniky diskutované zejména v posledních letech; je využito skutečnosti, že paměťové obvody i kryptografické čipy mají pro různé operace s různými čísly také různé odběrové charakteristiky (obr. 1 – měření odběrů při různých operacích však vyžaduje specializovaný hardware, který není běžně dostupný, což samotný útok omezuje na znalejší útočníky s většími zdroji);

- datové remanence – zhruba v roce 1980 se zjistilo, že nebezpečná teplota pro SRAM čipy, při které je možné snadno vyčíst data v ní nich uložená po dobu jedné minuty i po odpojení těchto čipů od napájení, je přibližně $-20\,^{\circ}\mathrm{C}$; tohoto faktu je následně možné využít při čtení dat bez vědomí zařízení.

Poslední skupinou útoků jsou útoky **semiinvazivní** [1]. Při nich zpravidla dochází k odstranění ochranného obalu čipu, místo aplikace mikrosond a reverse engineeringu se k modifikaci chování čipu používá nejčastěji některého z relativně dostupných záření – UV, mikrovlnného nebo za ionizovaného.

Jsou tak realizovány útoky na testovací části čipů, např. na paměťové pojistky, kdy je po jejich znovuaktivaci možné snadné vyčtení obsahu paměti bez účasti procesoru jeho bezpečnostních mechanismů.

4 Možnosti obrany

Jsou-li známy postupy, jak útoky vést, je možné začít se proti nim i proti jejich variantám bránit. V této části budou nastíněny postupy použitelné pro obranu jednotlivých zařízení.

4.1 Vzdálené útoky

Nejprve stručně nastíníme možnosti obrany před vzdálenými (softwarovými) útoky.

Od prvních existujících rozhraní pro práci s kryptografickými zařízeními prošly jejich softwarové části značným vývojem, objevily se lepší metody řízení přístupu k datům (například pomocí rolí), od monolitického návrhu se postoupilo k práci s objekty, definovanými čítači a paměťovými úložišti.

Dle [3] se však stále dají hledat lepší postupy – vhodnější návrhy Security API, lepší ověřování typů dat/operací, pečlivější kontrola přístupů.

Jednoznačně nejlepším by bylo mít softwarovou část zařízení s formálním důkazem správnosti. Verifikace kódu je však komplikovaná záležitost a automatizované techniky jsou v současnosti použitelné spíše na kratší bloky kódu.

To však navádí k první důležité zásadě – vytvářet API po malých, nějakým způsobem ověřitelných částech. Tomu by mohla napomáhat kvalitní analýza, jejímž výstupem by byla minimalizovaná sada nutných funkcí. Současně je vhodné v ideálním případě využívat co nejvíc explicitních specifikací typů dat (klíče, uložené uživatelské informace, datová úložiště), zdrojů a cílů operací nad daty, a to vše při současném zachování jednoduchosti.

Pro lepší pochopení a zvládnutí celého systému je vhodné také pro potřeby vývoje vytvářet grafické prezentace typového systému, zejména pak toků dat

mezi těmito typy uložišť – analýza toků dat a klíčového materiálu může pomoci zabránit mnoha budoucím problémům s únikem informací.

Při kontrole přístupů je možné uvažovat autorizaci nejen při přístupu k datům, ale i při vyvolávání jednotlivých operací API. V té souvislosti je možné v dalším kroku definovat i důvěryhodné cesty (*trusted paths*), které by zabránily vkládání nepovolených operací do již ustanovených relací při komunikaci se zařízením.

Vlastní ověření funkčnosti implementovaného API přitom nemusí probíhat pouze formou formální verifikace, ale i pomocí sad operací s náhodnými vstupy pro jednotlivé funkce a následným testováním výsledků

Komplikovaná a monolitická API jsou zdrojem mnoha zbytečných problémů, je však možné, že s rostoucími nároky se komplexním návrhům nebude možno vyhnout – dá se předpokládat, že s mohutnějším nasazením *Trusted Computing* dojde k vývoji API směrem k rozhraním mezi objekty/třídami. Řešením by mohly být vhodné nástroje umožňující tvorbu takovýchto rozsáhlejších a při současném zachování bezpečnostních vlastností.

4.2 Lokální útoky

Obrana před lokálními fyzickými útoky se může různit podle možností jednotlivých typů zařízení – jiné způsoby obrany jsou vhodné pro HSM, a jiné pro čipové karty, už z důvodu jejich různých velikostí i způsobů užití.

Mikrokontrolery jsou z pohledu bezpečnosti nejslabší, neboť většinu z nich je poměrně snadné napadnout některou z popsaných praktik, často i s poměrně nízkými náklady [1].

Čipové karty mají typicky malé rozměry, jsou napájené jen v okamžik, kdy se s nimi pracuje, a útočník má tedy možnost pracovat nerušeně na jedné části čipu, aniž by narušil data uložená v jiné části. I přesto poskytují mnohem větší ochranu před nejrůznějšími útoky než mikrokontrolery. Útoky na současné čipové karty bývají velice drahé a vyžadují speciální vybavení.

Hardwarové kryptografické moduly mají větší rozměry a tím i další možnosti zabezpečení např. větší tranzistory pro filtrování datově závislých signálů od externích vlivů. Interní napájení dovoluje neustálý dohled nad stavem zařízení a v případě narušení pak zničení citlivého kryptografického materiálu.

V [3] jsou uvedeny následující způsoby detekce a obrany průniku používaných u hardwarových bezpečnostních modulů, rozdělené na pasivní, aktivní a pokročilé.

- Pasivní techniky mají za úkol zejména ztížit pokus o narušení, nebo alespoň usnadnit jeho detekci. Jedná se buď o starší techniku uzavírání do ocelového pouzdra (vhodné pro nemobilní zařízení, kdy sama váha obalu může znatelně komplikovat manipulaci) nebo zalití do dalšího obalu.

- Aktivní techniky už využívají skutečnosti, že jsou HSM zpravidla napájeny nepřetržitě a mohou tedy při detekci pokusu o útok aktivně vyvíjet další aktivitu, jako například mazání klíčového materiálu. K aktivním způsobům obrany patří vodivé membrány, do kterých se klíčové části často chemicky obalují, teplotní a rentengová čidla, čidla výkyvů napětí a také spínače ve víku zařízení. Vzhledem k tomu, že u HSM je velice důležité i správné generování náhodných čísel, často se k nim připojuje modul na monitoring statistických vlastností právě generovaných posloupností.
- Mezi pokročilé techniky obrany hardwarových kryptografických modulů je možné řadit užívání vzduchových kapslí s volitelným tlakem a miniaturních náloží schopných bezpečně zničit úložiště dat při detekci pokusu o narušení.

Další možností ochrany dat uvnitř zařízení, tentokrát již na úrovni paměťových čipů, může být snížení rizika úniku informací pomocí *datových remanencí*. K tomu se běžně doporučuje nepracovat s kryptografickými klíči, hesly a dalšími citlivými informacemi v SRAM, přesunovat je mezi lokacemi v paměti a provádět bezpečné mazání původních míst. Dále je vhodná kombinace SRAM čipů s obvody na měření okolní teploty, společně s nějakým dalším, reakčním obvodem.

Je třeba pamatovat i na skutečnost, že některé paměťové čipy bývají až příliš inteligentní. Z důvodů rychlejších operací nebo případných eliminací částečných chyb udržují části dat na více místech současně, což může vést k dalším únikům dat. Něco podobného platí i pro souborové systémy, kde při mazání souborů dochází spíše k odstranění ukazatelů z tabulek než k fyzickému promazání dat.

Potěšující je jistě zpráva, že díky technologickému pokroku je dobývání dat ze soudobých zařízení s vysokou hustotou ukládání stále obtížnější. V kombinaci s šifrováním kdekoliv, kde je to možné, opět pomáhá zvýšit bezpečnost celého řešení. Naopak nepříjemná v tomto směru je studie [2], kde se autor zaobíral testem vzorků moderních SRAM čipů různých výrobců. Ověřil tak, že soudobé paměťové SRAM čipy jsou velice různorodé, a to jak typy od různých výrobců (kde se rozmezí nebezpečných teplot pohybovalo od běžné pokojové až po $-50\,^{\circ}\mathrm{C}$), tak byly zjištěny i významné rozdíly v měření charakteristik kusů z jedné série jednoho výrobce.

Proti UV útokům je možné se bránit kombinací paměťových obvodů s materiály citlivými na UV záření taková, že nadměrné UV záření dokáže bezpečně zničit veškerá data v paměťovém čipu uložená.

Obrana vůči *SPA/DPA útokům* je soudobým problémem, kterému je věnována spousta úsilí. Ideálním by byl stav, kdy jednotlivé operace nejsou rozlišitelné na pomocí měření spotřeby. Tomu ve velké míře napomáhají integrované kryptografické procesory, jejichž spotřeba se při různých operacích liší jen velice mírně; občas se je však nevyhnutelné využívat i čistě softwarovou implementaci. V takových případech se, je-li to možné, využívá modifikace kódu tak, aby

jednotlivé větve výpočtu byly co nejvíce podobné z hlediska spotřeby, což je nelehká úloha. Zhruba nejčastější je však využívání pseudonáhodného záměrného přidávání šumů, přeskládávání, přidávání a různé časové průběhy operací.

Unmarking, remarking and repackaging – mluví-li se o bezpečnosti mikrokontrolerů, přichází v úvahu i metody ve stylu *security by obscurity*. Čipy jednotlivých výrobců totiž bývají často víceméně známé a snadno podrobitelné bližší analýze. Výrobci bezpečnostních mikrokontrolerů tedy často používají metody mazání jakýchkoliv informací z obalu čipu, záměrné falšování údajů tam uvedených (například vydávání za specializované ASIC obvody pro zdání vyšší složitosti čipu), nebo vkládání čipů do jiných pouzder, než jaká bývají od výrobce obvyklá. Poslední možnost navíc dává šanci změnit orientaci čipu v pouzdře a ještě více zmást případného útočníka. Nevýhodou těchto obran je skutečnost, že fungují jen po dobu utajení a proti neznalým útočníkům.

Protiopatření vůči *chybovým útokům* mohou být realizována jak softwarem, tak i pomocí hardwaru. Obecně pak pomáhají obvodům detekovat nebo rovnou opravovat indukované chyby. I tentokrát je možné rozlišovat obrany pasivní a aktivní, nebo za pomoci redundancí. Aktivní obrany odpovídají zhruba tomu, co se používá u HSM – čidla světla či jiných záření, případně fólie, do níž je obvod zabalen. Pasivní jsou zpravidla realizovány záměrnými (pseudonáhodnými) změnami chodu systémových hodin, společně se šifrováním pamětí i sběrnice.

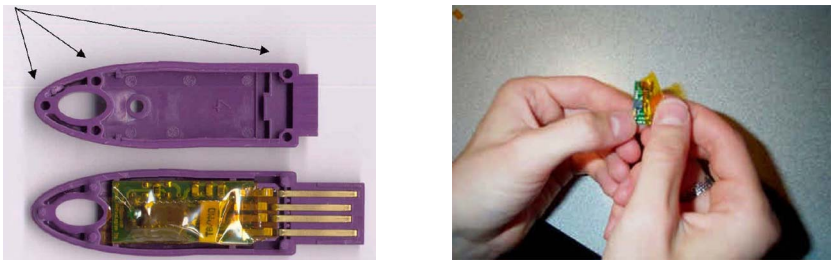
Jako nejúčinnější se však prokazuje být využití redundance, a to jak v násobných přístupech do paměti, tak při opakování výpočtu. Využívají se opravné kódy pro detekci/opravu chyb, časové nebo prostorové redundance, naivní metody v podobě jednoduché nebo násobné duplikace dat. Ačkoliv má výrazný vliv na praktické používání, neboť dochází k dramatické redukci výkonu a výpočet se stává značně neefektivním, jedná se překvapivě stále o velice účinné řešení. Další postupy s vyšší efektivitou byly navrženy zejména pro práci s blokovými šiframi. Technologicky je možné řešit obranu proti chybám i pomocí tzv. zdvojených (dual-rail) obvodů.

5 Příklad útoku

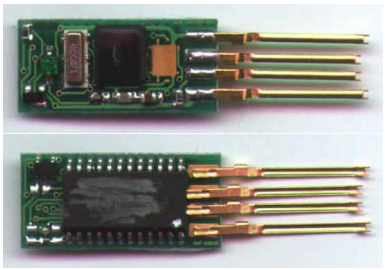
Že nejde jen o teoretické možnosti, ale že i existující zařízení jsou náchylná k popsaným typům útoků, dokazuje například studie [6] z roku 2001.

Autoři vzali tři typy zařízení, konkrétně USB tokeny Alladin Knowledge Systems eToken R1, Rainbow Technologies iKey 1000 a iKey2000 a podrobili je široké škále útoků; jak fyzických, tak i na bezpečnost implementace API.

Až na jedinou výjimku byla zařízení snadno rozebratelná bez známek narušení, onou výjimkou byla tenká folie kolem jednoho ze zařízení (obr. 2), která se však dala snadno odstranit a nebránila dalšímu útoku. Kvůli nevhodně chráněným externím EEPROM pamětem bylo možné vyčítat informace na zařízení,



Obr. 2 Preparace zařízení, odstranění obalu – iKey2000



Obr. 3 Skrytí informací o čipu – iKey2000

případně je i měnit. U modelu iKey2000 se snažil výrobce zamaskovat informace o použitém čipu, byť nepříliš úspěšně (obr. 3).

Poměrně zajímavým typem zjištění byl reset uživatelského PINu bez toho, že by došlo k vymazání dalších uživatelských dat – bylo pak možné snadno používat cizí klíče bez znalosti jeho PINu.

Po softwarové stránce byly zmíněny, byť nevyzkoušeny, techniky platné pro USB zařízení – vyhledání a používání nezdokumentovaných příkazů zařízení a zasílání nekorektních USB paketů.

Důležitý je pak závěrečný verdikt, v kterém je uvedeno, že i „důvěryhodné“ zařízení může dopadnout v přímém kontaktu se znalým útočníkem velice špatně. Při ztrátě některého z uvedených tokenů je doporučeno data považovat za vyzrazená.

6 Shrnutí náročnosti útoků

Odolnost vůči narušení není absolutní. Úroveň odolnosti vůči narušení konkrétního produktu je odvozena od času, peněz a úsilí, které musí útočník vynaložit. Vyčíslení toho je velmi důležité, i když je mu věnováno méně pozornosti, než by bylo třeba.

Tabulka 1 Přehled náročnosti lokálních útoků na jednotlivé třídy zařízení

Typ útoku	Mikrokont.		Čipové karty		HSM	
	út.[a]	čas[b]	út.	čas	út.	čas
Invazivní útoky:						
preparace čipu	≥ 1	hod.	≥ 1	hod.	≥ 3	měs.
rekonstrukce a analýza čipu	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
testování čipu s využitím mikrosond	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
čtení paměti nebo modifikace čipu	$\geq 1,5$	dny	≥ 2	dny	≥ 3	měs.
Semi-invazivní útoky:						
UV záření	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
ozáření CMOS tranzistorů	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
mikrovlnné radiace	$\geq 1,5$	dny	–	dny	≥ 3	měs.
chybová analýza	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
Neinvazivní útoky:						
výkonová analýza	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
časové útoky	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
diferenciální elektromagnetická analýza	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
datové remanence	$\geq 1,5$	dny	$\geq 1,5$	dny	≥ 3	měs.
reverse engineering	≥ 1	týd.	≥ 1	týd.	≥ 1	měs.

[a] třída útočníka dle předchozí klasifikace

[b] dolní odhad času pro úspěšnou realizaci útoku

IBM rozděluje útočníky do 3 tříd [10]:

Třída 1 (*clever outsiders*, chytří nezasvěcení útočníci) – mnohdy inteligentní útočníci, kteří nemají dostatek znalostí o systému. Využívají cenově běžně dostupných nástrojů nebo služeb s možností zmapování principu fungování přístroje v dohledné době. Obvykle zkouší útočit na známé bezpečnostní slabiny, nevyhledávají nové.

Třída 2 (*knowledgeable insiders*, zasvěcení insideři) – zkušení jedinci nebo týmy s nákladným a sofistikovaným vybavením, se kterým jsou schopni provést analýzu systému v dostatečném čase. Mají úzce specializované technické vědomosti a zkušenosti, různě hluboké pro jednotlivé části systému s možným přístupem k většině z nich.

Třída 3 (*funded organizations*, majetné organizace) – vysoce kvalifikované týmy využívající zařízení, která nejsou běžně dostupná na trhu. Mohou provádět

detailní analýzy systému, navrhovat komplexní útoky a využívat nejmodernější analytické nástroje. Příkladem jsou např. vládní organizace (NSA), které mají pro své aktivity značné finanční zabezpečení.

Je třeba poznamenat, že většina publikovaných útoků pochází od autorů spadajících do třídy 1. Zajímavé by byly výsledky při znalosti interních informací o HSM, nebo čipových kartách. Zvažujeme-li navíc i metody útoků přes remote API a chceme-li lépe rozlišit schopnosti odolávat napadení u jednotlivých skupin zařízení, je možné ještě použít jemnější dělení přidáním následujících tříd útočníků:

Třída 0 (*script kiddies*) – do této třídy řadíme útočníky, kteří nedisponují dostatkem znalostí o systému a využívají komukoliv běžně dostupné předpřipravené nástroje a postupy pro útoky na známé zranitelnosti. Ty jsou vedeny zkoušením těchto nástrojů metodou pokus-omyl nebo čistě náhodným necíleným zjištěním zranitelnosti (např. reakce zařízení na krátký a intenzivní výboj světla).

Třída 1.5 – jedná se o osoby nebo skupiny osob s možnostmi na rozhraní mezi 1. a 2. třídou. Útočníci jsou inteligentní, se základními znalostmi systému. Využívají cenově dostupných nástrojů, útočí na známé chyby a snaží se hledat i nové slabiny. Např. specializovaná pracoviště univerzit.

V tabulce 1 se jedná o dolní odhady náročnosti, jednotlivé typy útoků se mohou i nadále komplikovat – např. použitím vhodnějších typů pamětí u mikrokontrolerů, větším stupněm integrace čipů nebo některou z dalších technik zmiňovaných dříve.

Některá zkoumaná zařízení byla dokonce natolik málo odolná vůči semiinvazivním útokům, že i pouhé rozsvícení intenzivní zářivky v blízkém okolí mikrokontroleru mělo za následek jejich nesprávnou funkci.

7 Závěr

Jak je vidět, na tradiční kryptografická zařízení již v současné době existuje celá řada více nebo méně úspěšných útoků s různým stupněm náročnosti provedení. Jistě tedy není od věci se ptát, zda-li se zvyšující se cenou zařízení rostou i náklady na úspěšný útok. Dle dostupných dokumentů se dá konstatovat, že není-li v systému obsažena zásadní chyba v návrhu, odpovídá cena zařízení i odhadu jeho bezpečnosti při správě tajných informací.

Přitom diskutovaná problematika se týkala jen případů, kdy se útočilo na tajné informace uložené uvnitř takového přístroje. Jiné možnosti útoků (např. nedůvěryhodné prostředí při práci s čipovou kartou nebo USB tokenem) jsou předmětem dalších výzkumů a jistě také velice zajímavým zdrojem hrozeb.

Dalším směrem, kterým by se bezpečnost zařízení mohla ubírat, jsou možnosti zlepšování fyzické bezpečnosti čipových karet. Techniky útoků na čipové

karty v současné době využívají toho, že je čipová karta závislá na externím napájení (které je možné záměrně vypnout či modifikovat) a také neexistuje důvěryhodný interface pro práci s čipovou kartou integrovaný na kartě samotné. Vyřešení těchto bodů by opět zvýšilo náročnost útoků.

Při výběru vhodné technologie je proto třeba zvážit několik kritérií – v neposlední řadě prostředí, do nějž by se měla zakoupená zařízení nasadit, stupeň bezpečnosti, který plánovaná aplikace vyžaduje, a peněžní prostředky dostupné pro realizaci. Jak vidno, jmenované faktory se vzájemně doplňují a významnou měrou určují dostupná zařízení i vlastnosti výsledného řešení.

Literatura

[1] SKOROBOGATOV, S. P. *Semi-invasive attacks – A new approach to hardware security analysis.* 2005.
http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-630.html

[2] SKOROBOGATOV, S. P. *Low Temperature Data Remanence in static RAM*

[3] BOND, M. *Understanding Security APIs.* 2004.
http://www.cl.cam.ac.uk/~mkb23/research/Thesis.pdf

[4] BOND, M., CVRČEK, D., MURDOCH, S. J. *Unwrapping the Chrysalis.* Technical Report, UCAM-CL-TR-592, University of Cambridge, 2004.

[5] ANDERSON, R., BOND, M., CLUOW, J., SKOROBOGATOV, S. P. *Cryptographic Processors – A Survey.* 2005.
http://www.cl.cam.ac.uk/~mkb23/research/Survey.pdf

[6] GRAND, J., Grand Ideas Studio. *Attacks on and Countermeasures for USB Hardware Token Devices.* 2001.
http://www.grandideastudio.com/files/security/tokens/usb_hardware_token.pdf

[7] ANDERSON, R., KUHN, M. *Low Cost Attacks on Tamper Resistant Devices.* 2001. http://www.cl.cam.ac.uk/~mgk25/tamper2.pdf

[8] ANDERSON, R., KUHN, M. *Tamper Resistance – a Cautionary Note.* 1996. http://www.cl.cam.ac.uk/~mgk25/tamper.pdf

[9] STANDAERT, F. X. (UCL), et al. *Electromagnetic Analysis and Fault Attacks: State of the Art.*

[10] ABRAHAM, D. G., DOLAN, G. M., DOUBLE, G. P., STEVENS, J. V. Transaction Security System In *IBM Systems Journal* v 30 no 2, 1991.

[11] *Trusted Platform Module specifications.*
https://www.trustedcomputinggroup.org/groups/tpm/

[12] CVRČEK, D., KRHOVJÁK, J., MATYÁŠ, V. *Hardwarové bezpečnostní moduly – API a útoky.* Europen, 2004.

[13] ISO/IEC 7816 Identification cards – Integrated circuit cards – Cards with contacts.

Zkušenosti s nasazováním HW tokenů pro uživatele *METACentra*

Daniel Kouřil, Michal Procházka

E-mail: kouril@ics.muni.cz, michalp@ics.muni.cz

1 Úvod

Řešení bezpečnosti v distribuovaných prostředích s velkým počtem uživatelů z různých organizací naráží na mnohé problémy, které nejsou v menších systémech viditelné. Zejména pro spolupráci, která přesahuje hranice států nebo i kontinentů jsou zapotřebí autentizační mechanismy, které jsou dostatečně flexibilní a škálovatelné, aby pokryly komplexní požadavky na bezpečnost celého systému. Velkou část těchto požadavků lze splnit pomocí infrastruktury veřejných klíčů (*Public Key Infrastructure – PKI*), která však zároveň přináší jiné problémy, zejména související s bezpečnou správou soukromých klíčů.

V tomto příspěvku popisujeme nasazení PKI do prostředí *METACentra*, které nabízí kapacity pro náročné výpočty, tzv. *grid*. V tomto projektu jsme pro řešení problematiky správy klíčů použili zařízení založená na technologii čipových karet. Představíme řešení, která jsme vyvinuli pro integrování PKI a autentizačního systému Kerberos a také další nezbytné technické i administrativní úkony, které jsme realizovali.

V první části příspěvku popíšeme *METACentrum* a jeho bezpečnostní architekturu. Dále představíme technologii čipových karet a hardwarových tokenů, jejich typy, vlastnosti a možnosti použití. V další části popíšeme proceduru pro výběr vhodného tokenu pro uživatele *METACentra* a následně integrování tokenů s prostředím *METACentru*. Příspěvek ukončíme představením dalších aplikací, které jsou provozovány v *METACentru* a využívají možnosti tokenů.

2 Projekt *METACentrum*

Projekt *METACentrum* je aktivita sdružení CESNET, které je provozovatelem české akademické vysokorychlostní sítě CESNET2 a provádí výzkum a vývoj v oblasti aplikací využívajících toto prostředí. Jednou z klíčových aplikací je

META Centrum, které nad sítí CESNET2 buduje a produkčně provozuje infrastrukturu pro realizaci náročných výpočtů, tzv. *grid*. Cílem projektu je vybudovat národní gridové prostředí, které skrývá rozdíly v jednotlivých zapojených systémech a vytváří uživatelům iluzi jediného výpočetního uzlu. *META Centrum* se také podílí na převážné většině všech národních i mezinárodních gridových projektů, které se v ČR řeší.

Projekt *META Centrum* spojuje výpočetní a diskové kapacity několika akademických výpočetních center v ČR do jednoho logického celku a v současné době nabízí svým uživatelům zhruba 450 procesorů, 25 TB distribuované diskové kapacity a 400 TB záložní páskové kapacity. *META Centrum* registruje několik stovek uživatelů z nejrůznějších vědeckých a inženýrských oborů.

META Centrum poskytuje distribuované prostředí přinášející mnoho výhod na straně efektivního využíváni hardware a software. Na druhé straně se ovšem otevírají nové problémy spojené s bezpečností, k jejichž řešení často nestačí jednoduše přejímat řešení používaná v uzavřených doménách a je potřeba hledat nové přístupy. Zajištění bezpečného výpočetního prostředí a zpracovávaných dat patřilo vždy k prioritám *META Centra*. Bezpečnostní infrastruktura *META Centra* je nedílně integrována s jeho základním prostředím a garantuje dostatečné zabezpečení zdrojů i uživatelů a současně minimalizuje explicitní autentizační požadavky na jednotlivé uživatele.

2.1 Bezpečnostní infrastruktura *META Centra*

Primární autentizační rámec v *META Centru* je postaven na systému Kerberos [1]. Jedná se o protokol určený k vytvoření vzájemně autentizovaného spojení mezi klientem a serverem, kteří spolu komunikují přes otevřenou síť. Autentizace je založena výhradně na silných kryptografických algoritmech a na symetrických šifrovacích klíčích, které každý uživatel či služba sdílí se serverem služby Kerberos (*Key Distribution Center – KDC*). Protokol je navržen tak, aby poskytoval bezpečnou autentizaci v prostředí, kde předem nelze zaručit bezpečný komunikační kanál, tj. protokol je odolný jak proti pasivním odposlechům, tak i proti aktivním útokům, které modifikují zprávy posílané mezi klientem a serverem. Vedlejším efektem autentizace protokolem Kerberos je ustavení šifrovacích klíčů mezi klientem a serverem. Tyto klíče mohou být použity pro ochranu vzájemné komunikace mezi oběma komunikujícími stranami. KDC používá centralizovanou databázi, která obsahuje záznamy o všech uživatelích a službách spravované administrativní domény, včetně jejich klíčů. V případě uživatelů jsou tyto klíče odvozeny od uživatelských hesel, pro služby jsou klíče vygenerovány náhodně.

Vlastní autentizace využívá *lístky* (*tickets*) vydávané službou KDC. Lístek je datová struktura, která obsahuje informace o majiteli lístku, době platnosti a službě, pro kterou je lístek určen. Z pohledu autentizace slouží lístek jako certifikát potvrzující identitu klienta. Myšlenka je podobná mechanismu certifikátů

veřejných klíčů ze světa PKI, rozdílem je to, že kerberovské lístky jsou vydávány adresně pro každou službu zvlášť, mají výrazně kratší dobu platnosti (zpravidla deset hodin) a jsou vytvářeny výhradně za použití symetrických šifrovacích algoritmů. Aby uživatelé nemuseli zadávat své heslo při přístupu ke každé službě, zavádí Kerberos tzv. Ticket Granting Ticket, což je speciální typ lístku, který se používá místo hesla pro autentizaci uživatele vůči KDC. Uživatel tak zadává své heslo pouze jednou pro získání lístku TGT a po dobu platnosti tohoto TGT lístku již nemusí své heslo zadávat znovu. Kerberos také nabízí mechanismus pro bezpečné přenášení lístků mezi více počítači v síti, takže uživatel se autentizuje pouze jednou při zahájení práce a infrastruktura pak již zajistí, že lístky jsou transparentně přeneseny na stroje, které uživatel použije. Uživatel tak má lístek stále přístupný i když přechází mezi více počítači. Kerberos takto výrazně usnadňuje práci uživatelům a poskytuje podporu principu *Single Sign-On*.

Z pohledu administrativy jsou uživatelé a služby v mechanismu Kerberos seskupeny v samostatných administrativních doménách, nazývaných realm. Realm zpravidla kopíruje organizační členění, tj. např. každá fakulta univerzity může spravovat vlastní realm obsahující své uživatele a služby, který je výhradně pod její administrativou. Více takových administrativních jednotek může spolupracovat pomocí tzv. cross-realmových klíčů, které si vymění administrátoři jednotlivých realmů a uloží je do svých databází. Uživatelé pak mohou snadno a transparentně používat služby, které jsou spravovány jiným realmem.

Samotný protokol Kerberos je standardizován standardem IETF [2], existuje několik nezávislých implementací tohoto standardu, v oblasti open-source jsou dostupná řešení MIT a Heimdal. Kerberos je také použit jako základní bezpečnostní mechanismus v systémech MS Windows 2000/XP. Kerberovské nástroje jsou také běžně dostupné ve všech současných linuxových distribucích.

META Centrum aktivně přispívá k vývoji implementace Heimdal a také se se podílelo na návrhu a implementaci řady řešení používajících Kerberos. Významným příspěvkem byla např. open-source implementace podpory protokolu Kerberos v prostředí WWW, které je založeno na modulu `mod_auth_kerb`[1] a komponentě `negotiateauth`[2], která se stala standardní součástí prohlížečů založených na kódu Mozilla. Pracovníci *META Centra* také přispěli k zavedení podpory pro Kerberos i v dalších aplikacích jako je např. OpenSSH.

Kerberos je primárně určen pro prostředí s velmi statickou strukturou spolupracujících realmů. Při použití v geograficky a organizačně distribuovaném prostředí, kde je potřeba dynamicky přidávat a ubírat služby a uživatele, se objevují některé slabiny protokolu, zejména nízká škálovatelnost, které plynou z použití klasické kryptografie. S rostoucím zapojením *META Centra* a jeho uživatelů do nadnárodních projektů je však zapotřebí řešit i otázku snadného přístupu ke

[1] http://modauthkerb.sourceforge.net/
[2] http://www.mozilla.org/projects/netlib/integrated-auth.html

zdrojům mimo *META Centrum* i snadného zpřístupnění zdrojů *META Centrum* uživatelům, kteří stojí mimo ČR. Velkou část těchto problémů lze vyřešit nasazením PKI, která však zároveň generuje jiné problémy související s jejím praktickým nasazením. V posledních letech proto *META Centrum* věnovalo úsilí na zavedení PKI a rozšíření současné kerberovské infrastruktury o výhody PKI.

2.2 Rozvoj infrastruktury veřejných klíčů

V PKI má každý uživatel dvojici soukromého a veřejného klíče. Tyto klíče jsou navzájem svázány, soukromý klíč slouží pro generování elektronických podpisů, které lze ověřit pomocí odpovídajícího veřejného klíče. Veřejné klíče jsou volně distribuovány po uživatelské komunitě a umožňují tak autentizaci za použití elektronických podpisů, kdy libovolný držitel veřejného klíče umí ověřit, že příslušný elektronický podpis byl generován odpovídajícím soukromým klíčem. Pro identifikaci majitele veřejného klíče se používají standardizované certifikáty veřejných klíčů. Certifikát je podepsán certifikační autoritou (CA), která potvrzuje spojení vlastníka certifikátu a příslušného veřejného klíče. Certifikát bývá časově omezen, zpravidla na jeden rok a může obsahovat další informace, které slouží k procesu identifikace, případně pomáhají k navázání komunikace s majitelem certifikátu (příkladem může být emailová adresa, URL seznamu revokovaných certifikátů, apod.). Pro ověření samotného certifikátu je nutné znát veřejný klíč CA, který zpravidla také bývá distribuován ve formě certifikátu.

Praktické nasazení PKI ukazuje problémy, které nejsou zřetelné v menším měřítku, ale které významně ovlivňují bezpečnost celého systému. Jedním ze základních problémů je bezpečná správa soukromých klíčů. Klíče používané v PKI jsou velmi dlouhé řetězce znaků a na rozdíl od hesel si je člověk nemůže zapamatovat. Musí být proto uloženy v nějaké elektronické podobě tak, aby jej mohla přečíst aplikace, kterou uživatel používá. Nejčastěji se dnes klíče ukládají na lokální disk počítače, buď ve formě samostatného souboru nebo ve specializovaném úložišti, které poskytuje aplikace, příp. operační systém. Vždy se ale jedná o data která jsou uložena na disku počítače a tudíž čitelná kýmkoliv, kdo má oprávnění číst příslušnou část disku. Pro ochranu před neoprávněným přístupem k těmto datům se používá šifrování souborů heslem, které musí uživatel zadat při přístupu k soukromému klíči. Navíc, pokud to použitý souborový systém dovoluje, bývá přístup k datům na disku chráněn proti přístupu jiných uživatelů na úrovni operačního systému. Úroveň takové ochrany soukromého klíče je ale velmi křehká, zejména pokud se případnému útočníkovi podaří získat práva majitele soukromého klíče nebo administrátora příslušného systému. Technik jak získat příslušná data z lokálního disku je celá řada, od použití počítačových virů, přes zneužití různých chyb v aplikacích běžících na daném počítači, až k technikám sociálního inženýrství. Pokud se útočníkovi povede získat soubor se soukromým klíčem, může se pokusit najít správné heslo k rozšifrování tohoto souboru. Jeli-

kož má data plně pod kontrolou může např. nasadit klasické techniky pro lámání hesel, které známe z jiných oblastí, jako je hádání hesel hrubou silou nebo slovníkový útok.

Klíčovým problémem v oblasti správy klíčů je fakt, že zabezpečení souboru s klíčem je z velké části v rukách samotného uživatele, což se ukazuje jako nedostatečné. Navíc v oblasti PKI neexistují mechanismy, které by spolehlivě zajistily, že soubor s klíčem je patřičně ochráněn, tj. že použité heslo je dostatečně silné, aby odolalo běžným útokům, že jsou správně nastavena přístupová práva k souboru s klíčem apod. Uživatelé také mohou (a často tak také činí) libovolně manipulovat se souborem s klíčem, např. je kopírovat na jiné počítače, kde klíč potřebují a při těchto operacích může také dojít k prozrazení obsahu soukromého klíče. Důsledkem zneužití těchto problematických míst pak může být velmi oslabený systém.

Cílem *META Centra* bylo zavést PKI tak, aby byly minimalizovány problémy týkající se správy soukromých klíčů. Rozhodli jsme se proto použít technologii čipových karet, která umožňuje spolehlivé uložení PKI dat a jejich ochranu před zneužitím. Na tuto aktivitu jsme získali projekt Fondu rozvoje sdružení CESNET, který nám umožnil nákup potřebného hardwarového vybavení a také provedení změn v infrastruktuře *META Centra*.

3 Nasazení čipových technologií v *META Centru*

Čipové karty a další produkty založené na této technologii (obecně je nazýváme *tokeny*) jsou fyzicky samostatná zařízení vybavená pamětí, kam uživatelé mohou uložit své autentizační údaje. Pokročilejší typy obsahují procesor, který chrání přístup k uloženým datům a je schopen nad těmito daty vykonávat základní kryptografické operace. Citlivá data (např. soukromý klíč) tak nikdy nemusí opustit token, aplikace jen pošle tokenu data ke zpracování a obdrží zpět výsledek. Z bezpečnostních důvodů tokeny obecně neumožňují exportovat soukromý klíč z chráněného úložiště a citlivé objekty jsou ukládány do vnitřního souborovém systému, který je vybaven kontrolou přístupových práv. Jako prevence proti zneužití tokenu po zcizení či krádeži jsou data na tokenu zpřístupněna až po zadání hesla nebo PINu. Toto řízení přístupu je prováděno přímo vestavěným procesorem a operačním systémem.

Řešení založené na hardwarových tokenech poskytuje tzv. *dvoufaktorovou autentizaci*, kdy uživatel musí prokázat, že něco vlastní (tj. hardwarový token) a zároveň něco zná (tj. PIN zpřístupňující soukromý klič na tokenu).

V současné době je již na trhu mnoho druhů tokenů a lze je rozdělit do dvou základních kategorií: na klasické čipové karty a USB tokeny. Zatímco čipové karty potřebují pro připojení k počítači speciální zařízení, tzv. čtečku, USB tokeny kombinují funkcionalitu karty i čtečky v jediném kusu hardware.

3.1 Výběr tokenu

V první fázi projektu jsme se zaměřili na výběr vhodného tokenu, který bude vyhovovat požadavkům *META Centra*. Testovali jsme několik typů klasických čipových karet i USB tokenů s cílem ověřit, že tato řešení se opravdu liší pouze konstrukcí a že funkcionalita poskytovaná aplikacím nezávisí na typu provedení. Nicméně od počátku jsme preferovali řešení založené na USB tokenu, který umožňuje snadnější manipulaci, protože uživateli stačí pouze jedna komponenta místo dvou (tj. jediný USB token místo čtečky a karty). Čipové karty jsou vhodné tam, kde čtečky jsou přímo součástí vybavení organizace a uživatel nosí pouze kartu. V případě uživatelů *META Centra*, kteří jsou distribuováni po celé ČR, pocházejí z různých domovských institucí a také často cestují by toto řešení pouze zvyšovalo nebezpečí, že uživatel jednu nebo druhou komponentu zapomene a nebude se moci autentizovat. Naopak USB token jako jeden kus, který integruje čtečku i kartu, je pro tento typ práce vhodnější.

Pro výběr tokenů jsme stanovili následující kriteria:

- Možnost generování dvojice soukromého a veřejného klíče přímo na tokenu a nemožnost exportu soukromého klíče z tokenu. Tyto dva předpoklady zaručují vyšší bezpečnost citlivých dat a zároveň vynucené použití dvoufaktorové autentizace.
- Token musí podporovat standardy PKCS #11 [3], který definuje komunikaci mezi aplikací a tokenem a PKCS #15 [4], který definuje organizaci souborů na tokenu. Podpora těchto standardů umožňuje aplikacím komunikovat s tokeny bez závislosti na konkrétním typu hardware. Dodatečným kritériem byla podpora Microsoft CAPI[3], což je rozhraní pomocí kterého mohou aplikace pro MS Windows jednotně komunikovat s tokeny.
- Pro vývoj aplikací, které budou komunikovat s tokenem je nutné, aby dodavatel dodával vývojářský kit SDK a aby token byl také podporován v open-source řešeních, která umožňují tvorbu aplikací nad tokeny.

Pro vyhodnocení stanovených kritérií jsme tokeny podrobili sadě následujících testů, které se zaměřily na spolupráci tokenů a konkrétních aplikací:

- Token musí mít dobrou podporu v operačních systémech MS Windows 2000/XP a rovněž v hlavních distribucích Linuxu (Debian, SuSE, Fedora, Ubuntu). Tyto systémy používá většina uživatelů *META Centra*.

  V případě Linuxu jsme zjišťovali, zda je token podporován přímo distribučními nástroji, zda je správně detekován a zda funguje automatická detekce. Ve Windows byly podmínky jednodušší, protože dodavatelé se převážně orientují na tento operační systém, v tomto případě jsme tedy testovali, že ovladače jsou korektně nainstalovány a že systémové nástroje Windows

[3] http://www.microsoft.com/mind/0697/crypto.asp

korektně zavedou certifikáty do úložiště certifikátů po připojení tokenu a zase je odstraní po jeho vyjmutí. Přes systémové úložiště certifikátu pracuje většina Windowsových aplikací a jeho korektní spolupráce s tokenem je tedy nezbytná.

- Token musí být podporován open-source projektem OpenSC[4], který implementuje PKCS #11 a PKCS #15 rozhraní a umožňuje tak vyvíjet aplikace, které přistupují k PKI datům na tokenu. Pomocí implementace PKCS #11 z OpenSC je možné snadno nakonfigurovat přístup k tokenům u aplikací, které implementují svou bezpečnostní vrstvu pomocí tohoto rozhraní (příkladem může být Mozilla Firefox). Součástí balíku OpenSC je i implementace PKCS #11 modulu do aplikačního rozhraní OpenSSL. Díky tomuto modulu je možné snadněji upravit zdrojový kód aplikací, které jsou napsány pomocí OpenSSL API tak, aby využívaly data z tokenů.

  Prováděné testy ověřovaly funkčnost algoritmů implementovaných na tokenu, zejména hashovacích (MD5,SHA1, RIPEMD160) a šifrovacích (RSA, DES). Dále jsme testovali možnosti přístupu k datům na tokenu pomocí OpenSSL API i pomocí skriptů, které využívají řádkové příkazy z balíku OpenSC.

- Token musí být funkční s běžnými webovým prohlížeči jako je Internet Explorer a prohlížeče založené na Mozille. Testy jsme prováděli s IE 6.0, Mozilla 1.6/1.7 Firefox 1.0.x/1.5.x. V případě IE jsme testovali použití s dodávanými ovladači, které umožnily práci s tokenem pomocí MS CAPI. V případě Firefox/Mozilla jsme použili OpenSC implementaci PKCS #11 modulu jak pod operačním systémem Windows, tak Linux. Testem byl přístup na zabezpečené stránky, které vyžadovaly klientskou autentizaci certifikátem, který byl uložen na tokenu.

- Obdobně jako s prohlížeči musí být token schopen pracovat i s běžnými mailovými klienty (MS Outlook, Mozilla Thunderbird). Práce s tokeny je v Thunderbirdu stejná jako v prohlížečích založených na Mozille, tzn. je realizována přes PKCS #11 modul. Outlook pracuje s tokeny stejně jako IE pomocí MS CAPI. Tento test se sestával z podepisování a šifrování e-mailů pomocí klíče a certifikátu uloženého na tokenu.

Testům jsme podrobili tři modely tokenů:

- USB token EUTRON CryptoIdentity 5
- Aladdin eToken PRO PRO32
- USB token Rainbow iKey 3000

[4] http://http://www.opensc-project.org/

Všechny požadavky splnil nejlépe token iKey 3000 od firmy SafeSign Inc.[5], který má dobrou podporu v open-source komunitě a patří k nejprodávanějším zařízením své třídy. Během testování iKey 3000 také ukázal, že je odolný vůči fyzickému poškození.

3.2 Zapojení tokenů do infrastruktury *META Centra*

Po výběru konkrétního typu tokenu bylo zapotřebí upravit prostředí *META Centra* tak, aby umožnilo jeho hladké zapojení. Klíčovým rozhodnutím bylo stanovení způsobu spolupráce PKI a systému Kerberos, který je založen výhradně na symetrické kryptografii. Uvažovali jsme několik přístupů od kompletního nahrazení kerberovské infrastruktury řešením založeným výhradně na PKI až po možnost tunelování protokolu Kerberos v TLS/SSL. Po vyhodnocení možných řešení jsme se rozhodli ponechat beze změn základní infrastrukturu postavenou na systému Kerberos a rozšířit způsob získávání kerberovských lístků tak, aby vedle standardního použití hesla podporoval i autentizaci pomocí PKI. Z uživatelského pohledu se jedná o změnu způsobu přihlašování do *META Centra*, tj. získávání TGT lístku, kdy je místo standardního hesla použit token s certifikátem. Veškerá následná komunikace a principy autentizace však zůstávají beze změny.

Tento přístup nám umožnil využít kladných vlastností z obou systémů a je přitom je minimálně ovlivněn jejich negativy (např. složitou propagací revokačních údajů u PKI). Tento způsob také usnadnil nasazení v produkčním provozu, protože vyžaduje změnu pouze jedné části protokolu, zatímco zbytek infrastruktury zůstává beze změn. Používané služby v *META Centru* tak nebylo potřeba měnit a ve skutečnosti je pro ně změna transparentní, protože nepoznají, zda byl příslušný služební lístek, kterým se klient prokazuje vytvořen pomocí PKI certifikátu nebo klasickým heslem.

Existuje několik možností, jak zapojit PKI do procesu získávání iniciálního TGT lístku v systému Kerberos. Zřejmě nejdůkladněji se této problematice věnuje pracovní skupina IETF, která publikovala návrh standardu PK-INIT [5], který řeší právě tuto problematiku. Specifikace PK-INIT umožňuje uživatelům použít certifikát pro získání iniciálního lístku místo hesla a standardizuje protokol pro tento typ autentizace. Dokument PK-INIT existuje již dlouho, ale stále jen ve formátu návrhu, nicméně řada systémů jej již implementovala (mj. je součástí systému MS Windows 2000/XP). Bohužel v době zahájení našeho projektu neexistovala žádná implementace dostupná pro open-source distribuce systému Kerberos. Vzhledem k tomu, že PK-INIT přesně odpovídal našim požadavkům, rozhodli jsme se jej implementovat pro distribuci Heimdal, která je v *META Centru* nasazena.

[5] http://www.safenet-inc.com/

Výsledkem je poměrně rozsáhlá změna základního protokolu, která zavádí nezbytné závislosti na knihovnách třetích stran (zejména OpenSSL použité pro ověření X.509 certifikátů a RSA klíčů a jejich zpracování). Součástí první implementace byly také změny v ASN.1 parseru umožňující kódování a dekódování zpráv nesoucí PKI informace. Výslednou implementaci jsme úspěšně testovali proti komerčním implementacím z MS Windows a Packet Cable.

Naše implementace byla akceptována vývojáři systému Heimdal a je standardní součástí současných distribucí a na jejím dalším vývoji se podílí řada vývojářů. Podle nedávných testů je tato implementace interoperabilní s dalšími implementacemi PK-INIT, včetně současné verze z MS Windows Vista. V současné době je tato implementace založená na našem řešení jedinou open-source implementací této specifikace.

První verze implementace pracovala pouze s PKI daty uloženými v souborech. Pro naše účely bylo tedy nutné ji dále rozšířit tak, aby bylo možné použít i tokeny. K realizaci tohoto řešení jsme přistoupili až po akceptování kódu vývojáři a využili jsme tedy spolupráce s open-source komunitou, se kterou jsme na tomto řešení spolupracovali. Výsledkem je rozšíření PK-INIT, které podporuje tokeny přístupné přes PKCS #11 rozhraní. Podpora je implementována pomocí rozhraní OpenSSL Engine a umožňuje snadno zapojit většinu používaných tokenů, protože podpora PKCS #11 je dnes už častá.

Součástí této etapy bylo také vytvoření instalačních balíčků pro uživatele. Pro Linuxové systémy jsme připravili binární balíčky pro hlavní distribuce SuSE, Debian, Fedora. Obtížnější byla situace na platformě MS Windows, kam bylo nejprve nutné přenést použité kerberovské knihovny. Po úspěšném dokončení tohoto kroku jsme vytvořili instalátor i pro MS Windows. Součástí balíčků pro všechny platformy jsou základní nástroje pro přístup k *META Centru*, zejména příkaz pro získání lístků (tj. přihlášení do systému). Distribuujeme také ssh klienty, kteří umožňují autentizaci pomocí kerberovských lístků a jejich delegování. V případě Linuxu používáme standardní klienty produkované projektem OpenSSH, v případě MS Windows používáme úpravy pro klienty putty a WinSCP, které zavádějí kerberovskou autentizaci. K dispozici jsou samozřejmě i zdrojové texty pro veškerý software, který jsme v rámci projektu vyvinuli.

Nezbytné změny jsme museli aplikovat také na systém správy uživatelů Perun [6], kam bylo zejména nutné zavést nový typ uživatelské identity založený na certifikátech, který je odlišný od kerberovského. Výsledkem je systém, který umožňuje přiřadit více typů identifikací jednomu uživateli. Součástí těchto úprav byla také integrace mapování identit se serverem Kerberos, včetně automatizovaného propagování změn.

META Centrum nabízí svým uživatelům také portál s různými informacemi o poskytovaném prostředí. Část těchto informací (jako např. osobní údaje uživatele) jsou přístupné pouze oprávněným uživatelům. Pro jejich autentizaci je použit modul `mod_auth_kerb`, který zajišťuje ověření hesla. Při použití tokenu se

však uživatel autentizuje již svým certifikátem a není tedy nutné po něm heslo vyžadovat, naopak uživatelé, kteří pro autentizaci certifikát nepoužijí musí mít možnost, jak své heslo zadat. Vzhledem k současné architektuře serveru Apache není možné nakonfigurovat takový podmíněný mechanismus a bylo nutné pozměnit modul provádějící kerberovskou autentizaci tak, aby tento model podporoval.

3.3 Podpora tokenů v aplikacích

Vedle přístupu k prostředkům *META Centra* jsme použili tokeny i pro autentizaci vůči dalším službám. Soustředili jsme se na podporu VPN řešení pro mobilní uživatele a na autentizaci ve světě WiFi sítí.

Vzhledem k restriktivním firewallům a omezením v cizích sítích není možné z některých sítí navázat spojení s VPN serverem *META Centra* ani pomocí protokolu PPTP (Point-to-Point Tunneling Protocol), který je dnes nejpoužívanějším nástrojem pro VPN spojení. Proto vznikl požadavek na VPN, která bude pracovat na aplikační vrstvě a bude umožňovat tunelování přes standardní port HTTP nebo protokol UDP a jejíž parametry bude možné snadno měnit (např. číslo portu, na kterém VPN server přijímá spojení). Jelikož je HTTP provoz na drtivé většině firewallů povolen, umožňuje toto spojení obejít omezení dané lokálními firewally a vytvořit VPN spojení.

Pro realizaci VPN jsme zvolili řešení založené na koncentrátoru z projektu OpenVPN[6], který umožňuje vytvořit VPN připojení provozované na aplikační vrstvě v ISO/OSI modelu a provozujeme ji ve dvou režimech, kdy je VPN provoz buď tunelován přes HTTP nebo UDP.

Nezbytnou součástí řešení bylo zajistit spolehlivou autentizaci tak, aby bylo možné realizovat řízení přístupu k VPN serveru. Software OpenVPN obsahuje podporu pro použití tokenů k autentizaci klienta proti serveru a k sestavení chráněného kanálu. Pro účely řízení přístupu jsme využili možnosti OpenVPN serveru spouštět lokální skripty při každé žádosti o autentizaci klienta. Pro tento účel jsme vyvinuli skript, který kontroluje subjekt certifikátu uživatele proti statickému seznamu povolených certifikátů. Tento seznam je spravován systémem Perun.

META Centrum se připojilo do projektu Eduroam[7], který umožňuje transparentní používání propojených sítí, zejména se jedná o WiFi roaming. Nutnou podmínkou členství v Eduroamu je vlastní Radius server, který provádí autentizaci uživatelů připojené organizace. Na poli síťové autentizace je používán standard 802.1x, který zajišťuje zapouzdření zpráv protokolu EAP [7] (Extensible Authentication Protocol) přes LAN a Wireless LAN sítě. Autentizační protokol EAP tvoří obálku konkrétním autentizačním protokolům jako je TLS [8] (Trans-

[6] http://www.openvpn.net/
[7] http://www.eduroam.org/

port Layer Security), TTLS [9] (Tunneled Transport Layer Security) a PEAP [10] (Protected Extensible Authentication Protocol).

Protokol TLS je rozšířený protokol pro autentizaci serveru i klienta pomocí certifikátů. Používá se často např. pro zabezpečení webových serverů. U protokolu TTLS se ověřuje pouze certifikát serveru, následně se vytvoří TLS tunel pro další autentizační protokol, jako je například PAP [11] (Password Authentication Protocol) používající jméno a heslo v čisté formě. Protokol PEAP je podobný protokolu TTLS, vytváří bezpečný TLS tunel při ověření certifikátu serveru, kterým lze tunelovat autentizační protokoly jako je CHAP [12] (Challenge Handshake Authentication Protocol), MSCHAP [13] (MicroSoft Challenge Handshake Authentication Protocol), MSCHAPv2 [14] (MicroSoft Challenge Handshake Authentication Protocol Version 2). Nejznámější je verze PEAP-MSCHAPv2, kterou nativně používají operační systémy Windows od verze 2000 SP4 a Mac OS 10.3 a výše.

Jelikož nás primárně zajímá klientská autentizace pomocí certifikátů a tokenů, zeměřili jsme se výhradně na použití protokolu TLS, který jediný podporuje tento model. V současné době provozujeme prototypové řešení a máme nakonfigurováno několik WiFi přístupových bodů, které mohou použít uživatelé s tokeny pro přístup k bezdrátové síti. Klienti MS Windows jsou připraveni na autentizaci pomocí tokenů, stačí jen zatrhnout příslušné položky v konfiguraci. V případě Linuxu je potřeba použít balík `wpa_supplicant`, který již také podporuje tokeny přes PKCS #11 rozhraní.

Na straně Radius serveru používáme open-source implementaci Freeradius[8], který ale neposkytuje rozumné nástroje pro autorizaci. Množinu povolených certifikátů lze specifikovat pouze pomocí jednoho regulárního výrazu, což znemožňuje zadat rozsáhlejší seznam certifikátů. Proto jsme navrhli rozšíření modulu, který zabezpečuje TLS autentizaci tak, aby vracel kompletní informaci o klientském certifikátu. Tato informace bude následně použita během autorizačních kontrol, které jsou realizovány pomocí vyhrazených skriptů. Implementaci tohoto řešení dokončujeme v době psaní tohoto příspěvku. Po jejím dokončení bude možné na Radius serveru odlišit uživatele *METACentra* od ostatních držitelů certifikátu a nasadit podporu tokenů do produkčního prostředí.

3.4 Distribuce tokenů

V průběhu roku 2005 jsme začali distribuovat tokeny mezi koncové uživatele. Úzce spolupracujeme s certifikační autoritou sdružení CESNET, která je uznávaná v celosvětovém akademickém a gridovém prostředí a držitelé certifikátů této CA mají snadnější přístup ke zdrojům mimo ČR. Abychom usnadnili proces vydávání tokenů uživatelům, uspořádali jsme dvě školení v Praze a Brně, na nichž

[8] http://www.freeradius.org/

uživatelé nejen získali USB token, ale především jim byla názorně předvedena práce s ním a mohli rovnou své nové tokeny registrovat u Registrační autority (RA) CESNET CA.

Kromě dedikovaných školení byly využívány i další příležitosti přidělení tokenů uživatelům (např. akce pořádané *META Centrem*), uživatelé mají rovněž možnost získat USB token osobní návštěvou v Brně a v Praze.

Distribuce tokenů a zejména motivace uživatelů k používání tokenů se ukázala jako jeden z významných problémů při zavádění tokenů. Zejména uživatelé, kteří nepotřebují certifikát pro přístup k mezinárodním projektům jsou spokojeni se stávajícím stavem využívající jméno a heslo a nemají chuť měnit své návyky. Vzhledem k tomu, že *META Centrum* se bude stále více zapojovat do nadnárodních aktivit, kde certifikát je jedinou možností autentizace, snažíme se uživatele motivovat k používání tokenů tak, aby byli na tyto možnosti připraveni. Mezi motivační prvky patří např. zavádění nových služeb, které rozšiřují funkcionalitu *META Centra*, které jsou však přístupné jen uživatelům, kteří se prokazují certifikátem apod. Neustále také pracujeme na zlepšování nástrojů a mechanismů, které využívají tokeny a eliminují jejich negativní vlastnosti, zejména nízký výkon procesoru na tokenu.

4 Budoucí práce

Přestože USB tokeny umožňují širokou škálu použití, jsou oblasti kde je nelze použít. Některé počítače nejsou dostatečně důvěryhodné a nebo jednoduše nemají dostupný USB port. Plánujeme doplnit řešení založené na PKI tokenech o autentizaci pomocí jednorázových hesel (*One-Time Passwords–OTP*). Chtěli bychom využít kombinované USB tokeny, které integrují OTP do klasického USB tokenu.

5 Závěr

V příspěvku bylo představeno gridové prostředí *META Centra* a přiblížena jeho bezpečnostní architektura. Byla popsána technologie čipových karet a průběh výběru tokenu pro uživatele *META Centra*. Představili jsme naše výsledky v oblasti integrace mechanismů PKI a Kerberos i technické a administrativní aspekty související s distribucí tokenů mezi uživatele. Demonstrovali jsme, že technologie čipových karet lze úspěšně nasadit pro efektivní zabezpečení soukromých klíčů. Na druhé straně naše zkušenosti ukazují, že zásadním problémem nejsou jen technická úskalí, ale i nutnost změny uživatelských návyků.

Literatura

[1] NEUMAN, B. C., TS'O, T. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, roč. 32, s. 33–38, září 1994.

[2] NEUMAN, C., YU, T., HARTMAN, S., RAEBURN, K. *The Kerberos Network Authentication Service (V5).* IETF RFC 4120. Červenec 2005.

[3] RSA Laboratories. *PKCS 11–Cryptographic Token Interface Standard.* 1994.

[4] RSA Laboratories. *PKCS 15 v1.0: Cryptographic Token Information Format Standard.* 1999.

[5] ZHU, L., TUNG, B. *Public Key Cryptography for Initial Authentication in Kerberos.* Únor 2006. Internet-Draft, work in progress.

[6] KŘENEK, A., SEBEATIÁNOVÁ, Z. *Perun – Resource Access Management System in the $\mathcal{META}$ Centre Project.* Tech. zpráva, CESNET, 2004.

[7] ABOBA, B., BLUMK, L., VOLLBRECHT, J., CARLSON, J., LEVKOWETZ, H. (Ed.) *Extensible Authentication Protocol (EAP).* IETF RFC 3748. 2004.

[8] DIERKS, T., RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.1.* IETF RFC 4346. 2006.

[9] FUNK, P., BLAKE-WILSON, S. *EAP Tunneled TLS Authentication Protocol (EAP–TTLS).* Internet Draft, work in progress.

[10] PALEKAR, A., SIMON, D., ZORN, G., JOSEFSSON, S. *Protected EAP Protocol (PEAP).* 2003. Internet Draft, work in progress.

[11] LLOYD, B., SIMPSON, W. *PPP Authentication Protocols.* IETF RFC 1334. 1992.

[12] SIMPSON, W. *PPP Challenge Handshake Authentication Protocol (CHAP).* IETF RFC 1994. 1996.

[13] ZORN, G., COBB, S. *Microsoft PPP CHAP Extensions.* IETF RFC 2433. 1998.

[14] ZORN, G. *Microsoft PPP CHAP Extensions, Version 2.* IETF RFC 2759. 2000.

LiveCD a jejich použití

Michal Vyskočil

E-mail: xvysko02@stud.fit.vutbr.cz

1 Úvod

LiveCD, neboli obecněji živé systémy jsou speciálně upravené verze běžných operačních systémů, které jsou uzpůsobeny tak, že dokáží nabootovat a běžet z CD/DVD jednotky. Patří sem i distribuce specializované na běh z Flash paměti. Díky tomu, že nepotřebují přistupovat k datům umístěným na pevném disku, tak umožňují snadné spuštění prakticky libovolného systému. Není totiž potřeba je zdlouhavě instalovat a jakkoliv zasahovat do počitače. Navíc v dnešních dobách už bychom těžko hledali takový stroj, který neumí nabootovat z CD-ROM, nebo z flash disku, tak LiveCD vytlačili kdysi velmi populární bootovací diskety, nebo specializované diskteové distribuce Linuxu.

Protože se budu zabývat téměř výhradně distribucemi operačního systému GNU s jádrem Linux, budu pojmy *LiveCD*, *operační systém*, nebo *distribuce* označovat právě GNU/Linux upravený pro běh z CD/DVD, nebo Flash disku.

1.1 K čemu LiveCD slouží

Jak jsem již řekl, největší výhoda LiveCD spočívá právě v tom, že nikterak nezasahují do stávající systému a ani je k ničemu nepotřebují. Z toho také plynou jednotlivé případy použití:

Nový programu Je potřeba ukázat novou a převratnou verzi nějaké stávající aplikace a přesvědčit tak uživatele, aby jí přijal. Příkladem může být amaroK LiveCD.

Nová technologie Vyzkoušet novou aplikaci není tak těžké, ale pokud by se mělo jednat o větší zásah do systému – příkladem je Kororaa, které prezentuje XGL.

Jiný operační systém Ovšem nejtěžší překážka bývá odzkoušení nového, neznámého systému. Pro uživatele Windows jsou určeny klasické Linuxové LiveCD, které jim mají pomoci tento systém snadno poznat. Ovšem existují i LiveCD exotičtějších systémů, než je Linux.

Opravný nástroj Při pádu našeho hlavního systému není nic snažšího, než nabootovat LiveCD a opravit jej z něj. K tomuto účelu sloužily kdysi právě záchranné diskety, ale ty limitovala jejich malá kapacita.

Přenosné pracovní prostředí Něco jako notebook pro chudé, chtělo by se říct. Na druhou stranu nám LiveCD umožňuje pracovat v našem oblíbeném systému (a nastaveném prostředí) bez nutnosti zásahů třeba do firemního notebooku.

Instalace Klasické instalační CD jsou LiveCD, na kterých běží instalátor. Ale to je většinou před uživatelem skryto. Naproti tomu kompilované distribuce jako LFS (*Linux from Scratch*) nebo Gentoo mají instalační LiveCD, které obsahuje nezbytné vybavení pro tzv. *boostrapping*. Přeložení překladače a glibc a základních utilit.

Ostatní Například specializovaná distribuce určená pro routery, nahraje se na Flash kartu a máme Linuxový router. Ale možnosti použití jsou prakticky nekonečné, takže vše, co nepatří do předchozích bodů, tak patří sem.

1.2 Nelinuxové LiveCD

Přestože mezi LiveCD jsou asi nejvíce zastoupeny právě distribuce Linuxu a tento příspěvek se na ně specializuje, byla by škoda neuvést příklady těch ostatních operačních systémů a jejich Live verzí.

OpenSolaris	Belenix
OpenSolaris	SchilliX
GNU/Solaris	Nexenta
FreeBSD	FreeSBIE
NetBSD	NeWBIE
OpenBSD	OliveBSD
OpenBSD	Anonym.OS
ReactOS	ReactOS LiveCD
GNU/Hurd	Hurd LiveCD
BeOS	XBEOX
Plan9	Plan9 live/install CD
Minix	Minix

Pro operační systémy Mac OSX a Windows existují nástroje pro tvorbu LiveCD.

Mac OSX	BootCD
Windows	Bart PE

2 Bootování

Každý operační systém musí projít procesem bootování a LiveCD nejsou výjimkou. První nezbytnou položkou je tedy *zavaděč*, což je program, který zavádí (bootuje) vlastní systém. Zavaděče rozdělujeme na *primární* (first-stage) a *sekundární* (second-stage). Primárním je například BIOS, nebo Open Firmware společnosti Sun. Ty jsou zabudované přímo v hardware. Sekundární jsou ty, které volá hardwarový primární zavaděč a ty, které nahrávají a spouští vlastní operační systém. Mezi známé patří LILO (*Linux Loader*), GRUB (*Grand Unified Bootloader*), loadlin (nahrává Linux z DOSu), nebo třeba NTLDR, zavaděč operačních systémů z rodiny Windows NT.

Pro spouštění z LiveCD se hodí dva z nich. Je to právě univerzální GRUB a naopak specializovaný *syslinux*.

2.1 Bootování z CD obecně

Proces bootování z CD je popsán v El Torito Bootable CD Specification, což je rozšíření specifikace ISO 9660. El Torito přidává podporu pro které umožňuje bootování z CD-ROM. Z historických důvodů existují dva režimy zavádění

Floppy Emulation Mode Zaváděcí sekvence je uložena v obraze (*image file*) diskety. Obraz je nahrán z CD a zavádění probíhá naprosto stejně jako z klasického pevného disku nebo diskety. Protože souborovým systémem obrazu diskety bývá FAT, používaným zavaděčem je syslinux, nebo GRUB. Tento režim vznikl proto, že tehdejší BIOSy nedokázaly načíst bootovací sekvenci z ISO 9660 systémů.

No Emulation Mode V tomto případě je zaváděcí sekvence uložena přímo na disku CD-ROM a bootování probíhá přímo z něj. Tento režim je vyžadován zavaděčem isolinux (pochopitelně GRUB) a vzhledem k tomu, že všechny BIOSy z posledních asi 10 let podporují bootování z ISO 9660, tak není moc důvodů, proč používat emulační mód.

2.2 Syslinux

Ve skutečnosti se jedná o balík programů, protože pod označením syslinux se skrývá celá škála jednoúčelových zavaděčů:

- syslinux, pro bootování ze souborového systému FAT
- isolinux, pro bootování z ISO 9660 filesystému, který je používán na CD
- pxelinux, bootování ze síťového serveru

- extlinux, bootování ze souborového systému ext2/ext3
- memdisk, bootování starších operačních systémů

Pro nás je pochopitelně nejzajímavější právě zavaděč isolinux. Pro vytvoření obrazu CD navíc potřebujeme `mkisofs` ve verzi 1.13, nebo vyšší, který je součástí balíku `cdrtools`. Prvním krokem je do určitého adresáře nakopírovat věci, které mají být na LiveCD. Ten se stane kořenovým adresáře CD. Isolinux standardně hledá konfigurační soubor `isolinux.cfg` v adresářích `/isolinux`, `/boot/isolinux`, nebo v `/`, kde `/` označuje kořenový adresář CD. Toto je ukázka nastavení z distribuce ABC Linux 2005.

```
display boot/splash.cfg
default linux
prompt 1
timeout 0
F1 boot/splash.txt
F2 boot/license.txt

label abc
kernel boot/vmlinuz
append max_loop=255 initrd=boot/initrd.gz init=linuxrc
load_ramdisk=1
prompt_ramdisk=0 ramdisk_size=4444 root=/dev/ram0 rw

label linux
kernel boot/vmlinuz
append max_loop=255 initrd=boot/initrd.gz init=linuxrc
load_ramdisk=1
prompt_ramdisk=0 ramdisk_size=4444 root=/dev/ram0 rw

label memtest
kernel boot/memtest
```

Na příkladu rovněž vidíme, že isolinux nezavádí pouze linuxové jádro, ale kterýkoliv soubor typu ¡code¿x86 boot sector¡/code¿, v našem případě `memtest`. No a pokud máme všechny potřebné soubory na místě, můžeme vytvořit obraz CD.

```
mkisofs -o isolinux.iso \
-b boot/isolinux.bin -c boot/isolinux.boot \
-no-emul-boot -boot-info-table -boot-load-size 4 \
-v -J -R -D -A "LiveCD" -V "LiveCD" \
```

Parametr `-b` zaručí vytvoření bootovacího obrazu dle El Torito, `-no-emul-boot` aktivuje neemulační metodu bootování. Parametry `-J` a `-R` povolí rozšíření Jolitet (Windows), respektive Rock Ridge(unix), které umožňují delší jména souborů, anebo v případě Rock Ridge i unixová práva. Pozor, isolinux *neumí* soubory využívající tyto rozšíření číst!

2.3 Grub

Nebo-li *Grand Unified Bootloader*, je univerzálním zavaděčem. Je součástí projektu GNU a narozdíl od ostatních, zde uvedených, programů, dokáže zavádět libovolný operační systém. Buďto jej podporuje přímo (jako například Hurd, nebo Linux), anebo použije takzvaný *chainloading*, kdy spustí vlastní zavaděč operačního systému (například NTLDR). Podporuje celou řadu souborových systémů, jako jsou

- ext2/ext3
- IBM's JFS
- ISO 9660
- Minix file system
- ReiserFS
- SGI's XFS
- UFS/UFS2
- VFAT, (včetně FAT16 a FAT32)
- VSTa

Jak je vidět, tak GRUB je asi nejuniverzálnějším zavaděčem a díky podpoře ISO 9660 umí zavádět i LiveCD. Ten stejně jako isolinux podporuje neemulační bootovací režim. A pro zavádění z CD-ROM vyžaduje speciální Stage2 soubor nazvaný `stage2_eltorito`. Ten bývá umístěn v adresáři[1] `/usr/lib/grub/i386-pc/`. Dalším a nepovinným souborem je konfigurace – `menu.lst`. Narozdíl od isolinux GRUB hledá své soubory jen v adresáři `/boot/grub`. Následuje ukázka nastavení z distribuce minimax.

```
default 1
timeout 30
#splashimage=(cd)/boot/grub/robin.xpm
```

[1] Jak tvrdí dokumentace, ale konkrétně na mém systému je v adresáři /boot/grub

```
title=Minimax  1024x768
root (cd)
kernel (cd)/boot/vmlinuz root=/dev/ram0 init=/linuxrc \
load_ramdisk=1 promt_ramdisk=0 ramdisk_size=30000 \
vga=791
initrd (cd)/boot/initrd

title=Minimax text mode
root (cd)
kernel (cd)/boot/vmlinuz root=/dev/ram0 init=/linuxrc \
load_ramdisk=1 promt_ramdisk=0 ramdisk_size=30000 \
vga=0
initrd (cd)/boot/initrd

title=Memtest
root (cd)
kernel (cd)/boot/memtest.bin
```

Zde je vidět, že se grub odkazuje na kořenový adresář CD jako na `(cd)`, což je důležité, protože klasické `(hdx,y)` nelze v tomto případě použít, protože každý počítač má umístění disků a mechanik odlišné. Vytvoření isa je prakticky totožné, jako v případě isolinux.

```
mkisofs -o grub.iso \
-b boot/grub/stage2_eltorito -no-emul-boot \
-boot-info-table -boot-load-size 4 \
-v -J -R -D -A "LiveCD" -V "LiveCD" \
```

3 Předstartovní příprava

Klasický postup bootování jádra je takový, že se

1. Zavede do paměti a spustí.
2. Provede se inicializace potřebného hardwaru.
3. Není-li určeno parametrem `init=` jinak, jádro spustí program `/sbin/init`.

U LiveCD to tak jednoduše nejde, protože to musí běžet bez načítání potřebných programů z pevného disku. Stejný problém nastává v situaci, kdy máme distribuční jádro, které obsahuje podporu pro souborové systémy na disku…,

který ovšem nedokáže bez těch modulů přečíst. Tento problém se nazývá *bootstrapping* a bylo by chybou si myslet, že lidé od počítačů na něj narazili jako první. Podobné problémy nastávají všude, kde platí kauzalita, například v kosmologii. Vývojáři jádra ovšem mají situaci oproti filozofům jednodušší. Vymysleli `initrd` (*Initial Ram Disk*), což je speciální soubor, který se zavede do paměti a jádro z něho při bootu načte potřebné soubory. Musí obsahovat patřičnou podporu:

```
Device Drivers > Block devices:
<*> RAM disk support
(4096) Default RAM disk size (kbytes) (NEW)
[*]   Initial RAM disk (initrd) support
```

U klasických distribucí bývá součástí `initrd` ovladač pro souborové systémy a případně také `bootsplash` a moduly pro `framebuffer` pro grafické bootování. Live systémy tu mají uložen minimální systém, který musí připravit všechno potřebné pro spuštění systému a teprve potom spustí samotný `/sbin/init`. Ramdisky mají velmi omezenou kapacitu, takže není vůbec snadné do nich dát byť jen minimální systém, který by obsahoval klasické GNU nástroje, na něž jsme zvyklí.

3.1 Busybox

Řešením je věc, která se nazývá `busybox`, což je *The Swiss Army Knife of Embedded Linux*. Jedná se o kolekci nejpoužívanějších unixových nástrojů, které jsou součástí jednoho spustitelného souboru. Plná instalace zahrnuje více než 190 programů, ale není pochopitelně potřeba překládat úpně všechny nástroje. Lze jej přeložit s knihovnou glibc, ale i s menší uClibc, což je vhodné především pro zabudované systémy.

Volat binární soubory busyboxu je možné dvěma způsoby. Buď explicitně napsat busybox příkaz [parametry], anebo vytvořit soustavu symbolických odkazů a busybox podle prvního argumentu (`argv[0]`) pozná, který program má spustit. Je nutné mít na paměti, že GNU nástroje, které denně používáme, mají oproti těm z busyboxu přeci jen více možností.

3.2 Po spuštění jádra

Konkrétní posloupnost kroků, které musí předstartovací skripty provádět se liší od systému k systému. Obecně se dá říct, že je nutné

1. Vytvořit ramdisk, ve kterém se vytvoří základní systémové adresáře
2. Připojit základní adresáře `/proc` a `/sys`, které jsou důležité pro běh jádra

3. Nahrát základní moduly z ramdisku
4. Připojit různé místa na CD do různých míst v pamět[2]
5. Je možné zkopírovat celý systém do paměti (týká se hlavně menších CD)
6. Některé distribuce vytváří `\etc\fstab`
7. Změna kořenového adresáře někam do paměti a spuštění `/sbin/init`, čímž začne stejný proces bootování jako u klasicky nainstalovaných distribucí.

4 Specializované souborové systémy

LiveCD mají oproti klasickým distribucím jistá specifika. Například daleko menší kapacitu, než pevné disky. Navíc na ně nejde zapisovat. Klasický ISO 9660 i včetně rozšíření nepodporuje příliš dlouhá jména a tak dál. Linuxové jádro už velmi dlouho obsahuje mechanismus zvaný VFS (*Virtual File System*), což je abstraktní rozhraní pro ovladače konkrétních souborových systémů. Díky VFS je snadné dopsat do jádra modul, který umí pracovat s libovolným souborovým systémem. A protože existují stovky různých souborových systémů, Linux dokáže pracovat s prakticky libovolným z nich právě přes volání VFS. Počínaje klasickými diskovými, jako je primitivní FAT nebo transakční a žurnálovací ext3. Dále sem patří síťové souborové systémy, jako například NFS nebo Coda. Existují i databázové, jakým je BFS (BeOS FS). V Unixu a v Linuxu zvlášť je také silná tradice používání pseudo filesystémů, jako procfs, relayfs nebo sysfs. No a také existuje řádka souborových systémů, které jsou velmi specifické.

Důsledkem tohoto přístuu je, že jádro samo o sobě neumí pracovat s žádným souborovým systémem. Jenže při bootu se musí připojit kořenový souborový systém, ale naše jádro nekodáže načíst z disku potřebné moduly. Jak jsem uvedl dříve, `initrd`, tento problém elegantně řeší. A navíc obsahuje i předstartovní skripty, které upraví systém do podoby, která je vhodná pro běh z CD.

Poměrně horkou novinkou je FUSE, která je v hlavním jádře od verze 2.6.14 (v době psaní tohoto přípěvku byla stabilní 2.6.16.1). Tato zkratka znamená *Filesystem in Userspace* a jeho myšlenka pochází už z Plan9. Ve zkratce umožňuje uživatelským programům používat další, speciální souborové systémy. Mezi ně patří třeba GmailFS, který interně používá poštovní schránku na serveru gmail.com, BitTorrent File System, nebo SshFS. Zatím je dostupný pro linuxová jádra 2.4 a 2.6 a jako port pro jádro FreeBSD. FUSE by v budoucnu mohlo nahradit věci, které dnes složitě zajišťuje každá aplikace samostatně (například mc VFS, kio-slaves z KDE, nebo Gnome VFS).

[2] Já vím, takto obecně to zní opravdu strašně

4.1 Komprimované souborové systémy

Plná instalace Slackware zabere 3GB, a to je distribuce obsahující velmi málo software (ve srovnání s distribucemi jako Debian, Gentoo nebo Suse). Je tedy nutné data *on-the-fly*[3] komprimovat a dekoprimovat, tak, aby se na CD vešlo mnohem více dat. Ovšem za cenu větší zátěže procesoru při běžném procházení.

4.1.1 Cramfs

Celé jméno je compressed ROM filesystem. Je součástí jádra od verze 2.3 (respektive 2.4). Jak už napovídá název, jedná se o komprimovaný souborový systém určené jen pro čtení. Některé konvenční distribuce (například Debian) používají initrd právě v tomto formátu. Byl využíván staršími verzemi distribuce Slax (3 a starší) nebo v embedded zařízeních jako handheldy a podobně. Použití

```
mkfs.cramfs source destination
```

Stejně jako mnoho dalších komprimovaných filesystémů používá knihovnu zlib. Bohužel `cramfs` trpí spoustou omezení:

- Maximální velikost souboru je 16 MB.
- Maximální velikost souborového systému je něco málo přes 256 MB.
- Podporuje hardlinky, ale počet odkazů stále zůstává roven 1.
- Není přenositelný na stroje s různým endianness (tedy s různým pořadím vyšších a nižších bytů ve slově).

4.1.2 Cloop

Čili *compressed loopback device* není souborový systém, ale modul, který poskytuje komprimované read-only loopback zařízení. Vytvořil jej Paul Russell (autor `ipchains` a `iptables`) pro distribuci Linuxcare a pro jádra 2.2. Mohou být naformátovány libovolným souborovým systémem (podobně jako tradiční loopback zařízení). Asi nejznámější distribucí, která tyto obrazy používá, je Knoppix (a jeho deriváty). Samotný Klaus Knopper je správcem patchů pro jádra 2.4 a 2.6, protože ve vanilla jádře tento modul není.

```
# vytvoreni obrazu
mkisofs -r -l datadir | \
create compressed fs - 65536 > isoimg.z
```

[3] Čech by řekl za běhu, nebo transparentně, ale anglický termín zní učeněji. Není-liž pravda?

```
# pripojeni (linuxrc distribuce Knoppix)
insmod cloop.o file=/cdrom/KNOPPIX/KNOPPIX
mount -r /dev/cloop /mnt/knoppix
```

Data jsou komprimována po jednotlivých blocích (opět pomocí zlib), typická velikost na LiveCD se pohybuje kolem 256 kB. I cloop trpí některými problémy:

1. `create_compressed_fs`, což je utilita, která vytváří obraz, nepodporuje proudové zpracování. Celý komprimovaný obraz se musí vejít do operační paměti.

2. ovladač `cloop` je velmi pomalý při častých čtení, například v systémech s malou operační pamětí (neustálé načítání a rozbalování bloků) nebo při nahrávání programu s velkým množstvím sdílených knihoven.

4.1.3 zisofs

Je klasický iso 9660 s podporou transparentní komprese a dekomprese (jak jinak než pomocí knihovny zlib). V jádře se objevil ve verzi 2.4.14. O vytváření obrazů se stará balík nástrojů `zisofs-tools`, které spravuje H. Peter Anvin, správce zavaděčů syslinux a isolinux, s nimiž jsme se už setkali. Tento způsob komprese používala distribuce Slax ve verzi 4.x.

Použití programu `mkzftree(1)`, kde `source` je vstupní adresářová struktura a `destination` je výstupní komprimovaná struktura. Výsledek může být zpracován klasickými nástroji, jako `mkisofs`, nebo `growisofs`.

```
mkzftree source destination
```

- `-z` Stupeň komprese (1–9, výchozí 9)
- `-u` Rozbalit, užitečné v systémech, které nativně nepodporují zisofs
- `-x` Nepřistupovat na jiné oddíly, pouze vytvořit přípojné body
- `-X` Nepřistupovat na jiné oddíly a přípojné body nevytvářet.

4.1.4 e2compr

Toto je pouze patch ovladače souborového systému ext2, který přidává podporu transparentní komprese na úrovni souborů. Do konfiguračního souboru přidává volbu `EXT2_COMPR_FL`. Pro jeho použití není potřeba daný oddíl znovu přeformátovat, protože dokáže zcela transparentně používat jak komprimované, tak nekomprimované soubory. Ovšem jeho vlastnosti nejsou ty nejlepší pro LiveCD a nenašel jsem žádné, které by používalo tento souborový systém.

4.1.5 SquashFS

Phillip Lougher uvolnil v roce 2002 první verzi tohoto komprimovaného filesystému, který odstraňuje nevýhody `cramfs`. První verze byla pro jádro 2.4.19; bohužel až do teď nebyl tento souborový systém do vanilla jádra začleněn. Ale Greg KH už před časem zařadil patch do genpatches, což je patchset pro gentoo-sources (výchozí jádro distribuce Gentoo).

V čem se tedy liší od `cramfs`? Preferuje velikost bloku 32 kB (od verze 2 64 kB), která má mnohem lepší kompresní poměr než 4 kB. Jádro nativně podporuje 4 kB bloky a tak tento souborový systém dává zbytek dat explicitně do cache. Komprimuje i informace z adresářů a inodů, které, opět pro větší kompresi, shlukuje dohromady. Podporuje 32bitové uid/guid (cramfs má uid 16 b a guid 8 b), velikost souboru je až 2^32 B (cramfs 2^24). A v neposlední řadě podporuje časové značky (timestamps).

Pro práci s tímto souborovým systémem je nutné mít program `mksquashfs` z balíku `squashfs-tools`. Ten má poněkud nezvyklou syntaxi:

```
mksquashfs source1 source2 ...  dest [options] \
[-e list of exclude dirs/files]
```

Kde `sourceX` označují zdrojové adresáře, `dest` je cílový soubor, `options` jsou volby a na konci je, uvozen parametrem `-e`, seznam souborů adresářů, jež se mají vynechat.

`-all-root`, nebo `-root-owned`	vlastík souborů v novém filesystému bude
`-be`, nebo `-le`	nastav velký (b), nebo malý (l) endián
`-ef <exclude_file>`	seznam vynechaných souborů
`-noI`	nekomprimuj tabulku inodů
`-noD`	datové bloky
`-noF`	fragmentované bloky

4.2 Podpora pro zápis

Kvůli technickým omezením není možné na CD[4] zapisovat, ale existují způsoby, jak systém přinutit, aby se tvářil, že to možné je. Princip je prostý; adresář, do něhož chceme zapisovat, je přípojným bodem pro dva jiné, které nazvěme `ro` a `rw`. Zatímco `ro` je, jak název napovídá, jen pro čtení, a tudíž se vyskytuje na disku CD-ROM, adresář `rw` je pouze v operační paměti a obsahuje změny oproti adresáři `ro`. Potom je nutný speciální souborový systém, který dokáže oba adresáře spojit do jednoho.

[4] A pochopitelně i jiná read-only média, jako třeba ROM paměti.

4.2.1 ovlfs

Jedním z prvních byl takzvaný překryvný (overlay) souborový systém `ovlfs`. Vzhledem k tomu, že je dostupný pouze ve formě patchů pro jádra 2.4, uvádím ho zde především z historických důvodů. Ostatně i nápis na stránce projektu „Last Updated: June 10, 2003“ hovoří za vše. Byli sice snahy portovat tento systém také na jádra řady 2.6, ale k ničemu takovému nakonec nedošlo.

Tento systém dokáže do jednoho přípojného místa připojit dva různé adresáře (souborové systémy). Primární, který je na LiveCD read-only, a sekundární, který bývá v operační paměti. Všechny změny jsou zapisovány na sekundární. Modul `squashfs` si udržuje v paměti jádra vlastní tabulku inodů, odkazů na konkrétní soubory a adresáře. Při požadavku na nový/změněný soubor vrátí odkaz na sekundární adresář. Tento systém se sice vyznačoval některými problémy (zamykání souborů a rychlost), ale vaz mu srazila spíše skutečnost, že nebyl portován na jádra 2.6.

4.2.2 COW

COW není jen samice tura domácího, ale i název technologie *copy on write*. Starší implementací bylo COW-Links, které není příliš dobré a také už nežije. Poslední patch je pro jádro 2.6.11. Z něj vychází projekt `cowloop`: *copy-on-write loop driver*. Uvádím je spíše pro zajímavost a rozšíření obzorů, protože ani jeden z nich není bez hacků vhodný pro tvorbu LiveCD. Více vysvětluje FAQ projektu cowloop.

4.2.3 mini_fo

Projekt `mini_fo` je virtuální souborový systém, který dokáže simulovat zápis na systémech jen pro čtení. Jeho princip je naprosto stejný jako u ostatních systémů. Přesměruje operace zápisu do odkládacího adresáře (*storage directory*). Při čtení automaticky spojí novější data z odkládacího prostoru s těmi, co jsou na médiu jen pro čtení. Je primárně vyvíjen pro embedded systémy, takže má malou velikost (50 kB) a spotřebu paměti. Nenašel jsem žádné LiveCD, které by používalo tento projekt, pouze stesky na značnou chybovost.

4.2.4 UnionFS

Unionfs znamená *stackable unification file system* (což by se dalo přeložit jako stohovatelný/vrstvitelný unifikující souborový systém). Mimo jeho funkčnosti je na něm velmi zajímavé to, že je součástí projektu FiST (*Stackable File System Language and Templates*). Ten používá vysokoúrovňový jazyk specializovaný na tyto souborové systémy. Překladačem `fistgen` je potom proveden překlad na

spustitelný modul jádra. Podporovány jsou jádra systémů Linux, FreeBSD a Solaris.

UnionFS vytváří jeden logický souborový systém, který je tvořen spojením obsahu mnoha různých adresářů (větví), přičemž jejich fyzické umístění je oddělené. Je možné například spojit oddělené programové adresáře do jednoho a vytvořit systém podobný GoboLinuxu, ale bez starostí se symbolickými odkazy. Pro LiveCD je velmi podstatné, že dovoluje kombinovat větve jen pro čtení s těmi určenými pro zápis. Díky funkci copy-on-write je možné vytvořit iluzi zapisovatelného souborového systému s tím, že změny se transparentně přenáší do adresáře určeného pro zápis. První LiveCD, které tento systém používalo na jádře 2.6, bylo DeadCD českého autora iSteve.

Protože je UnionFS asi nejpoužívanějším souborovým systémem na LiveCD, hodí se sem napsat krátký návod na instalaci. Nejprve si zkontrolujte, zda nemáte balíček s unionfs přímo ve své distribuci. Pokud ne, stáhněte si jej ze stránek projektu. Díky svému původu se modul přeloží a nainstaluje sám, takže není nutné překládat jádro.

```
$ wget ftp://ftp.fsl.cs.sunysb.edu/pub/unionfs/unionfs-1.0.14.tar.gz
$ tar -xzf unionfs-1.0.14.tar.gz
```

Doporučuji si přečíst dokumentaci (soubor `README`) a instalační instrukce (soubor `INSTALL`). UnionFS má ve výchozím nastavení zapnutý ladící mód, což velmi nepříjemně zvětší velikost modulu (83 kB versus přibližně 5 MB). Pochopitelně musíte mít hlavičkové soubory jádra
v `/lib/modules/‘uname -r‘/build/include`. Když chcete instalovat modul pro jiné, než aktuálně běžící jádro, vytvořte soubor `fistdev.mk` a napište do něj `TOPINC=-I/cesta/k/jádru/linux-2.6.xx/include`. Dále je požadováno mít hlavičkové soubory `e2fsprogs` a program `ctags`.

```
$ # vypnuti debug flagu
$ cat fistdev.mk
EXTRACFLAGS=-DUNIONFS_NDEBUG
UNIONFS_DEBUG_CFLAG=
$ make
# make install
# depmod -a
```

Tím máme v systému jaderný modul `unionfs.ko` (pro jádra 2.4 `unionfs.o`), a program `unionctl` pro přidávání a odebírání větví a manuálové stránky.

K připojení se používá klasický příkaz `mount`, ale je nutné pomocí volby `-o` specifikovat další parametry.

```
# modprobe unionfs
```

```
# mount -w -t unionfs -o dirs=/tmp/foo/=ro:/tmp/bar/=ro
unionfs /tmp/union/
```

Spojí adresáře `/tmp/foo` a `/tmp/bar` do jednoho adresáře `/tmp/union`. V případě konfliktů má první adresář největší prioritu. Připojené souborové systémy zjistíme příkazem

```
# mount | grep unionfs
unionfs on /tmp/union type unionfs
(rw,dirs=/tmp/foo/=ro:/tmp/bar/=ro)
```

Nebo lépe, příkazem `unionctl`, kterým můžete dynamicky přidávat, nebo odebírat nové přípojné body.

```
# unionctl /tmp/union --list
        /tmp/foo (r-)
        /tmp/bar (r-)
# unionctl /tmp/union --add --after /tmp/bar --mode ro /tmp/ham
# unionctl /tmp/union --list
        /tmp/foo (r-)
        /tmp/bar (r-)
        /tmp/ham (r-)
# unionctl /tmp/union --remove /tmp/bar
# unionctl /tmp/union --list
        /tmp/foo (r-)
        /tmp/ham (r-)
```

Pokud změníte obsah větví, můžete obnovit obsah přípojného bodu příkazem

```
# uniondbg -g /tmp/union/
New generation 3
```

Spojování adresářů jen pro čtení nám v LiveCD nepřinese žádný užitek. Důvodem, proč unionfs používáme, je podpora „zápisu“ na read-only médium. Pojďme tedy něco zapsat na disk CD-ROM.

```
# mount /mnt/ro/
# ls -l /mnt/ro/
total 76
...
-rw-r--r--  1 root root  2546 Sep 27 21:28 requirements.txt
drwxr-xr-x  2 root root  2048 Sep 27 21:27 rootcopy
# ls /mnt/ro/new-file.txt
ls: /mnt/ro/new-file.txt: No such file or directory
# touch /mnt/ro/new-file.txt
touch: cannot touch ‘/mnt/ro/new-file.txt’: Read-only file system
```

V adresáři `/mnt/ro` (read-only) máme připojené CD s distribucí ABC Linux 2005. V adresáři nemáme žádný soubor `new-file.txt` a nejde ani vytvořit.

```
# mount -t unionfs -o dirs=/mnt/ro/=ro unionfs /mnt/cdrom/
# diff -r /mnt/ro/ /mnt/cdrom/
```

Tímto příkazem jsme vytvořili přípojný bod `/mnt/cdrom/`, který je zcela totožný s adresářem `/mnt/ro/`. Ale ani do jednoho z nich nemůžeme zapisovat. Je potřeba ještě určit adresář, kam se budou ukládat změny:

```
# unionctl /mnt/cdrom --add --before /mnt/ro/ --mode rw /mnt/rw/
# unionctl --list /mnt/cdrom/
        /mnt/rw (rw)
        /mnt/ro (r-)
```

A voila, adresář `/mnt/cdrom/` s právem zápisu je k dispozici! Samozřejmě s tím, že se veškeré změny ukládají do adresáře `/mnt/rw`.

```
# echo "burned on CD-ROM " > /mnt/cdrom/new-file.txt
# cat /mnt/cdrom/new-file.txt
burned on CD-ROM
# cat /mnt/rw/new-file.txt
burned on CD-ROM
# cat /mnt/ro/new-file.txt
cat: /mnt/ro/new-file.txt: No such file or directory
# cmp /mnt/cdrom/new-file.txt /mnt/rw/new-file.txt
```

Při překladu bylo nezbytné odstranit ladící flagy, protože přes vysoké číslo verze trpí unionfs poměrně vysokou nestabilitou. Občasné kernel oops, nebo dokonce kernel panic jsou při použití tohoto souborového systému vcelku časté.

4.2.5 Ještě jiná cesta – tmpfs

Z různých důvodů se mnoho distributorů rozhodlo do své distribuce podporu pro zápis nezařadit. Někteří nevěří méně stabilnímu UnionFS a pro některé je podpora pro zápis zbytečná. Díky historické praxi v unixech, která odděluje adresáře podle účelu, nikoli podle aplikace, je jejich úloha o dost zejdnodušená. Bez ohledu na výstřelky typu GoboLinux se tato praxe velmi osvědčila. Například při použití v síti, anebo právě při vytvoření LiveCD.

V klasické unixové hierarchii existují tři adresáře, do nichž je nutné povolit zápis. Jsou to `/etc`, `/tmp` a `/var`. V závislosti na účelu distribuce se ještě může jednat o `/opt` a `/home` (respektive `root`). Ty je proto nutné při startu rozbalit do paměti. Posloupnost kroků může vypadat následovně:

```
mkdir -p $ROOT/etc
mount -t tmpfs -o size=5M none $ROOT/etc
tar -cjf $CD_ROOT/$etc.tar.bz2 -C $ROOT/
```

Souborový systém *tmpfs* vytvoří ramdisk v operační paměti, do něhož je nakopírován obsah adresáře `/etc`.

4.3 Flashové souborové systémy

Po komprimaci a simulaci zápisu máme další specializované souborové systémy. Pro Flash paměti, kde mají za úkol především minimalizovat počet zápisů a tím zvýšit životnost těchto pamětí.

4.3.1 JFFS

Zkratka znamenající *Journaled Flash File System*, což je první souborový systém specializovaný na Flash paměti typu NOR. Pracuje naprosto odlišným způsobem, než klasické souborové systémy určené pro magnetické disky. Změny adresářové struktury probíhají nejprve v paměti, odkud se postupně zapisují. Všechny změny se zapisují způsobem, že nový soubor je zapsán do volného místa a je uvedno umístění nové verze. V případě, že dojde volné místo, spustí se garbage collector, který uvolní prázdná místa. Souborový systém se rovněž snaží zápisy pravidelně rozporostřít na celý disk tak, aby nedocházelo k exponování pouze jedné části paměti.

4.3.2 JFFS2

Protože JFFS byla prvotina a jako taková obsahuje některé závažné návrhové chyby, které mohou v jistých situacích vést ke zvýšenému optřebování paměti. Proto brzy vznikl jeho druhá verze. Pravděpodobně nejpodstatnějším vylepšením je to, že se do žurnálu zapisují pouze změny oproti starší verzi, nebo příznak, že došlo ke smazání souboru. Ve starší verzi se soubory zapisovaly neustále.

- Podpora pro NAND paměti
- Pevné odkazy (nebyly ve starší verzi dostupné)
- Komprese, poskytuje tři kompresní algoritmy. Jmenovitě zlib, rubin a rtime
- Lepší výkon, minimalizuje zbytečné I/O a počet volání garbage collectoru
- Odolnost proti výpadkům v napájení (i v situaci, kdy probíhá gc)

4.3.3 YAFFS

Jehož název značí *Yet Another Flash File System* byl prvním specializovaným souborovým systémem pro paměti typu ROM. Existuje ve dvou verzích 1 a 2, které jsou více méně zpetně kompatibilní. Druhá verze se liší především podporou novějších flash zařízení s většími stránkami. Navíc má abstraktnější definici NAND paměti, takže funguje na pamětech s odlišnou geometrií, pravidly pro označení špatných bloků a podobně.

5 Linuxové LiveCD a jejich účel

V první části jsem uvedl některá možná použití LiveCD, ovšem jejich skutečné možnosti použití jsou prakticky neomezené. Proto není asi jiný způsob, jak poznat šíři záběru těchto CD, než si představit některé zajímavé z nich.

5.1 LiveCD „velkých“ distribucí

Mnoho distributorů vytváří živé verze svých distribucí, aby je bylo možné vyzkoušet jejich produkty. Pro mě osobně je jejich přísnos minimálně diskutabilní, protože výhody, či nevýhody konkrétní distribuce se projeví až při delším používání. Navíc je volba té jediné „správné“ distribuce spíše emocionální, než racionální otázkou. Zajímavou možností je spojení LiveCD a instalátoru. Příkladem mohou být distribuce (K)Ubuntu. Kompilované distribuce Gentoo a LFS potom mají díky LiveCD základní prostředí pro tzv. *bootstrapping*.

Příkladem mohou být

Suse Linux Live	Live verze distribuce Suse
Mandriva Linux One	Live verze distribuce Mandriva
Ubuntu	LiveCD a instalační CD distribuce Ubuntu
Kubuntu	LiveCD a instalační CD distribuce Kubuntu
Gentoo LiveCD	Live verze distribuce Gentoo s instalátorem
Lfs LiveCD	Live verze distribuce Linux from scratch

5.2 Obecná LiveCD

Tato LiveCD bývají založena na některé z velkých distribucí, ale nemají za cíl propagovat svého předka. Jsou to jednoduše specializované Live distribuce, ovšem bez konkrétnější cílové skupiny. Podle mého názoru jsou jako „reklama na Linux“ vhodnější, nežli distribuce z předchozí skupiny.

ABC Linux 2005	Obsahuje offline obsah portálu abclinuxu.cz, zaměřena na běžného uživatele. Založena na Slax 5.0.5
Slax	Populární mini (190MB) distribuce Tomáše Matějíčka, vychází v několika verzích. Založena na Slackware. Významným plusem je její snadná rozšiřitelnost pomocí modulů
Knoppix	Asi první široce používaná živá distribuce, na níž je postaveno několik obecných distribucí. Založena na Debianu
Danix	Varianta Knoppixu vyvíjená v českých zemích. Podle anglických stránek projektu se zdá, že znovu ožil, na českých se od roku 2004 nic nepohnulo
Kanotix	Velmi povedená německá distribuce a orientovaná na běžné uživatele, bohužel bez přímé podpory češtiny. Základem je Knoppix
Damn Small Linux	Minidistribuce (méně jak 50MB), založená na Knoppixu. Problematická je její špatná rozšiřitelnost.

5.3 Bezpečnostní

Díky tomu, že se dá snadno vyloučit jakákoliv modifikace CD, přináší použití LiveCD zvýšenou bezpečnost. Velmi častá je také tvorba specializovaných distribucí určených pro bezpečnostní testy v síti[5]. Tyto distribuce bývají často minimalistické, zato obsahují kompletné „hackerskou“ výbavičku, jako nástroje `crackerjohn`, `nmap`, nebo třeba `nessus` a mnoho dalších. Některé dokonce obsahují i sbírky známých exploitů. Zde jsou některé z nich

Arudius	Specializuje se na průnikové (penetrační) testy. Jejím základem je Slackware (Zenwalk) a seznam nástrojů dosahuje úctyhodných 139 položek! To vše na 208 MB
SlackPen Beta 0.3	Jak už název napovídá, je založena na Slackware a linux-live skriptech. Poslední verze má 322 MB
BackTrack	Tato distribuce vznikla sloučením distribucí Whax a Auditor. Je založena na distribuci Slax a je tedy silně rozšiřitelná. Zajímavostí je, že obsahuje i sbírky exploitů, které je možné použít na testovaný systém.

[5] A samozřejmě je druhá strana může používat k průnikům

JayOS	Není zaměřená jenom na průniky. Data jsou navíc v paměti šifrována a navíc obsahuje User Mode Linux, nebo Qemu. Postavena na LFS, iso je velké asi 600 MB
nUbuntu	Ani současná hvězda distribucí, Ubuntu, nesmí být ochuzena o bezpečnostní a testovací variantu.

5.4 Serverové a síťové

Většina z nich trochu souvisí s předchozí skupinou. Nejčastěji se jedná specializované firewally, anebo routery. Ale je možné najít i zajímavější distribuce

Server Site	Na pouhých 129 MB obsahuje Apache web server s podporou pro PHP, Perl, Python, MySQL a PostgresSQL. Mimo to obsahuje vsFTPD, MySQL a PostgreSQL servery, OpenSSH server, phpMyAdmin a nakonfigurované iptables. Jako desktopové prostředí bylo vybráno xfce. Založena na Gentoo.
ZeroShell	Poskytuje mnoho síťových nástrojů, jako Kerberos, LDAP, NIS, RADIUS, certifikáty X509, lze použít jako služba pro Windows Active Directory, firewall a paketový filtr, routování, podporu pro VPN a mnoho dalších.
Devil Linux	Původně specializovaná distribuce na router/firewall se časem rozrostla, takže dnes obsahuje proxy server, http server a jiné. Nemá žádný X server, jádro má aplikován GRSecurity patch a programy jsou přeloženy s GCC Stack Smashing Protector.
Smoothwall	Miniaturní (44 MB) firewall, plně nastavitelný z webového rozhraní, takže jeho konfiguraci zvládne i člověk, který sice rozumí sítím, ale neovládá Linux a jeho příkazovou řádku.

5.5 Demonstrační

Posuneme se od vážného tématu sítí a bezpečnosti k příjemnějšímu tématu. Občas tvůrci některého programu vydají LiveCD, které umožní vyzkoušet jejich produkt bez toho, aby bylo nutné jej instalovat do systému.

Kororaa	LiveCD, díky němuž si mohla spousta lidí vyzkoušet divy Xgl, pokud jim to grafická karta dovolila. Instalace Xgl totiž představuje značný zásah do systému a navíc zatím nefunguje zcela spolelivě, kvůli nedostatkům v binárních ovladačích.
Elive CD	CD, které prezentuje novou verzi zajímavého grafického prostředí Enlightenment E17, které si ihned získalo spousty příznivců, stejně jako spousty odpůrců. Jisté je, že se od starší verze E16 velice liší.

Amarok Live	V „dávných“ dobách, kdy světu vládl amaroK verze 1.2.x se autoři rozhodli ukázat novinky v právě dokončené verze 1.3 právě pomocí LiveCD.
lg3d-livecd	Technologie Sunu Looking Glass je na trhu už nějaký čas, přesto se zatím příliš nerozšířila. Změní toto LiveCD založené na Gentoo zásadním způsobem situaci?

5.6 Multimediální

Nejen průhlednými a vlnícími se okénky, živ jest člověk. Proto také existuje mnoho specializovaných distribucí, které využívají síly multimédií v Linuxu. Obecně jediným problémem se tak stávají DRM ochrany, kvůli nimž je nutné používat i jen jako hloupý přehrávač ten jediný správnýTM operační systém. Všechny tyto distribuce mají skvělou autodetekci hardware a ovládají se pomocí grafického menu, takže je zvládne i běžný uživatel. Problémy mohou způsobovat české titulky ve Windows kódování.

LiMP	Multimediální přehrávač, podporuje většinu známých formátů, jako avi, wma, nebo QuickTime. Přehrává (S)VCD, i DVD. Navigace je pomocí menu a i připojení k síti se dá nastavit z grafického rozhraní
WOMP	Další z řady multimediálních LiveCD, pro přehrávání používá framebuffer a vidix[6] pro akceleraci
Movix	Jeho zajímavostí je verze eMovix, kterou stačí připálit k filmu CD se potom distribuuje přímo s přehrávačem. Ale obsahuje i verze (24 MB), která se zavede do paměti a potom už jen čeká na film
GeeXboX	Je miniaturní LiveCD založená na mplayeru, která promění váš počítač v multimediální centrum. Ovládání je také grafické.

5.7 Ostatní

Ani výše uvedená CD nepokrývají všechny možnosti. Proto jsem vybral některá zajímavá využití, která nezapadají do žádné z výše uvedených kategorií.

Harv's Hamshack Hack	Specializovaná distribuce pro radioamatéry. Zakládá si na tom, že není určena pro geeky, ale pro běžné uživatele, kteří nemají s Linuxem žádnou zkušenost. Založena na Knoppixu
Arch-egis	Arch Environmental/Geographical Information Systems) Project, alespoň tak zní název. Domácí stránka byla během psaní tohoto příspěvku bohužel mimo provoz. Tato distribuce by měla být zaměřena na OpenSource GIS aplikace a jejich vývoj.

Quantian	Distribuce specializovaná na numerické výpočty v mnoha oblastech. Kombinuje Knoppix a clusterKnoppix a je rozšířen o openMosix. Jeho specialitkou je tudíž podpora distribuovaných výpočtů. Umí zárobeň i vytvářet clustery typu Beowulf. Díky tomu se maximálně hodí do vědecké a technické praxe, kde může poskytnout množství levného výpočetního výkonu.
Wolwix	Jeho podtitul Game Edition mluví za vše, jedná se o sbírku asi 50 rozličných her.
SuSE Linux 10 for Windows	Suse Linux jako screensaver ve Windows? Zdá se, že pro vývojáře IBM není ani tohle nemožné. K dispozici je torrent pro stažení. Není to sice přímo LiveCD, ale s tématem to vcelku souvisí.

6 Závěr

Linuxová LiveCD jsou velmi rozsáhlou oblastí. Ať už z hlediska jejich technického řešení a použitých prostředků, nebo z hlediska jejich použití. Nepřestávají mě fascinovat možnosti jejich použití, které se na světě neustále objevují. Navíc na světě existuje mnoho nástrojů pro snadnou tvorbu LiveCD, například

- linux-live – skripty, jemiž je tvořena distribuce Slax
- mkdanix – nástroje od tvůrců distribuce Danix
- HOWTO build a LiveCD – návod na to, jak vytvořit LiveCD
- dscdbuilder – Debian from Scratch CD Builder

Ty ovšem také trochu přispívají k tomu, že Live distribucí Linuxu snad existuje víc, než těch klasických. Při zkoumání a používání LiveCD jsem také našel jistá specifika, kterými se liší od těch klasických.

6.1 Méně je více

Ovšem u LiveCD všeobecně platí toto jednoduché pravidlo. Ukazuje se, že menší a specializované distribuce bývají hodnoceny lépe, než ty, které se snaží naplnit CD až po okraj různým softwarem. Takový malý Slax potom získává oproti třeba Knoppixu body i díky své větší rychlosti, kdy se nemusí neustále prohledávat

rozsáhlý komprimovaný souborový systém. Navíc tyto projekty často bývají *one-man-show*[7], takže není v silách jednoho člověka vyladit velkou spoustu SW tak, aby výsledek stál za to.

6.2 Dobré výchozí nastavení

Další ze specifických věcí pro LiveCD. Zatímco spousta klasických distribucí přímo počítá s tím, že je uživatel bude po instalaci nastavovat, u LiveCD je tohle zabiják. Takže i když se většina z nich honosí tím, že nejsou učeny pro běžné uživatele, tak musí být ve výchozím stavu plně funkční, protože neustálé předělávání výchozí konfigurace, nefunkční položky v menu, neustálé odškrtávání tipů dne, ... se časem stane příčinou uživatelova odchodu k distribuci, která je vyladěná. S tím souvisí i množství software, čím méně je ho, tím lépe snadněji se dá systém vyladit.

[7] Další z cizojazyčných slov, jimiž autor ukazuje své rozsáhlé vzdělání. Slovo jednomužné by bohatě stačilo

Získávání informací z logů s využitím shlukové analýzy

Marek Kumpošt

E-mail: xkumpost@fi.muni.cz

Abstrakt

Cílem příspěvku je představit techniku zpracování kontextových informací. Kontextová informace vznikne při každé interakci uživatele v systému a může se jednat např. o čas odeslání dat, frekvenci komunikace, velikost dat (v situaci, kdy např. uživatel odešle obyčejný e-mail). Vhodným zpracováním kontextových informací můžeme získat „typické chování“ uživatele, což může být v některých situacích citlivá informace. Jako vhodná technika pro modelování typického chování subjektů se jeví tzv. shluková analýza, což je statistická metoda zpracování dat. Použitím shlukové analýzy můžeme kategorizovat subjekty do shluků (clusters) takovým způsobem, že vlastnosti subjektů v rámci jednoho shluku jsou maximálně podobné a vlastnosti subjektů v rámci různých shluků jsou maximálně odlišné.

V příspěvku si krátce představíme grafový model PATS (Privacy Across The Street), který byl navržen s cílem modelovat dostupné kontextové informace a hledat mezi nimi „zajímavá“ spojení. Cílem je zjistit jaké „soukromé“ informace můžeme získat zpracováním kontextových dat. Pokud totiž máme informaci o typickém chování uživatele, tak jej následně lze v systému snadněji rozpoznat. Metoda shlukové analýzy by měla posloužit jako první filtrace „nezajímavých“ dat a detailní zpracování zbylých dat bude následně provedeno s použitím modelu PATS.

V závěrečné části příspěvku si ukážeme výsledky shlukové analýzy při zpracování logu univerzitní síťové komunikace a uvedeme další možné oblasti pro využití této statistické metody zpracování dat.

1 Model PATS

Cílem modelu PATS (Privacy Across The Street) je zobrazení kontextových informací a vztahů mezi nimi. Kontextovou informaci si lze představit jako záznam o aktuální činnosti uživatele – čas přihlášení do systému, navštívení webové

stránky, navázání ftp spojení aj. Pokud vezmeme v úvahu veškeré možné kontextové informace vzhledem k určitému uživateli, můžeme takto získat jakýsi „popis typického chování uživatele“ (např. víme, že daný uživatel velmi často navštěvuje určité webové stránky a velmi často vzdáleně pracuje s nějakou ip adresou). Snahou modelu je reprezentovat všechny relevantní kontextové informace a následným vyhodnocením vyvodit závěry o soukromí uživatele(ů).

Množiny kontextových informací jsou v grafu reprezentovány jako uzly a vztahy mezi uzly jsou reprezentovány hranami. Hrany mohou být ohodnoceny pravděpodobnostmi, které reflektují vztahy mezi různými uzly. Cílem je poté nalezení nejpravděpodobnějších cest v grafu a spojení dvou nebo více uživatelů dohromady. Detailnější informace a formální definice lze najít v původním článku [3].

Vzhledem k tomu, že cílem uvedeného modelu je velmi detailní analýza kontextových informací, tak celkové množství informací, které budou zobrazeny, může pro velké množství vstupních objektů učinit výsledný graf značně nepřehledný. Z tohoto důvodu jsme začali přemýšlet o možnostech „filtrace“ vstupních dat takovým způsobem, abychom ve výsledku získali řádově menší množství objektů. Vzhledem k tomu, že cílem je nalezení spojení mezi více objekty, by bylo výhodné při prvním filtrování vstupních dat vybrat pouze takové objekty, které vykazují určitou míru „podobnosti“. Takto se zbavíme „nezajímavých“ dat a detailnější analýze podrobíme jen ty objekty, které již vykazují určité vlastnosti. Jako vhodným kandidátem pro omezení velikosti vstupních dat se ukazuje metoda shlukové analýzy.

2 Shluková analýza

Shluková analýza je statistická metoda pro klasifikaci objektů do disjunktních skupin. Metodu lze charakterizovat následujícími třemi vlastnostmi:

- Shluková analýza zahrnuje několik různých algoritmů a metod použitých pro shlukování objektů se stejnými vlastnostmi do odpovídajících kategorií.
- Cílem shlukové analýzy je třídění objektů do různých kategorií takovým způsobem, že „stupeň podobnosti“ je maximální pro objekty v rámci jedné skupiny a minimální pro objekty v rámci různých skupin (shluků).
- Shluková analýza může být využita pro odhalení struktur ve vstupních datech bez další interpretace.

2.1 Metody shlukování

Metod použitých ve shlukové analýze je několik, zde zmíníme pouze tři nejdůležitější [2].

- **Spojování (stromové spojování)** je metoda, která spojuje objekty do postupně větších shluků za použití nějaké míry vzdálenosti. Typický výsledek této metody shlukování je hierarchický strom.
- **Dvoucestné shlukování (blokové shlukování)** seskupuje objekty do shluků, které jsou následně považovány jako jednotlivé případy. Následně je aplikováno standardní shlukování.
- **Shlukování metodou k-průměrů** vytvoří přesně k shluků, které jsou maximálně odlišné.

Dále potřebujeme nějaké kritérium pro výpočet vzdálenosti mezi objekty a vytvoření shluků. Existují metriky jak pro jednodimenzionální, tak pro vícedimenzionální objekty. Nejběžnější metoda pro vícedimenzionální objekty je *Euklidovská metrika*, což je geometrická vzdálenost mezi dvěma body ve vícedimenzionálním prostoru.

$$vzdálenost(x, y) = \sqrt{\sum_i (x_i - y_i)^2} \tag{1}$$

Jiné běžně používané metriky jsou:

- *Squared Euclidean* ve srovnání s euklidovskou vzdáleností se klade větší váha na izolované objekty.

$$vzdálenost(x, y) = \sum_i (x_i - y_i)^2 \tag{2}$$

- *City-block (Manhattan)* poskytuje často podobné výsledky jako euklidovská vzdálenost, ovšem potlačuje vliv odlehlých pozorování (outliers).

$$vzdálenost(x, y) = \sum_i |x_i - y_i| \tag{3}$$

- *Chebychev* se používá v situacích, kdy je potřeba odlišit objekty, které jsou odlišné třeba jen v jedné dimenzi.

$$vzdálenost(x, y) = \max_i |x_i - y_i| \tag{4}$$

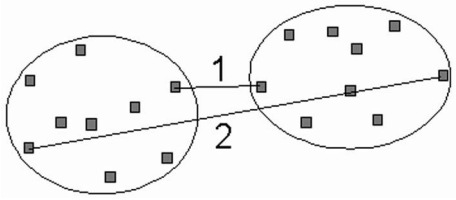
2.2 Hierarchické shlukování

Tato metoda vytvoří, na základě matice vzdáleností, hierarchickou posloupnost rozkladů tak, že celý proces začneme s n shluky, kde každý z nich sestává pouze z jednoho objektu. Následně dojde k nalezení shluků, které mají nejmenší vzdálenost a tyto spojíme do jednoho. Celý proces končí v okamžiku, kdy dojde ke spojení všech objektů do jednoho shluku. Je potřeba definovat, jakým způsobem se určuje vzdálenost mezi shluky. Uvádíme několik základních postupů:

- *Metoda nejbližšího souseda* – Vzdálenost mezi shluky je určena jako minimální vzdálenost mezi objekty, kde každý objekt je z jiného shluku. Nevýhodou tohoto přístupu je, že vzdálené objekty jsou často zařazeny do stejného shluku.

- *Metoda nejvzdálenějšího souseda* – Odlišnost mezi dvěma skupinami odpovídá největší možné vzdálenosti objektu ze shluku i a objektu ze shluku j. Tato metoda má tendenci produkovat velmi těsné shluky.

- *UPGMA (unweighted pair group method using averages).* Vzdálenost mezi dvěma shluky je průměrná vzdálenost mezi všemi možnými dvojicemi objektů z těchto shluků. Tato metoda je obecně více preferovaná než metoda nejvzdálenějšího souseda, protože výsledek je založen na větším množství vstupních dat.



Obr. 1 Určování vzdálenosti mezi shluky

2.3 Použití shlukové analýzy

Nyní se podíváme, jak můžeme aplikovat metody shlukové analýzy při práci s kontextovými informacemi. Jako úplně první krok je nutné identifikovat, jaké kontextové informace budou použity (tj. budeme např. sledovat přihlášení do systému, navštívené webové stránky, použití FTP, odeslání emailové zprávy, apod.) pro popis chování uživatelů. V následujícím kroku pak pro každého uživatele zjistíme celkový počet provedení dané akce (např. kolikrát uživatel odeslal emailovou zprávu nebo použil FTP), tj. že např. uživatel A během jednoho dne třicetkrát navštívil stránku vyhledávače jyxo, stokrát se přihlásil do systému apod. Tímto postupem si připravíme množinu vektorů, jeden pro každého uživatele, kde máme záznamy typu uživatel u_i provedl c_j operací typu o_j. Každý vektor pak vyjadřuje typické chování uživatele v rámci např. jednoho dne.

Získaná množina vektorů je soubor vstupních dat pro zpracování metodou shlukové analýzy. Výsledky procesu shlukování nám poskytnou informace o „podobnosti“ v chování uživatelů (neboť uživatelé, jejichž vektory chování jsou od

sebe málo vzdálené, budou umístěni do jednoho shluku). S určitou pravděpodobností pak bude možné tvrdit, že u uživatelů v rámci určitého shluku lze očekávat podobné rysy v jejich chování. Jiný typ hypotézy může být podložen zjištěním, že v systému existují např. dva objekty, které mají velice podobné chování. Interpretace tohoto výsledku může být taková, že tyto dva objekty jsou ve skutečnosti jen jedna fyzická osoba, která v rámci systému používá dva počítače, na kterých vykazuje téměř totožnou aktivitu (typicky stolní počítač a notebook připojený do sítě).

2.4 Praktická aplikace

Vhodné prostředí pro aplikaci výše uvedeného způsobu zpracování logů je velmi důležité. Příkladem „vhodného“ prostředí je např.:

- Sledování síťového provozu a vyhodnocování „chování“ zdrojových IP adres.
- Různé chatovací nebo freemailové služby, kde je cílem zjistit, zda se některé pseudonymy nebo emailové adresy chovají podobně.

Obecně je vhodné jakékoliv prostředí, kde je uživatelům umožněno vystupovat pod různými identitami. Na druhé straně v prostředí, kde uživatel může využívat pouze jeden účet, může metoda shlukování odhalit možné vyzrazení a zneužití cizího hesla nějakým jiným uživatelem v systému.

3 Příklad

V této části ukážeme aktuální výsledky při zpracování globálního síťového logu univerzitní sítě Masarykovy univerzity metodou shlukové analýzy. Globální síťový log obsahuje informace o veškeré odchozí komunikaci mimo univerzitní síť. Data jsou uložena v databázi a každý řádek obsahuje množství informací z nichž následující výčet je relevantní pro naše účely: zdrojová/cílová IP adresa, čas začátku komunikace, zdrojový/cílový port a použitý protokol. Počet záznamů, který je vygenerován během jednoho dne je cca 120 000 000. Toto je obrovské číslo a pro efektivní zpracování je potřeba vybrat pouze nějakou část dat. Jinak by analýza trvala poměrně dlouhou dobu.

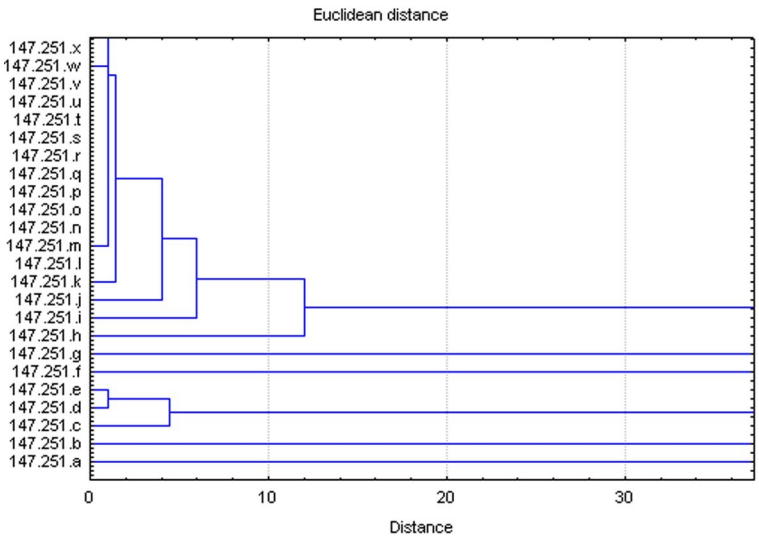
V našem příkladě se zaměříme na zdrojové IP adresy fakulty informatiky a bude nás zajímat komunikace přes http protokol (port číslo 80). Počet záznamů vyhovujících těmto kritériím je potom zhruba 1 000 000. Tato tabulka je potom použita pro výpočet množiny vektorů. Jak bylo řečeno v části 2.3, je potřeba stanovit, jaké kontextové informace budeme u každé zdrojové IP adresy sledovat. V tomto případě se bude jednat o nejnavštěvovanější cílové IP adresy, tj. webové

servery. V našem příkladu budeme uvažovat 50 nejnavštěvovanějších cílových adres.

Tabulka 1 obsahuje vektory popisující chování daných IP adres (zde je vidět pouze část z celé tabulky, která má rozměry $50 \times 1\,000$).

Tabulka 1 Tabulka vektorů

	dst 1	dst 2	dst 3	dst 4	dst 5	dst 6	...
147.251.aaa.bbb	5	16	3	20	14	0	...
147.251.ccc.ddd	0	0	0	0	0	0	...
147.251.eee.fff	8	14	10	88	4	0	...
147.251.ggg.hhh	0	0	0	0	0	13	...
147.251.iii.jjj	120	0	0	0	0	0	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱



Obr. 2 Část dendrogramu

Tato tabulka je použita jako vstupní množina dat pro shlukovou analýzu. Na obrázku 2 je pak vidět část celého stromu (dendrogramu), kde lze vidět, že IP adresy `e` a `d` jsou spojeny na velmi nízké úrovni. Tento výsledek indikuje,

že oba vstupní vektory jsou vzájemně velmi „podobné“. Pokud se podíváme zpět do tabulky 1, tak popsaná situace odpovídá IP adresám `147.251.aaa.bbb` a `147.251.eee.fff`. Zde můžeme vidět, že vektory popisující tyto dvě adresy jsou velmi podobné. Navštěvované cílové IP adresy se shodují, odlišnosti jsou zde pouze v množství návštěv.

4 Závěr

Vzhledem k tomu, že je tento projekt teprve ve svém začátku, tak místo závěru nastíníme možné směry další práce. Původně jsme si nebyli jisti, zda tato technika může odhalit nějaké zajímavé informace. Z příkladu, který je založen na reálných vstupních datech, je vidět, že metoda shlukování je pravděpodobně vhodný nástroj pro hledání podobností v takto velkém množství dat.

Vzhledem k velikosti vstupní databáze je nutné najít efektivní způsob zpracování a získávání množiny vstupních vektorů. Je nutné aplikovat nějaká omezující pravidla pro zmenšení velikosti vstupních dat takovým způsobem, že neztratíme žádné podstatné informace. Jako vhodný se zatím jeví přístup, kdy se zaměříme jenom na určitou část sítě (např. fakulu a koleje, na kterých jsou ubytovaní studenti) nebo omezíme rozsah cílových portů (80, 21, ...).

Další důležitou oblastí je poměr mezi počtem pozorovaných objektů a množstvím použitých kontextových informací. Pokud bychom např. v našem případě použili 100 nejnavštěvovanějších cílových adres, tak shlukování skončí s výsledkem, kdy každá zdrojová IP adresa bude v samostatném shluku – spojení získáme teprve v okamžiku, kdy snížíme počet nejnavštěvovanějších destinací.

Bylo by též vhodné nějakým způsobem vizualizovat aktivitu sledovaných objektů, abychom získali lepší pohled na „strukturu“ vstupních vektorů a na základě toho např. rozhodnout o počtu použitých kontextových informací.

Poslední věcí je korektní interpretace výsledků a diskuse vlivu použitých kontextových informací na výsledky shlukování.

Na úplný závěr bychom rádi poděkovali správcům univerzitní sítě za poskytnutí přístupu k databázi logů.

Literatura

[1] JOHNSON, A. R., WICHERN, W. D. *Applied Multivariate Statistical Analysis.* 4th. Upper Saddle River, New Jersey : Prentice Hall, 2002.

[2] LUKASOVÁ, A., SARMANOVÁ, J. *Metody shlukové analýzy.* Praha : SNTL – Nakladatelství technické literatury, 1985.

[3] MATYÁŠ, V., CVRČEK, D. On the role of contextual information for privacy attacks and classification, *Privacy and Security Aspects of Data Mining Workshop, IEEE ICDM*, Brighton, UK, November 2004.

[4] RAZZAQUE, M. A., DOBSON, S., NIXON, P. Categorization and Modelling of Quality in Context Information, *School of Computer Science and Informatics University College, Dublin IE*, 2005.
http://csiweb.ucd.ie/UserFiles/publications/1124274826156.pdf

PERL a síťová zařízení s administrací přes webové rozhraní

Štěpán Bechynský

E-mail: Stepan.Bechynsky@microsoft.com

Velké množství síťových zařízení, jako jsou ADSL modemy, WiFi Access pointy, tiskárny a další, obsahují interní webový server a jednoduchou webovou aplikaci sloužící pro jejich administraci. Nastavení zařízení pomocí takovéto webové aplikace je většinou jednoduché, obsahuje různé průvodce, ale bohužel bývá také jedinou možností jak dané zařízení administrovat.

Problémy nastanou ve chvíli, kdy potřebuje něco na zařízení změnit, ale nemáme k němu přístup pomocí webového prohlížeče s podporou grafiky a skriptování. Potíže nastanou i ve chvíli, kdy potřebujeme jeden úkon provádět pravidelně nebo několikrát po sobě s drobnou změnou. V tuto chvíli je webové rozhraní naprosto nevhodné.

Řešením je vytvoření konzolové aplikace, kterou můžeme snadno parametrizovat, zařazovat do konzolových skriptů a snadno spouštět z příkazové řádky. Jako velmi vhodný se jeví PERL díky modulu WWW::Mechanize a modulů pro parsování HTML (HTML::TokeParser, HTML::TreeBuilder).

Modul WWW::Mechanize, zjednodušeně řečeno, simuluje webový prohlížeč. Umožňuje procházet po stránkách tak, jako kdyby po nich procházel uživatel v klasické webové prohlížečce. Modul WWW::Mechanize nepodporuje skriptování, což může způsobit při vytvářeni konzolové varianty webového rozhraní potíže. Pokud webové rozhraní používá skriptování je třeba projít kód skriptů, zjistit co dělají a snažit se je obejít. Častým úkolem skriptů bývá nastavení neviditelných polí formuláře na hodnoty podle toho na jaké tlačítko se kliknulo nebo kontrola vstupu uživatele. Pokud webové rozhraní používá ActiveX, tak máme většinou smůlu a administrátor se ukliká.

Moduly HTML::TokeParser a HTML::TreeBuilder slouží pro snadné získání informací z obsahu webové stránky. Velkou výhodou obou modulů je, že nevyžadují XHTML a poradí si i s HTML kódem, který plně nedodržuje standardy.

Převod webového rozhraní do konzole vypadá následovně:

1. Zjistíme, jakým způsobem se k webové aplikaci přihlašuje a jak funguje zabezpečení. Velmi důležité je vědět, kde se uchovává informace o tom, že

jsem přihlášen (cookies vs. sessionid v URL). Podle toho se rozhodneme, zda můžeme požadovanou stránku volat přímo, nebo se k ní musíme „proklikat“.

2. Analyzujeme strukturu stránky, s kterou potřebujeme manipulovat – formuláře, rámy, tabulky, ...
3. Převedeme klikání po obrazovce do volání metod modulu WWW::Mechanize.
   (a) Nastavení jména a hesla pro basic http autentifikaci – `$mech->credentials(...)`
   (b) Natažení požadované stránky – `$mech->get(...);`
   (c) Kliknutí na odkaz – `$mech->follow_link(...)`
   (d) Nastavení aktuálního formuláře – `$mech->form_number(...);`
   (e) Nastavení hodnoty ve formuláři – `$mech->field('name', $name);`
   (f) Odeslání formuláře – `$mech->submit();`
4. Zjistíme, zda se akce povedla analýzou vrácené stránky. Typicky dohledáním části textu ve vrácené stránce.
   (a) Obsah stránky – `$mech->content();`
   (b) Pomocí HTML::TokeParser nebo HTML::TreeBuilder se dostaneme na místo ve stránce, kde má být požadovaný text.
   (c) Dohledání textu např. pomocí regulárních výrazů.

Na závěr bych doporučil knížku Spidering Hacks, vydanou v nakladatelství O'Reilly, která je celá věnovaná problematice „surfování“ pomocí PERL skriptů.

Xen – virtuální hrátky v praxi

Michal Švamberg, Štěpán Kadlec

E-mail: svamberg@civ.zcu.cz, steve@students.zcu.cz

Klíčová slova: virtualizace, xen, migrace

Virtualizace se dříve v rutinním provozu příliš nepoužívala, zvláště z důvodů potřeby šáhnout ke speciálnímu hardwaru nebo komerčnímu softwaru. Na poli svobodného softwaru existovaly virtualizační nástroje, které převážně na platformě i386 dokázaly vytvořit virtuální stroj. Jejich velkou nevýhodou byla pomalost, nebo velmi úzké zaměření. Do vzniku projektu Xen neexistovalo řešení virtualizace založené na svobodném softwaru, které by bylo možné nasadit do běžného provozu na stranu serverů.

1 Virtualizace

Jak již bylo naznačeno, v příspěvku se budeme nadále věnovat pouze softwarovému řešení virtualizace. Zde existují tři základní přístupy:

plná virtualizace Simulací kompletního hardware se vytváří virtuální stroj, na němž běží hostovaný systém. Díky tomuto přístupu nevyžaduje hostovaný systém žádné úpravy; lze takto dokonce provozovat systémy určené pro jinou architekturu než je fyzická.

Nevýhodou tohoto pojetí virtualizace je vysoká režie, vzniklá vlivem softwarové emulace CPU instrukcí. Existuje několik optimalizací (dynamická rekompilace, přímé vykonání instrukce), avšak režie je i tak značná.

Zástupci plné virtualizace jsou VMware, VirtualPC a QEMU.

nativní virtualizace Virtuální stroj simuluje pouze minimální část hardware, ke zbytku přistupují hostované systémy přímo. Jejich vzájemná izolace je zajištěna vytvořením bariér mezi těmito systémy. Hostované systémy opět nevyžadují žádné úpravy, musí však již odpovídat architektuře fyzického stroje.

Cestou nativní virtualizace se vydaly projekty Virtuozo, Vservers a Zones.

paravirtualizace Třetí skupina se pokusila spojit to lepší z obou předchozích přístupů. Virtuální stroj se nesnaží hardware simulovat, namísto toho poskytuje hostovanému systému speciální rozhraní, které k němu umožní přistupovat.

Pro tento případ je nutné hostované systémy přizpůsobit tak, aby dokázaly využít virtualizační rozhraní. Nutnost takové úpravy je určitou komplikací tohoto řešení, nicméně univerzalita a výkon, kterého lze tímto přístupem dosáhnout, jednoznačně předčí oba výše zmíněné.

Paravirtualizace je použita v projektech UML a Xen.

2 Xen

Xen je *monitor virtuálních strojů* pro `x86`[1] architekturu. Umožňuje bezpečný provoz několika virtuálních strojů – každý se svým vlastním operačním systémem – na jediném fyzickém stroji. Výkon virtuálních strojů spravovaných Xenem se blíží jejich výkonu při nativním chodu.

Xen vznikl a je vyvíjen na University of Cambridge za podpory společností EPSRC (UK Research funding Council), Intel, Network Appliance a XenSource Inc.

Xen byl původně zamýšlen jako podkladní vrstva pro klastrová řešení. A to jak například pro vývoj aplikací, kdy na jednom železe lze připravovat a testovat klastrové úlohy, tak hlavně pro možnosti správy jednotlivých nodů, aby bylo možné vzdáleně nody uspávat či je přesouvat mezi fyzickými stroji. Projekt ale našel daleko širší uplatnění.

Ve své podstatě je Xen sada patchů pro jádra Linuxu či OpenBSD a několik skriptů, kterými lze virtualizaci ovládat. Základním kamenem je hypervisor. Hypervisor je malá část jádra, která má za úkol řídit nedělitelné operace. Je to zvláště plánování procesorového času, řízení přerušení, mapování stránek či bootovací sekvence stroje a start *Domain-0*, tzv. privilegovaného stroje.

Pro řízení Xenu je třeba mít odpovídající prostředí, nejlépe s plnohodnotným operačním systémem a ovladači k hardwaru. Tuto službu nám zajišťuje *Domain-0*, který se startuje po zavedení hypervisora. Ovladače se v pojetí Xenu dělí na dvě skupiny: *front-end* a *back-end*. Ovladače typu *back-end* spravují přístup k fyzickému zařízení, tedy ke každému back-end ovladači přísluší nějaký běžný ovladač. *Front-end* ovladač je pak jakási trubka, která se z virtuálních strojů připojuje k back-end ovladači. Nejběžnější způsob je, že back-end ovladače jsou v Domain-0 a front-end ovladače ve virtuálních strojích. Vzhledem k tomu, že události v systému nám spravuje hypervisor, je možné, aby zařízení bylo „exportováno“ do samostatného virtuálního stroje, který bude přes back-end ovladač zpřístupňovat toto zařízení dalším virtuálním strojům.

[1] V plánu jsou i další architektury jako je PowerPC či ARM.

Aby to bylo ještě zajimavější, lze mezi sebou kombinovat různé druhy systémů. Například na Domain-0 s jádrem v2.6 lze provozovat linuxové stroje s jádry v2.4 i v2.6, a navíc si jako bonus pustit OpenBSD[2]. S nástupem Xenu 3.0 se do toho přimíchává 64bitová architektura, a také možnost spouštět nemodifikované systémy.[3] Tím se otevírá další možnost využití Xenu – vývojář softwaru již nebude potřebovat pro testování své aplikace několik strojů, ale bude mu stačit jeden s mnoha virtuálními stroji.

Xen je založen na principu paravirtualizace. S minimální režií je rozumně univerzální. Vzhledem k nutnosti zásahu do zdrojových kódů jej lze použít jen s operačnímy systémy, které jsou otevřené. Ve verzi Xen 3.0 přibyla podpora technologie IntelVT, s jejíž kombinací je možné provozovat též neupravené hostované systémy. Technologie IntelVT obsahuje podporu virtualizace v samotném hardware. Vývojáři Xenu pracují též na implementaci ekvivalentní technologie Pacifica od společnosti AMD.

2.1 Přínosy Xenu

Zásadním přínosem nasazení Xenu jsou peníze. Použitím jednoho stroje pro více hostovaných systémů se šetří hlavně za hardware, elektřinu na napájení, místě v servrovně, ale také na chlazení nebo zálohovacích zdrojích elektrického proudu. V případě, kdy na jednom stroji je hostováno více aplikací, roste hodnota takového stroje, a jeho výpadek může být kritičtější. Proto lze doporučit pro Xen zakoupit lepší hardware, nejlépe s redundantním zdrojem a vyšším počtem disků tak, aby bylo možné zapnout HW nebo SW RAID. Tím sice rostou pořizovací náklady, ale zároveň každý hostovaný systém dostává kvalitní základ.

Dalším přínosem je jednodušší správa. Snadno vytvoříte nový stroj, nemusíte napřed shánět železo, umístit jej někde a zapojovat k elektrickým rozvodům, ale ani hledat volný port na switchi. Velmi snadno lze zjistit stav hostovaných systémů a vytvořit nový stroj, včetně instalace.[4] Samozřejmě zrušení nebo přeinstalace virtuálního stroje je opět snažší a výrazně šetří čas obsluhy.

Přestože přínosy virtualizace jsou obecně stejné pro všechny virtualizační nástroje, má Xen jednu velkou výhodu – *migraci*. Pokud jsou pro Xen splněny i některé podmínky z pohledu IT infrastruktury, pak vám umožní migrování (přesunutí) virtuálních strojů. Tím lze zvýšit dostupnost hostovaných strojů, protože před plánovanou odstávkou lze virtuální stroje přemigrovat na jiný Xen server.

[2] Nesmíme zapomenout ani na podporovaný NetBSD či Plan9

[3] Je třeba HW podpory pro virtualizaci nemodifikovaných systémů a speciální zavaděč, který dokáže zavést operační systém v chráněném módu procesoru.

[4] Automatická instalace virtuálního stroje metodou FAI, včetně přípravy konfiguračních souborů pro FAI i Xen, zvládáme na ZČU do 5 minut.

Nasazení Xenu má i jedno negativum. Je jím riziko HW poruchy a tím odpadnutí všech hostovaných strojů. Tento problém je nevyzpytatelný a každý s ním má osobní zkušenost, ale lze jej omezit redundancí nejproblémovějších částí a poctivým zahořením jednotlivých komponent. Pokud je k dispozici migrace, pak v případě nečekaného pádu, lze virtuální stroj okamžitě nastartovat na jiném serveru a není třeba čekat na opravení původního stroje. To vše lze samozřejmě udělat vzdáleně.

2.2 Omezení

Z pohledu hostovaného operačního systému by se měl, dle konceptu virtualizace, chovat virtuální stroj, na němž systém běží jako reálný hardware. V praxi však přece objevujeme určité rozdíly, které jsou v tomto smyslu více či méně jistým omezením. Pro virtuální stroje spravované Xenem se jedná zejména o následující případy:

Nelze virtualizovat pevné disky jako celky, pouze jednotlivé oddíly, které jsou při konfiguraci virtuálního stroje vybrány pro export. Jinými slovy, pro virtuální stroj neexistuje zařízení `/dev/hda`, přestože např. zařízení `/dev/hda1` již ano. S tím je třeba počítat např. při instalaci systému na virtuální stroj (pokud se instalátor bude pokoušet rozdělovat disk na oddíly, dojde patrně k selhání).

Dalším omezením je možnost pouze uniprocesorových virtuálních strojů (neplatí již pro Xen 3.0). Hypervisor přiřadí při bootu virtuálnímu stroji jeden z fyzických procesorů a s ním je virtuální stroj svázán po celou dobu chodu (je možné vybrat a měnit konkrétní procesor, ovšem to lze provádět zatím pouze ručně – Xen dosud nemá implementováno automatické vyvažování zátěže mezi procesory).

2.3 Xen na ZČU

O nasazení virtualizačního nástroje se začalo uvažovat v polovině roku 2003, kdy se nahromadilo několik požadavků na samostatný testovací stroj, umístěný na serverovně. V té době chybělo skoro vše: místo v racku, záložní zdroj, a hlavně volné porty na switchích. Naštěstí v tu samou dobu byl již projekt Xen stabilizován v podobě řady 2.x. V porovnání s jinými virtualizačními nástroji měl vždy navrch, a to jak s cenou, tak rychlostí či dokumentací. Následovalo testování možností Xenu, ale i "kompatibility" s ostatním vybavením IT prostředí ZČU, zvláště AFS[5], FAI[6] a podpora VLAN (802.1q).

Na základě zkušeností byl zakoupen stroj s potřebným HW vybavením a rozhodnuto, že datové disky virtuálních strojů budou umístěny na Fibre Channel[7].

[5] Andrew File System – http://www.openafs.org/
[6] Fully Automated Installation – http://www.informatik.uni-koeln.de/fai/
[7] Fibre Channel – http://hsi.web.cern.ch/HSI/fcs/

První rok byly na takto vzniklém stroji provozovány virtuální stroje, kterým by případný pád nehrozil, převážně pro testování. Ovšem jak čas plynul, software určený k testování přecházel v ostrý provoz, a s ním narůstala také důležitost zachování provozu Xenu. Po naplnění kapacity se pořídil druhý stroj a život šel dál.

Nyní máme téměř tříletou zkušenost s provozem Xenu a směle jej mohu prohlásit za velmi stabilní. Nikdy jsme neřešili SW problém pádu hypervisora a naštěstí ani HW problémy. Nyní máme celkem 3 stroje, z nichž jeden je určen pro provoz ostrých virtuálních systémů, druhý pro testovací virtuální stroje. Třetí stroj byl koupen z grantu Fondu Rozvoje CESNETu na ověření migrace virtuálních strojů. Konfigurace tří serverů je téměř stejná:

- 2×CPU Xeon na 3,2 GHz s podporou Hyper Threadingu
- 4 GB RAM
- 2×Gbit ethernet (zatím používán jen jeden)
- 2×80 GB SATA disk (partitiony v SW RAIDu nástrojem `mdadm`)
- Fibre Channel adapter QLA2300
- redundantní zdroj

2.4 Problémy při nasazování

Při zavádění Xenu jsme se potýkali s několika zásadnějšími problémy. Jedna z významných potíží byla kompilace jádra do `.deb` balíků nástrojem `make-kpkg`. V tomto případě šlo zvláště o nastavení správných parametrů a přejmenování defaultních názvů patchovaných jader. Zprovozněním kompilace přes `make-kpkg` byl také úspěšně zkompilován modul klienta pro OpenAFS do balíčku.

Kompilace jádra a rozšiřujících modulů do debianích balíčků byla důležitá prerekvizita pro nasazení FAI. Ten se ale potýkal při rozdělování disků s problémem neexistence disku (partitiony ale existují). Tento problém byl překonán vlastním nástrojem pro rozdělování disků; nebylo potřeba zasahovat do kódu FAI.

Virtuální stroje měly být umísťovány na různých síťových segmentech dle typu provozované aplikace. V hypervisorovi je síťové rozhraní nastaveno tak, že obsahuje několik interních bridgů (pro každý segment jeden) a ke každému bridgi je přivedena daná tagovaná podsíť dle 802.1q. Xen při vytváření nového hosta sám, dle konfigurace, napojí síťové rozhraní na daný bridge.

2.5 Migrace

Jak jste si všimli, byla kompletně vynechána část o nástrojích a provozu Xenu. Důvodem je nošení dříví do lesa, tato část je totiž velmi dobře zdokumentována na stránkách projektu[8] a k dispozici je také LiveCD, na kterém si lze Xen vyzkoušet.

Proto rovnou odskočíme k tématu migrace, kterou v posledních dnech na ZČU testujeme. Xen dokáže hibernovat hosta do souboru (virtuální stroj je zastaven a obsah paměti uložen) a později jej znovu probudit. Pokud bychom jej chtěli probudit na jiném stroji, musíme splnit dvě podmínky:

- zachování disku jako blokového zařízení (SAN, NAS, iSCSI, ...),
- zachování síťového segmentu.

Právě tomuto se říká v pojetí Xenu migrace. Tu rozlišujeme na *off-line* a *on-line*. Rozdíl mezi nimi spočívá v rychlosti migrace a dostupnosti systému během migrace. *Off-line* migrace probíhá prakticky ve výše popsaných třech fázích: uspání, přesun obrazu paměti na nového hypervisora, probuzení. Během těchto fází není hostovaný systém dostupný, ale migrace je provedena bez zbytečného zdržování. Naproti tomu *on-line* migrace umožňuje jen minimální výpadek (méně než 1 vteřina), ale provedení migrace je časově i procesorově náročnější – musí se navíc hlídat zápisové operace do paměti.

Od vývojářů se pak můžeme těšit na další zdokonalování. Jedná se hlavně o možnost vyvažování zátěže mezi jednotlivé procesory nebo mezi fyzickými servery s využitím on-line migrace.

[8] Xen – http://xen.sf.net/

Embedded Systems and Open Source

Pavel Čeleda

E-mail: celeda@liberouter.org

Abstract

The article describes a development of embedded systems. It focus on possibilities of using open-source software during design, development and deployment of embedded systems. It points out particulars of these systems and how the open-source community handles it. It shows advantages and disadvantages of using open source in comparison with other available solutions for this application area.

Introduction

Wikipedia defines an embedded system [11] as a special-purpose computer system, which is completely encapsulated by the device it controls. An embedded system has specific requirements and performs pre-defined tasks, unlike a general-purpose personal computer. An embedded system is a computer-controlled system. The core of any embedded system is a microprocessor, programmed to perform a few tasks (often just single task). This denotes the main difference to the other computer systems with general purpose hardware and externally loaded software. Embedded systems are often designed for mass production.

Embedded systems are part of our everyday life. Examples of embedded systems range from consumer electronics to real-time controls for high-reliability systems like the nuclear power plant. Most embedded systems are designed to do specific task at a low cost. The development of embedded systems is very interdisciplinary and range from electronic engineering to computer science.

Like in the other areas the open source community had started some time ago its journey in the embedded world. Some companies and individuals are promoting open source as the solution to all woes of embedded world while others are more sceptic. The following text will try to explain some facts and myths about the open source and embedded systems.

Hardware design

Most articles about open source cover areas like a selecting OS, writing the code, compiling, debugging, etc. In the real-life after finishing the specification and analysis phase, we start with designing own hardware or if available purchasing COTS (*Commercial Off-The-Shelf*) hardware.

Primary the design of a PCB (*Printed Circuit Board*) must be done. Available open source programs for drawing PCBs are not very well appropriate for high density and multilayer PCBs. We can often use evaluation boards or COTS hardware to reduce cost and build first prototype. Final PCB can be smoothly modified to support different hardware configurations. The same PCB can be reused in several applications and only the firmware will change.

The PCB must be designed very carefully, any bug (swapped I/O pins, unconnected power supply, wrong package, etc.) results in board redesign. Wrongly chosen componets can dramatically change following software development (e. g. no open source compiler for CPU).

Suggestions

- choose CPU with GCC (*GNU Compiler Collection*) support. There is a wide range of supported CPUs, but some special architectures might not be supported,
- choose components with good availability (long deliver times weeks, months) and publicly available documentation (datasheets, standards),
- don't underestimate the design of PCB, in the most cases you will redesign your boards several times, high density devices ($>$ 100 I/O pins) must be factory-assembled,
- calculate power supply consumption (battery $\times$ wall outlet), active $\times$ passive cooling of components,
- calculate environmental conditions (temperature ranges, vibrations, EMC (*Electromagnetic Compatibility*), mechanical construction)

The resulting PCB must be checked and possible bugs must be fixed (short-circuits, unconnected or misrouted wires, etc.). Digital scope, logic analyzer and simple test/diagnostic programs are necessary to activate the PCB board. Then the developers can start writing the code for CPU's, FPGA's (*Field-Programmable Gate Array*), CPLD's (*Complex Programmable Logic Device*), etc. Don't forget provide them with good documentation – circuit's schemes, connector assignments, I/O descriptions, etc.

Hardware programming

Programable logic forms an integral part of recent hardware design. Hardware description languages (HDL) e. g. VHDL and Verilog are used for formal description of electronic circuits. The hardware developers can describe the circuit's operation, its design, and tests to verify its operation by means of simulation. The simulation is used during design of the logical functions. The synthesis is the process of conversation from HDL into logical structures used in integrated circuits.

Only the vendors [4], [12] know in details the internal structure of the circuits. The third party simulation and synthesis tools are offered for purchase. No equivalent open source tools are available. Vendor university programs offer the IP (*Intellectual Property*) cores for free or low price. On the other hand VHDL and Verilog sources are available [8], [7] under open source licence terms.

Main advantage of using programable hardware is a high performance for simple algorithms. The CPUs are speed limited (program execution) and able to handle tasks with response time in several microseconds. The FPGAs are able to handle tasks in few nanoseconds, with high precision and low jitter. The hadware-software co-design discipline combines the software and hardware parts together to obtain the best performance. Available MCUs (*Microcontroller Unit*) integrate a wide range of hardware components (counters, timers, etc.) which can partially replace the expensive FPGAs.

Software design

Software (firmware) plays the key role in the embedded applications. New features are requested and added continuously. The software becomes the most expensive thing in universe. To succeed in the market the vendors are looking for means to reduce total development cost and to shorten product development cycle. Today the open source is one of the most used buzzword.

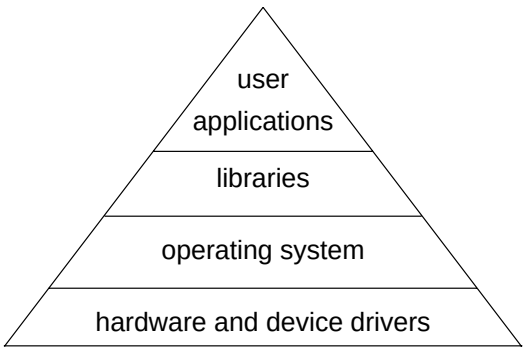


Figure 1 Software layers

The open source has strong position in server and desktop applications. Typical embedded application runs on a single chip microcontroller with few kilobytes of RAM and FLASH memory. In the last half-decade more and more embedded applications become network-enabled. New hardware has enough resources to run full featured OS and to serve several tasks. Both worlds are becoming closer and closer. Figure 3 shows typical structure of software layers.

The hardware architecture determines chances of taking advantage of available open source software. Embedded devices with compatible hardware can be easy modified to run open source. Some basic requirements are:

- GCC availability, cross-compiler for target platform (ARM, AVR, PowerPC, Hitachi SuperH, etc.),
- PPMU (*Paged Memory Management Unit*) for kernel porting (Linux, *BSD). This is not necessary for OS-less systems, kernels or kernel ports for non PMMU systems (uClinux).
- Basic libraries – libc, newlib, uClibc or diet libc.
- Bootloader – grub, lilo, uboot, etc. to start operating system.

Depending on the developers community and how popular your hardware is you will be more or less successfull in the open source deployment. The Linux leads the embedded market [5] and is supported by big vendors. Other operating systems miss the commercial support and the developers community is not large. Be aware of the fact that Linux kernel is not the same on all supported platforms and some features are not available.

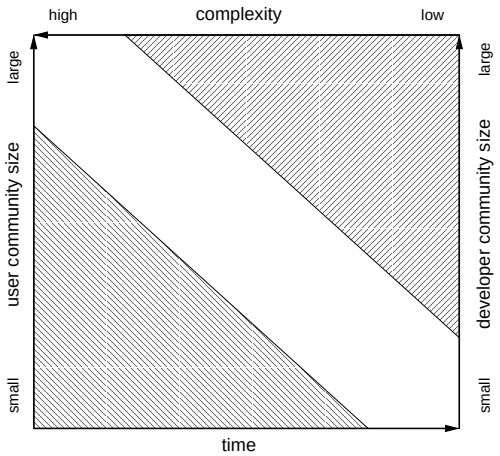


Figure 2 Open source stability [3]

Detailed comparison between Linux and NetBSD approach is described in [6]. Practical advice about using Linux in embedded projects is available here [2]. The growing role of Linux and open source software in the mobile industry is presented here [10].

Nowadays most interesting architectures are ARM for network applications (low price, low power consumption, e. g. switches, access points, etc.) and PowerPC for control applications (long term industry applications). There is a wide range of other open source operating systems and libraries which are focused on 8 and 16-bit CPUs. These processors are more common in low-end devices and enthusiast's constructions.

The embedded applications are often time critical and real-time requirements must be fullfilled. Most of the desktop and server operating systems are GPOS (*General-Purpose Operating System*). There are serious problems due to the misunderstanding the term real-time and RTOS (*Real-Time Operating System*). To add real-time behaviour to GPOS an extra kernel is necessary [9] (hard real-time) or modified kernel with preemption support (soft real-time).

Case study – UAV

We will show real-life example how to build an embedded device using open source as much as possible. The UAV (*Unmanned Aerial Vehicle*) avionics contains width range of control systems. Open source resources can be used for different avionics parts which are often mission critical. Figure 3 shows simplified UAV block schema.

Each avionics's bloc has other requirements and must fulfill specific criteria and standards. The developers of UAV solve problems in the following areas:

- software
  - RTOS – time and mission critical tasks (autopilot, gyroscope, navigation system),
  - GPOS – time non-critical tasks (data logging, visualization),
  - OS-less system – small systems with limited resources (sensors and actors),
- hardware (programmable logic components)
  - FPGA – complex logic for time critical tasks (stepper motor control, radio links),
  - CPLD – simple logic with low power consumption (support units for CPU, etc.).

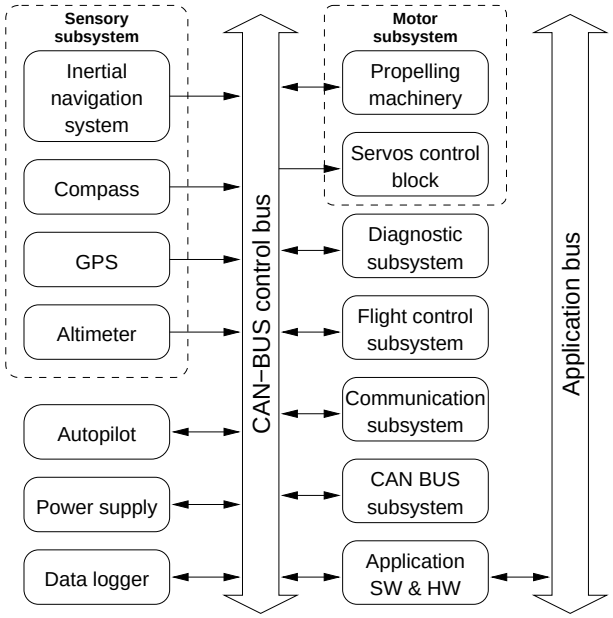


Figure 3 UAV block schema

The final electronic equipment will be used for R&D purposes. We will not discuss the law and the certifications problems necessary for deployment of avionics control systems. Currently the lack of certifications presents the most problematic area during deployment of the open source embedded applications in the industry.

The system will be battery powered with expected mission time about 30 minutes. If possible the COTS components will be used. Small PCBs and some microcontrollers units will be made in-house. The centre of the work will be in the sub-systems integration, design of algorithms and development of software. The software will be written in C, C++ *de facto* standard languages for embedded applications. We are expecting semiautomated flight mode, take off and landing controlled by human operator. The rest of mission will be fully controlled by autopilot.

UAV avionics

The system architecture was made with the focus on dependability which is necessary for air and space applications. The control blocks are interconnected via the CAN-Bus network. CAN-Bus was initially created for automotive purposes

(as a vehicle bus) and it is recently used in many embedded control applications. Each control block contains two independent CAN-Bus interfaces.

Central processing unit based on PC/104 architecture runs Linux OS with RTAI (*Real-Time Application Interface*) extension [9]. PC/104 is intended for specialized embedded computing environments. The Geode and Elan processors used on our PC/104 processor modules are i386 compatible (no cross-compilation is required). Several PC/104 modules can be stacked together and the system functionality can be extended (e. g. CAN-Bus module).

To reduce a final footprint (2–3 MB) the BusyBox [1] and customized kernel are used. System runs from Compact Flash and is remotely controlled over SSH (*Secure Shell*). The application bus (Fast Ethernet) is used to serve high-band datastreams (e. g. video stream).

The other microcontroller units are build as OS-less. They serve special task(s) (e. g. gyroscope, digital compass, GPS) and output the results to the CAN-Bus. Operating system is replaced by newlib library and the hardware adaption layer.

The system is installed in UAV airframe. Radio data link transmits data to a ground station. The UAV is still remotely controlled by human operator.

Figure 4 UAV – central processing unit (PC/104 processor module)

Open and commercial embedded tools

There are two camps of embedded developers. The bigger one are the developers working with commercial tools using MS Windows platform. Most of them have none or low knowledge of UNIX-like operating systems and the development techniques common in the open source community. There are a bit “scared” to get out of their IDE (*Integrated Development Environment*) view of word. The other group are developers interested in UNIX-like operating systems and embedded devices.

Each group prefers tools which are more common for they favorite OS. The problem is that the entire embedded system space is horribly fragmented. There are too many processors (8, 16, 32, ... bits), hardware vendors, integrated circuits, etc. There exists a lot of poor quality products released each few months, with extra new features, fixed some known bugs and introducing new ones. On the other hand they are mature tools developed for long time by recognized vendors or open source projects. The big advantage of commercial tools is that they are covering wide range of non-mainstream HW platforms (compilers, debuggers, ICE (*In-Circuit Emulator*)). The open source tools are offered for free and they are no extra cost in comparison with the commercial tools.

Developing embedded devices is not just writing code. It's necessary to do time analysis, emulate worst case behavior, try to check all possible scenarios. Relability is the one of the most requested features. A reliable toolchain which is easy to learn, use and maintain can save a lot of time and money. The good tools are a right way how to increase the productivity. You can be an experienced developer but with poor tools you will have a poor results.

Conclusion

It is necessary to distinguish between open source code and open source tools. We have mentioned areas where it will be very difficult to find quality open source tools. This is a reason why a lot of companies and embedded developers will never leave the MS Windows platform. Despite it they can release their code under open source licence terms.

The embedded engineers solve problems which are similar to the open source developers. They have good knowledge of C and other programming languages. They must understand the hardware principles and be able to design big things with low resources.

The technological progress brings together the embedded word and open source community. The embedded developers benefit from increasing performance of new devices and rich features of open source operating systems. Today they can easy add a network support or graphical user interface to their new embedded devices.

Most of the open source projects have origins in desktop and server applications. If you are designing similar application in your embedded device you will be happy. In the other case when e. g. hard real-time behavior is required you can get into serious troubles. There is a lot of pitfalls you will need to manage.

References

[1] ANDERSEN, E. *BusyBox – The Swiss Army Knife of Embedded Linux.* [online], last revision 2006–01–11 [cit. 2006–03–14], 2006. http://busybox.net/

[2] BRAKE, C. *Tips for planning an embedded Linux project.* [online], last revision 2006–02–21 [cit. 2006–02–22], 2006. http://www.linuxdevices.com/articles/AT2191860258.html

[3] BRAUN, G. D. *Available source code can work wonders, but watch out.* [online], last revision 2003–09–01 [cit. 2006–03–14], 2003. http://www.isa.org

[4] Altera Corporation. *The Leader in Programmable Logic.* [online], last revision 2006–03–1 [cit. 2006–03–01], 2006. http://www.altera.com/

[5] LinuxDevices.com. *Embedded Linux market snapshot, 2005.* [online], last revision 2005-05-04 [cit. 2006–03–14], 2005. http://www.linuxdevices.com/articles/AT4036830962.html

[6] MICHAELSON, D., BROWNLEE, J. *BSD or Linux: Which Unix is best for embedded applications?* [online], last revision 2003 [cit. 2006–03–14], 2003. http://www.wasabisystems.com/pdfs/Linux_or_BSD.pdf

[7] Opencores.org. *Open-source cores (chip designs) and supplemental platforms (boards).* [online], last revision 2006–03–1 [cit. 2006–03–01], 2006. http://www.opencores.org/

[8] Liberouter project. *Programmable hardware.* [online], last revision 2006–03–1 [cit. 2006–03–01], 2006. http://www.liberouter.org/

[9] RTAI team. *RTAI – Real-Time Application Interface.* [online], last revision 2006–02–11 [cit. 2006–03–14], 2006. http://www.rtai.org

[10] WEINBERG, B. *The new era of mobile Linux ubiquity.* [online], last revision 2006-02-23 [cit. 2006–03–14], 2006. http://www.linuxdevices.com/articles/AT9892857755.html

[11] Wikipedia. *Embedded system.* [online], last revision 2006–02–21 [cit. 2006–02–22], 2006. http://en.wikipedia.org/wiki/Embedded_system

[12] Inc. Xilinx. *The Programmable Logic Company.* [online], last revision 2006–03–1 [cit. 2006–03–01], 2006. http://www.xilinx.com/

Nasazení jednočipových mikropočítačů v praxi

Zbyněk Bureš

E-mail: zbynek.bures@unob.cz

Abstrakt

Příspěvek poukazuje na problémy spojené s nasazením soudobých jednočipových mikropočítačů do reálné praxe. Autor popisuje metody výběru vhodného typu jednočipového mikropočítače, posouzení hardwarových požadavků a možnosti nástrojů pro vývoj aplikací. V další části jsou ukázány prostředky pro zavedení výsledného kódu do cílového mikropočítače, umožňující jeho ladění v cílovém zařízení. Vzhledem k rozmanitosti druhů a typů jednočipových mikropočítačů byly vybrány často nasazované typy od firem Atmel, Microchip, Freescale a Texas Instrument.

Úvod

Dnešní doba je charakteristická vysokým stupněm nasazení vestavných zařízení. Své uplatnění nacházejí v širokém spektru průmyslových a zdravotních odvětvích i jiných oblastech. V případě nalezení mezery na trhu je každý vedoucí projektu postaven před nelehký úkol, jak správně zvolit koncepci, předběhnout konkurenci a maximalizoval vlastní zisk. Při volbě optimální strategie do rozhodovacího procesu vstupují následující parametry:

- předpokládaný zisk,
- předpokládané vstupní náklady,
- míra rizika projektu,
- předpokládaná doba vývoje,
- předpokládané blokování vlastních zdrojů,
- znalost dané problematiky,
- přínos projektu pro další aplikace,

- možnosti outsourcingu,
- používaná součástková základna.

Paradoxně ani současná široká součástková základna rozhodování neulehčuje. Velký výběr různých typů procesorů a podpůrných obvodů v mnoha cenových hladinách nutí konstruktéry udržovat si přehled o nových obvodech a možnostech jejich nasazení. Sledování nových obvodů a jejich dostupnosti na trhu je časově náročné a vyžaduje odborníka se znalostmi z více oborů. Daný odborník musí být schopen posoudit a odhadnout potenciál obvodu pro případnou budoucí aplikaci. Tyto specialisty si mohou často dovolit jen větší firmy. Přesto společnosti, které realizovaly větší počet aplikací, si tuto nezbytnost uvědomují. K hlavní náplni jejich zaměstnanců přibývá i funkce „průzkumníků„ v jednotlivých oblastech součástkové základny. Výstupy pak bývají odborné vnitrofiremní semináře pro ostatní členy z jednotlivých vývojových divizí. Nastíní nové trendy a zajímavé obvody od konkrétních výrobců. Velkým přínosem bývají i informační katalogy distribučních firem. Ty obsahují přehled nových zajímavých obvodů, ceny a jejich dostupnosti.

Metody výběru řídicího obvodu

Jak bylo nastíněno v úvodu, výběr jednočipového mikropočítače pro konkrétní aplikaci je velice obtížný. Špatný výběr může značně zvýšit náklady na vývoj a cenu koncového zařízení. Z těchto důvodů je vhodné před volbou jednočipového mikropočítače zvážit následující kritéria:

a) Počet a typy vnitřních komponent.

   V průběhu analýzy zadání je důležité definovat přesné požadavky na vstupně/výstupní podsystém a specifikovat:

   - počet vstupních a výstupních vývodů,
   - počet vstupních analogových signálů, rychlost a přesnost měření,
   - počet generovaných analogových signálů, rychlost a rozlišení generovaného signálu,
   - počet a typy komunikačních rozhraních (RS232, RS485, CAN, Ethernet),
   - maximální povolený příkon a další.

   V případě nedostatku vstupně/výstupních vývodů na čipu lze jejich počet rozšířit doplňujícím HW. Tato rozšíření přináší řadu komplikací. Proto je nezbytné zodpovědně posoudit požadavky aplikace a v případě nejistého a měnícího se zadaní použít mikropočítač s větším počtem vstupně/výstupních vývodů a periferií.

b) Cena procesoru a vývojových prostředků.

Podstatným problém je cena procesoru a vývojových nástrojů nutných pro návrh programového vybavení. Při výběru vhodného procesoru je nezbytné kalkulovat s počtem kusů, s cenou vývojových nástrojů, s finální cenou výrobku a s potenciálním využitím v navazujících aplikacích. Cenu vývojových prostředků lze snížit použitím open-source programového vybavení. Open source nástroje pokrývají nejrozšířenější typy mikropočítačů.

c) Posouzení z hlediska dostupnosti.

Firmy plánující výrobu nového produktu v delším časovém horizontu, často akceptují pouze obvody, jež jsou vyráběny více nezávislými výrobci. Důvodem je možnost jejich nahrazení v případě, že primární výrobce obvodů má výpadek nebo nečekaně ukončil výrobu. Nezanedbatelnou roli hraje i dostupnost obvodu na trhu a minimální balicí jednotka.

d) Další možnosti rozšíření.

V průběhu řešení projektu se může nastat situace kdy je nutno navýšit počet HW zdrojů, které nebyly při zadání známy. Často se jedná o velikost paměti dat nebo kódu. Z tohoto důvodu je vhodné projekt řešit na procesoru výkonnějším a následně vybrat vhodného kandidáta z dané procesorové rodiny. Obdobný přístup se například využívá při realizaci projektů s programovatelnými strukturami.

Možnosti nástrojů pro vývoj aplikace

Při řešení projektu je nezbytné využívat celou řadu programového vybavení. Je nezbytné si uvědomit, že mezi nástroje nepatří pouze překladače a kompilátory pro daný mikropočítač. Nástroje je třeba rozdělit do několika kategorií:

1. nástroje pro plánování a řízení projektu,
2. nástroje pro HW úroveň (návrh desek plošných spojů, nástroje pro simulaci návrhu číslicových a analogových obvodů),
3. nástroje pro SW úroveň (editory zdrojových kódů, validátory kódu, překladače a kompilátory),
4. nástroje pro ověření funkčnosti SW a HW (programové simulátory, hardwarové emulátory),
5. podpůrné nástroje pro ověření činnosti a diagnostiku (osciloskopy, logické analyzátory, funkční generátory).

Výše uvedené kategorie jsou seřazeny v pořadí v jakém jsou uplatňovány v průběhu vývoje. Pro programátory je často problematika hardwarového návrhu skryta. Vychází z informací o:

- typu použitého procesoru,
- přiřazení vstupně/výstupních vývodů,
- požadavků na řízení vstupně/výstupních periferiích na čipu,
- požadavků na řízení vstupně/výstupních obvodů připojených skrze tyto vývody mikropočítače.

Pro zjednodušení desky plošného spoje složitých návrhů se využívají programovatelné struktury (FPGA). FPGA obvody umožňují provádět programovou rekonfiguraci vývodů, ulehčují procesoru od rychlostně nebo výkonově náročných operací. Na druhou stranu použití FPGA prodražuje cenu návrhu, vyžaduje vývojáře FPGA a klade vyšší nároky na zkušenosti návrhářů desek plošných spojů. U menších firem je častým zvykem, že návrhář hardwaru, desky plošného spoje i programátor je jedna a táž osoba. Tento přístup přináší řadu výhod i nevýhod. Pozitivem je komplexnější zhodnocení všech požadavků od jednotlivých periferních obvodů, vstupů/výstupů a posouzení vazeb a návazností hardwaru na software. Negativem je pak špatná zastupitelnost jednotlivých návrhářů, neboť obecně platí, že každý má svůj zažitý styl práce, návrhu i psaní zdrojových kódů.

U projektů vyznačujících se vysokou rychlostí nebo objemem zpracovávaných dat, je často nezbytné využívat programové vybavení pro optimalizaci návrhu desky plošného spoje z hlediska rychlosti šíření signálů v čase. Tyto programy jsou schopny v závislosti na použitém materiálu desky plošného spoje, rozmístění součástek a topologii sběrnic vypočítat kapacity vodičů, pravděpodobnosti přeslechů na sběrnicích a potenciální místa problémů s odrazy a nepřizpůsobením na vedení. Zejména u návrhů s vysoce rychlými sériovými kanály v řádech GHz jsou podobné nástroje pro simulaci a optimalizaci nezbytné. Přes veškerou snahu ze strany SW o ulehčení práce návrhářů, je část návrhu a optimalizace desek plošných spojů stále ponechána na návrhářích desek plošných spojů.

Dalším krokem při realizaci projektu je vytvoření zdrojových kódů pro použité programovatelné struktury. FPGA jsou v současnosti převážně programovány v jazyce VHDL nebo Verilog. U starší generace návrhářů je preferován kombinovaný návrh vyššího jazyka se schématem. Velkým usnadněním je možnost aplikace „IP jader“ (Intelectual Property) [1, 2]. Jedná se o hotová a ověřená jádra periferií, procesorů, řadičů a jiných obvodů, která jsou optimalizována pro danou rodinu obvodů FPGA. Každý výrobce FPGA pak poskytuje svá vlastní IP jádra nebo jádra od „třetích výrobců“. Jejich nákupem uživatel získá důležité komponenty pro návrh, aniž by musel podrobně studovat problematiku

například PCI, řadičů paměti a dalších. Kromě této možnosti lze také využít open-source projekty [3].

Nedílnou součástí návrhu FPGA je simulace a časová optimalizace. Výrobci obvodů poskytují stále kvalitnější a propracovanější produkty pro analýzu. Rychlé návrhy a zejména návrhy s vysokým stupněm zaplnění FPGA však stále vyžadují kvalifikované návrháře a dlouhou dobu vývoje.

Dalším krokem po návrhu desky plošného spoje a naprogramování obvodů FPGA je realizace programového vybavení pro řídicí mikropočítač. Vývoj programového vybavení často probíhá paralelně s vývojem desky plošného spoje a programováním FPGA.

Procesory obecně mohou být programovány v jazyce symbolických instrukcí (assembler) nebo vyšších programovacích jazycích (C, C++, Ada, Pascal a dalších). Zejména u malých jednočipových mikropočítačů s omezenou pamětí RAM nebo DSP procesorů stále převládá programování dílčích částí nebo celého kódu v assembleru. Paradoxně ani špatná přenositelnost kódu nedonutila programátory přestoupit k vyšším jazykům. Často je tato „nechuť“ způsobena velkým množstvím zdrojového kódu v assembleru z dřívějších projektů, nedostatek lidského a finančního potenciálu.

Stále častěji se ale setkáváme s projekty, kde aplikace vyžaduje operační systém a vyšší programovací jazyky jsou nezbytné. V současnosti je možné využít několika výrobců operačních systémů pro embedded systémy, jejichž produkty jsou certifikovány pro průmyslové nebo letecké nasazení. Cena těchto operačních systémů je ale vysoká a pro běžné firmy je jejich nasazení nemyslitelné. Stejně jako u FPGA i zde je možné využít open-source operační systémy.

Pokud je vývoj postaven na platformě x86, je možné části kódu ověřit na běžném počítači. V opačném případě je nezbytné využít hardwarových prostředků emulátorů. Většina mikropočítačů pro vestavné aplikace jsou vybaveny rozhraním JTAG, umožňujícím provádět diagnostiku, řízení běhu, činnost mikropočítače a ovlivňovat řídicí registry periferií. Pro tyto mikropočítače výrobci dodávají vývojová prostředí nebo nabízí produkty třetích firem.

Prostředky pro zavedení kódu do cílového zařízení

Zdrojový kód je třeba přenést a uložit do cílové paměti. Záleží na typu mikropočítače/procesoru zda je jeho programový kód uložen ve vnitřní paměti FLASH nebo v externí paměti na desce plošného spoje. Externí paměti bývají programovány ve specializovaných programátorech pamětí. V průběhu programování dochází k okamžité verifikaci zapsaných dat, je možné často programovat více

obvodů současně a doba programování u paralelních pamětí je kratší než u sériového programování. Nevýhodou tohoto řešení je manuální manipulace se součástkami citlivými na statickou elektřinu a komplikace při změně programového vybavení. Při programování „na desce“ je procesor nebo mikropočítač programován přes sériové rozhraní. V praxi je nejčastěji využíváno programování přes JTAG rozhraní nebo vestavěný asynchronní sériový zavaděč. Oba způsoby programování ulehčují sériovou výrobu, snižují dobu manipulace s obvody citlivými na elektrostatické pole a poskytují možnost snadné změny obsahu vnitřní paměti i za běhu systému. Existují i systémy s kombinovaným přístupem, kdy prvotní obsah paměti včetně zavaděče nové verze kódu je naprogramován externě. Následně část paměti obsahující zavaděč je uzamknuta proti dalšímu zápisu a poté je paměť osazena na desku plošného spoje. V případě potřeby změny programu je spuštěn zavaděč v chráněné oblasti. Kód zavaděče pak následně umožní přepis nechráněné paměti FLASH. Při vhodném rozdělení paměti a drobné programové úpravě je možné provádět za běhu přepis programu. Tohoto postupu se využívá u aplikací, kde není možný fyzický zásah obsluhy (vzdálená průmyslová zařízení, serverové switche, a jiné).

Řešené projekty

V následujícím textu autor popisuje jím řešené projekty. V prvním případě se jedná o zařízení pro řízení spalovacího motoru. Zde řešitelé špatně provedli analýzu zadání a špatně odhadly dobu a náklady nezbytné na jeho řešení. V druhém případě je popsáno zařízení pro měření teplot a otáček ve zkušebně motorů. Řešitelům se již podařilo lépe odhadnout dobu řešení a náklady s tím spojené.

Zařízení pro řízení spalovacích motorů

Zadáním byla realizace zařízení určeného k řízení dvoutaktních spalovacích benzinových motorů. Zařízení má provádět řízení otáček v rozsahu 1 000 až 8 000 otáček za minutu. Umožnit výběr z vestavných křivek předstihu a programování vlastní křivky předstihu. Zařízení mělo provádět řízení škrticí klapky sytiče a tím zjednodušit startování motoru v nestandardních polohách.

V tabulce 1 jsou uvedeny vstupní parametry vývoje.

Tabulka 1 Vstupní charakteristiky projektu, optimistický odhad

Vstupní znalosti problematiky:	problematika motorů – výtečná problematika jednočipových mikropočítačů – značně pokročilá
Velikost týmu:	návrhář motorů, vývojář HW a SW.
Předpokládaná doba vývoje:	6–9 měsíců
Předpokládané náklady na vývoj:	elektronika – 100 000 Kč mechanika – 250 000 Kč
Stav zadání projektu:	rámcové zadání, znalost cílů
Předpokládaná cena produktu:	neznámá
Předpokládaná roční produkce:	cca 3 000 kusů

V tabulce 2 jsou uvedeny reálné hodnoty doby vývoje a nákladů.

Tabulka 2 Výstupní charakteristika projektu

Doba vývoje:	25 měsíců
Doba přerušení vývoje:	4× (průměrně po dobu jednoho měsíce)
Korigovaná délka vývoje HW a SW:	21 měsíců
Počet verzí desek plošného spoje:	6
Náklady na vývoj:	elektronika – 200 000 Kč, mechanika – 1 200 000 Kč, mzdy – 0 Kč
Stav vývoje projektu:	stále otevřený (v současnosti tři vyráběné modifikace)
Potvrzená roční produkce:	3 500 kusů
Stav dokumentace:	neuspokojivý
Verze SW:	verze 3 (třetí modifikovaný algoritmus)

Z výše uvedeného shrnutí je možné si udělat představu, jak byla podceněna analýza a definice zadání. V průběhu realizace byl kritickým prvkem mikroprocesor. Během projektu byly vytvořeny prototypy s procesory MC56F801 [4] a MC56F8323 [5] firmy Freescale. Jejich vlastnosti převyšovaly potřeby specifikované v zadání a umožňovaly další četná rozšíření. Vlastní cena mikropočítače však řešitele donutila v polovině projektu hledat vhodnou náhradu. Poměr ceny procesoru vůči ceně finálního zařízení byl nevyhovující. Dalším řídicím mikropočítačem byl ATmega64 [6] firmy Atmel. V důsledku změny zadání byl procesor zamítnut a projekt dokončen na mikropočítači MSP430F133 [7] firmy Texas Instruments.

Byla realizována životnostní zkouška a vyrobeno prvních 600 kusů. V současnosti začíná nabíhat sériová výroba. Další vývoj produktu je pozastaven do doby, kdy bude uzavřena dokumentace. Zařízení je vyráběno ve třech verzích. Uzavření dokumentace je plánováno na červen 2006. V tu dobu se předpokládá již plně samostatná sériová výroba.

Závěr a zhodnocení

Délka vývoje byla ovlivněna následujícími okolnostmi:

1. byla špatně provedena analýza problému,
2. nebylo přesné a pevné zadání projektu,
3. nebyl vytvořen a dodržován harmonogram práce,
4. změnila se požadovaná cena výsledného produktu a tím i maximální cena procesoru,
5. byly aplikovány čtyři typy procesorů od tří výrobců,
6. nebyly dobře odhadnuty ceny forem a přípravků pro výrobu,
7. spoluautor neměl zkušenosti se zajištěním sériové výroby,
8. v průběhu vývoje bylo nezbytné čtyřikrát přerušit práci na projektu a soustředit se na jiný „důležitější„ projekt,
9. byly významně podceněny lidské zdroje.

Za největší nedostatek celého projektu lze považovat špatné řízení. Dobu řešení ovlivnilo podcenění lidských možností a nutnost řešit více projektů současně. Nabyté zkušenosti byly využity v následných pěti projektech.

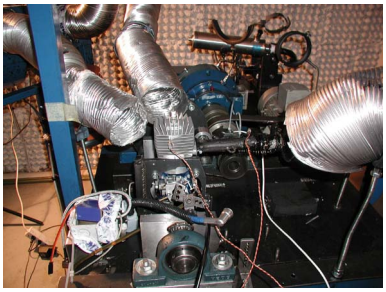
Zařízení pro měření teplot na motorové brzdě

Úkolem byla realizace měřicího systému pro potřeby vývoje spalovacích motorů. Zařízení mělo umožnit měření, záznam a vizualizaci teplot v závislosti na otáčkách motoru. Výstupní soubor měl obsahovat informace o atmosférických podmínkách ve zkušebně v průběhu celého měření. Obslužné programové vybavení mělo být platformově nezávislé.

Při realizaci byla velkou výhodou dělba práce projektu na dva programátory. Nabyté zkušenosti z předchozího projektu společně s dobrou znalostí vývojového prostředí cílového mikropočítače zkrátily dobu vývoje. V tabulce 4 jsou uvedeny konečné parametry projektu.

Tabulka 3 Vstupní charakteristika projektu

Vstupní znalosti problematiky:	problematika motorů – výtečná problematika jednočipových mikropočítačů – značně pokročilá problematika programování aplikací pod Linuxem – výtečná
Velikost týmu:	motorář, návrhář HW a SW pro jednočipový mikropočítač, programátor aplikace na PC
Předpokládaná doba vývoje:	6–9 měsíců
Předpokládané náklady na vývoj:	elektronika – 10 000 Kč, mechanika – 10 000 Kč, el. komponenty od třetích firem – 20 000 Kč, mzdy – 20 000 Kč
Stav zadání projektu:	rámcové zadání
Předpokládaná cena:	neznámá
Předpokládaná roční produkce:	1–2 kusy



Tabulka 4 Výstupní charakteristika projektu

Doba vývoje:	3,5 měsíce
Doba přerušení vývoje:	1× (průměrně cca 0,5 měsíc)
Korigovaná délka vývoje HW a SW	3 měsíce
Počet desek plošných spojů:	1 verze + drátové modifikace
Náklady na vývoj:	elektronika – 3 500 Kč mechanika – 5 500 Kč, el. komponenty od třetích firem – 18 500 Kč, mzdy – 20 000 Kč
Stav vývoje projektu:	otevřený (v současnosti na zkouškách)
Potvrzená roční produkce:	1–3 kusy
Stav dokumentace:	uspokojivý
Verze SW:	druhá

Závěr a hodnocení

Popisovaný projekt byl lépe odhadnut. Podařilo se redukovat náklady na HW i mechaniku a dodržet stanovenou dobu vývoje. Jak bylo zmíněno dříve, části práce se opíraly o zkušenosti a výstupy předchozích projektů (řešených v průběhu projektu zařízení pro řízení spalovacích motorů). Nevyrábělo se více verzí desek plošného spoje. Poupravila se pouze první verze. Rozdělení práce programátorů umožnilo rychleji odhalit některé skryté chyby v průběhu vývoje.

Závěr

Je velice obtížné postihnout všechna rizika a úskalí spojena s vývojem jednočipových aplikací. Nelze jednoznačně specifikovat kritéria, dle kterých je možné se řídit. Jiná situace bude u firmy skládající se z jednoho až dvou zaměstnanců na poloviční úvazek. Jiná bude ve firmě, kde pouze sekce plánování a řízení projektu má 10–30 lidí. Obecně lze ale říci, že analýza projektu, podrobná specifikace zadání a „pochopení zadání“ je klíčem k úspěchu a minimalizaci nákladů na vývoj.

Literatura

[1] Altera. *Intellectual Property.* [online].
http://www.altera.com/products/ip/ipm-index.html

[2] Xilinx. *Intellectual Property.* [online].
http://www.xilinx.com/ipcenter/index.htm

[3] Open-cores. *Projects.* [online].
http://www.opencores.com/browse.cgi/by_category

[4] Freescale. *MC56F801.* [online].
http://www.freescale.com/files/dsp/doc/data_sheet/MC56F801.pdf

[5] Freescale. *MC56F8323.* [online].
http://www.freescale.com/files/dsp/doc/data_sheet/MC56F8323.pdf

[6] Atmel. *ATmega 64.* [online].
http://www.atmel.com/dyn/resources/prod_documents/2490S.pdf

[7] Texas Instruments. *MSP430F133 16-Bit Ultra-Low-Power Microcontroller* [online]. http://focus.ti.com/docs/prod/folders/print/msp430f133.html

Formal Views on Rapid Application Development with Processor Expert

Petr Stružka, Dušan Kolář

E-mail: PStruzka@unis.cz, DKolar@unis.cz

Abstract

Processor Expert™ (PE) is an advanced, component oriented, open Rapid Application Development (RAD) environment for embedded systems. The paper describes basic formal views on RAD using PE components – Embedded Beans™. They are ready-to-use building blocks for easy application creation with intuitive graphical user interface. They provide hardware abstraction layer (HAL) for thousands of MCUs widely used in embedded systems and automotive industry.

Based on the Processor Expert technology, there are tools supporting formalisms in software development. In Processor Expert context, we can introduce Processor Expert blockset for Matlab Simulink® that allows source code generation for embedded systems from Simulink® diagrams by Real-Time Workshop® and the code generation based on state-diagrams using State Builder™ for Processor Expert.

Keywords: Formal description, Processor Expert, Embedded Beans, Rapid application development, Model based development, Processor Expert Blockset, State Builder.

1 Introduction

The growing industry in the field of embedded systems stands for higher requirements for the software developing tools for those systems. One of the requirements is a system development time-to-market that should be as short as possible. Another important requirement is to define and to fulfill the standards. This leads in the necessity of hardware independency of the application. Such an independence can be accomplished when a HAL (Hardware Abstraction Layer) is used in the application development. The HAL is strongly suitable for using in the model based application development. The need of standards means need of formal description of the algorithm.

2 Hardware Abstraction Layer

The hardware abstraction layer provides an interface between hardware and software. It provides a consistent platform that should be used either for running an embedded application or for the creation of other (higher) layers of software encapsulation interfaced to the application. When a HAL is incorporated, applications do not access hardware directly, but they access an abstract layer provided by the HAL. The resulting application becomes a device-independent one and the platform-specific dependencies are hidden for it.

The following figure shows a possible general structure of an application when using a HAL. Each peripheral driver (low-level layer) is encapsulating one or several peripherals of MCU (Microcontroller device).

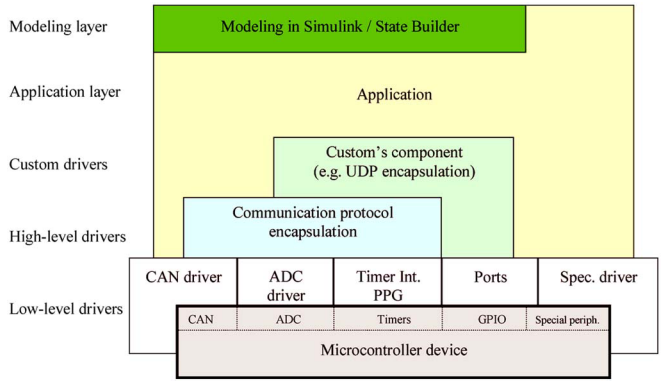


Figure 1 Application scheme with HAL

The application is using the API (Application Programming Interface) provided by such a driver. As shown on the figure, additional high-level drivers can use the low-level API and provide such an additional API (independent) on hardware platform that can be used in the application. The HAL can be implemented several ways, UNIS provides the following HAL implementations: Embedded beans (they provide a generic interface for an application creation; they are implemented over multiple architectures from several manufacturers), HIS and AutoSAR SPAL (implementation of SPAL – Standardized Peripheral Abstraction Layer drivers of the AutoSAR standardization).

Using HAL allows users to create an application code that is easily portable over several platforms. Nevertheless, the application logic need not be definitely the top layer of the HAL hierarchy. The modeling techniques that are able to generate application logic can be used on the top of the HAL hierarchy.

2.1 Component Technology in Embedded Systems

The component technology fulfils the requirement to have a specific functionality separated into small objects usable in the whole system. A component can be comprehended as an object with the defined interface (like a class in the object-oriented programming). The component's interface consists of methods (procedures that can be executed), events (indication that something important has changed – state of component changed) and a set of properties that can modify/customize the object behavior.

It is very important to make a good connection of components and a HAL. The goal is to organize and unify a HAL on several kinds of MCUs (even for different architectures) to achieve the ability to configure peripherals. The main emphasis is on the user-friendly interface (best graphical user interface). The configuration should allow setting peripheral parameters, such as a connection to hardware resources (pin assignment), timing requirements, mode selectors, behavior modifiers etc.

From the embedded systems programmer's point of view the component should be understood as an object that provides its methods (function calls), events (indication that something important has changed on peripheral, usually mapping of ISRs (Interrupt Service Routine) and set of properties, that can modify/customize the peripheral behavior.

In the terms of HAL, the component is a basic encapsulation of the defined MCU functionality and it provides a unique API that can be used in applications independently on the target platform.

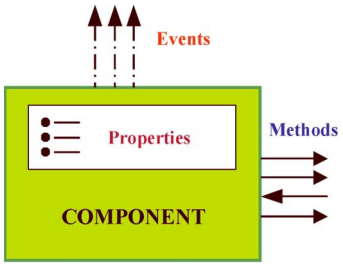


Figure 2 Embedded Bean interface

Such a connection is enabled by the Embedded Beans™ – components of Processor Expert. They are basic building blocks for an application creation. Embedded Beans™ provide unified interface and functionality across the target platforms. They encapsulate certain features of MCU (several Embedded Beans™ can encapsulate several modes of one peripheral) through their interface – set of properties, methods and events.

Thanks to the Embedded Beans™ phenomenon, they are suitable not only as a HAL implementation of API, but also as application components suitable for rapid application development.

3 Rapid Application Development

The fact that we can use the Embedded Beans™ – the components implemented on a large scale of MCUs – allows us to use them effectively for creation of the wide variety of applications or for application prototyping. Thus, we are talking about rapid application development.

There are the following reasons for using the RAD techniques:

- Application creation speed-up

  Application can be composed from already created components that are able to control their behavior through a set of properties.

- Portability

  These components can have their implementation for several (different) kinds of MCUs. Nevertheless, the interface to the application is always the same (principle of HAL).

- Standardization

  The interface of components can be designed to fulfill standardization requirements. Several sets of components with a different interface can be implemented on one architecture (e. g. UNIS's Embedded Beans™ vs. HIS or AutoSAR SPAL).

- Code re-usability

  Encapsulation of custom's code in new components allows others to use such a code in application and effectively re-use what is already done.

3.1 Model based application design

A technique representing RAD is model based application design. From the formal point of view, the system can be described in a graphical form. This paper shows two different principles. The first is Simulink® model that allows describing the model as a block that provides transition function between input(s) and output(s). Of course, the block could be assembled from several smaller sub-blocks that form the whole system. This is suitable for process control systems. Another technique is the description of the algorithm via state charts. This allows tool State Builder for Processor Expert that generates the source code for the state machine implementation.

3.2 Simulink

The Mathworks® Simulink® with Real-Time Workshop® allows creating a control system model and then verifying it using a simulation. Thanks to the Real-time Workshop® (RTW) it is possible to generate the algorithmic logic as a source code in C language. If we do not consider some specific RTW targets, it generally means that the generated code can be used for specific target embedded systems. Nevertheless, we have to add libraries, or selected MCU specific code (mostly peripheral drivers) to make the whole system working properly.

Processor Expert provides a HAL interface between an embedded application and platform specific behavior of MCU. The integration of Simulink® and Processor Expert system gives the possibility of using the generated driver's code in the resulting code generated by RTW. This is achieved through the Processor Expert blockset used inside Simulink®. Blocks consist of basic Processor Expert components and they provide a simulation features too. Thanks to the connection of blocks to Processor Expert, it is possible not only to set up the parameters needed for selected peripheral configuration, but even to set up parameters for simulation in Simulink® as well.

Composition of resulting code ensure Processor Expert Embedded Real-Time Target (PEERT), which in addition to code generated for algorithm, provides also connection of inputs and outputs of each block with proper function of MCU's peripheral.

The following steps of model based application design with Simulink® and Processor Expert demonstrate usage of both tools together:

1. Design of the control system using Simulink®
   - Standard Simulink® blocks used
   - Processor Expert blockset used
2. Verification of the model using simulation
3. Configuration of used peripherals by Processor Expert
4. Code generation
   - MCU initialization (PE)
   - MCU peripheral drivers (PE)
   - Control algorithm (RTW – PEERT)
5. Building and verifying the output on a real hardware

The main advantage of this solution is generality of the target (thanks to Processor Expert HAL) and then the possibility of using the model on several

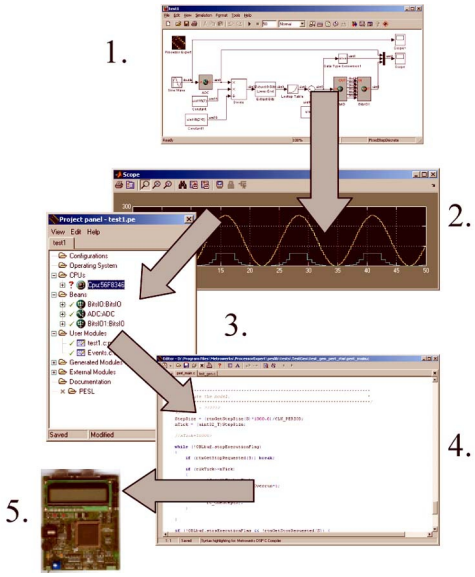


Figure 3 Model based application design

platforms. The assessment of this approach is the generality of the simulation that is not able (at this moment) provide certain simulation according to the selected target CPU.

3.3 State Builder

State Builder for Processor Expert is an easy-to-use solution focused on use of state machines. It allows formally describe application by state charts (deterministic finite state machines) and than generate the source code from them. The graphical notion and behavior semantics comes from the UML, a part of which state charts are.

Simply saying the main idea of application design in the State Builder is the following: State Builder allows a user to describe application logic by a simple diagrams called State Charts. These diagrams are then used as an input for generation of source code, which is exported into Processor Expert. State Builder tightly cooperates with Processor Experts and uses its APIs in generated source code so that it is applicable to any hardware platform supported by Processor Expert.

The following steps are typical steps of application creation in the State Builder:

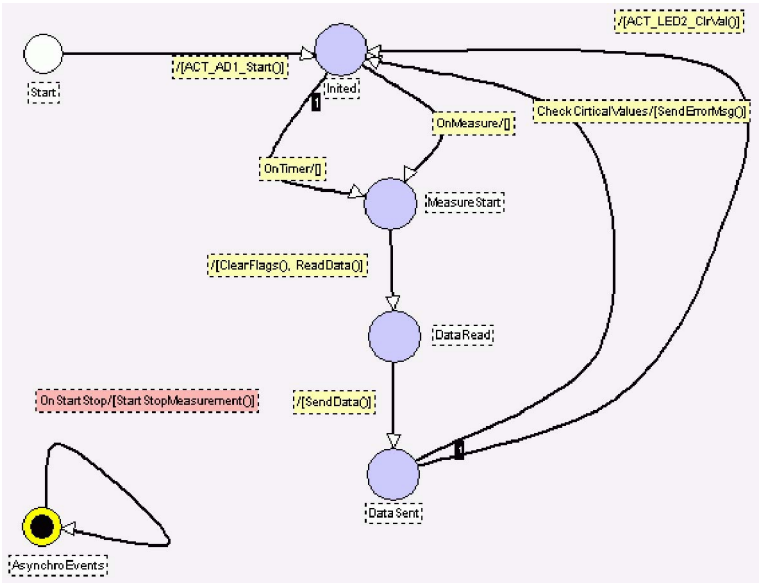


Figure 4 State chart

- Create a new project (standard Processor Expert items like CPU and Embedded Beans can be added)
- Open the StateBuilder and design a state chart

  When the state chart is created, the user defines actions and conditions for transitions between states. The implementation of actions and conditions can be either written by the user (custom code), or Processor Expert code can be utilized. The code can be represented by methods and conditions can be represented by events of beans in the Processor Expert project (method are used in a role of actions performed during transition firing)

- Validate state chart

  During the validation process, it is checked whether the state chart structure is correct and checks for these common errors:

  – Unconnected states

  – Too many/not enough initial states – every automaton must have at least one initial state; a topmost automatons can hold only one initial state.

  - Too many joined asynchronous states – only one asynchronous state can be joined to another non-asynchronous state.
  - Wrong defined transition paths
  - Priorities' conflicts in transition paths
  - Duplicated events in transition paths

- Code generation. Two algorithms can be used
  - Goto algorithm
  - Loop-Case algorithm

  The Goto algorithm uses goto commands and labels for the jumps in contrast to the Loop Case algorithm where a switch-case structure (see Figure 5) is used for differentiation between given program states.

```
void Control(void)
{
  /* State: Start */
  /*** SB code: begin of 'SwitchOFF' action code. ***/
  Bit1_ClrVal();
  /*** SB code: end of 'SwitchOFF' action code. ***/
  /* State: Control */
  State_ID_Control:
  {
    /*** SB code: begin of 'Algorithm' action code. ***/
    if(AD1_Measure(1)<SET_POINT)Bit1_SetVal();
    else
        Bit1_ClrVal();
    /*** SB code: end of 'Algorithm' action code. ***/
    goto State_ID_Control;
  }
}
```

Figure 5 Example of Goto algorithm generated code

These two different programming approaches lead to different properties of generated source code. The Goto algorithm produces more efficient and smaller program code that can lead to the more optimized code than a program code produced by the Loop-Case algorithm. On the other hand, the source code produced by the Loop-Case algorithm (see Figure 6) is much more transparent and human-readable than output of the Goto algorithm.

- Optimization

  The optimization process performs various optimization algorithms on validated structure of the user-created state chart. The goal is to minimize and simplify the automaton's structure so that the generated source code based on it will be as efficient as possible. Optimizations performed by the

```
void Control(void)
{
  TYPE_STATE state=ID_Start;
  for(;;)
  {
    /* Main loop */
    switch( state )
    {
      /* State: Start */
      case ID_Start:
        /*** SB code: begin of 'SwitchOFF' action code. ***/
        Bit1_ClrVal();
        /*** SB code: end of 'SwitchOFF' action code. ***/
        state=ID_Control;
      /* State: Control */
      case ID_Control:
        /*** SB code: begin of 'Algorithm' action code.***/
        if(AD1_Measure(1)<SET_POINT)Bit1_SetVal();
        else
            Bit1_ClrVal();
        /*** SB code: end of 'Algorithm' action code. ***/
        state=ID_Control;
        break;
    }
  }
}
```

Figure 6 Example of Loop-case algorithm generated code

optimizer do not affect a structure or layout of a source state chart created by user; it creates a new optimized state chart with the same functionality as the user's one.

- Code synchronization

  When all the code is generated and the user makes any change in the generated code during debugging, it is possible to expose such a change in the source code back to the state chart definition. Of course, only condition or action code can be modified, not the state machine structure. The code is labeled with special marks (as comments) that guide the user in source code.

Such a development cycle is enabled due to strong orientation of the State Builder on formal base – finite state automata.

Definition 1: *A deterministic finite automaton is a 5-tuple* $(\Sigma, Q, q_0, \delta, F)$*, where:*

- Σ *is a finite nonempty set representing the input alphabet.*
- Q *is a finite nonempty set of states,* $Q \cap \Sigma = \emptyset$*.*
- q_0 *is an initial state, it is an element of* Q*.*

- *δ is the state transition function: $\delta: Q \times \Sigma \to Q$.*
- *F is the set of final states, a (possibly empty) subset of Q.*

Every state chart built in the State Builder can be mapped onto a deterministic finite automaton, where input alphabet is composed from conditions/events of the state charts and actions are omitted in such an automaton. Such a mapping allows more thorough and deep investigation of the state chart properties on a formal basis. Thus, we can detect various features of the state chart exploiting vast resources of already described and developed techniques for finite automata.

To be even more precise, every state chart designed in the State Builder is a Mealy automaton, which can be formally defined this way:

Definition 2: *Mealy automaton is a 6-tuple $(\Sigma, \Lambda, Q, q_0, \delta, \omega)$, where:*

- *Σ is a finite nonempty set representing the input alphabet.*
- *Λ is a finite nonempty set representing the output alphabet.*
- *Q is a finite nonempty set of states; Q, Σ, and Λ are pair-wise disjoint.*
- *q_0 is an initial state, it is an element of Q.*
- *δ is the state transition function: $\delta: Q \times \Sigma \to Q$.*
- *ω is the output function: $\omega: Q \times \Sigma \to \Lambda$.*

In the case of Mealy automata, the mapping of state chart to the formal base includes even actions that are mapped onto output symbols.

The State Builder for Processor Expert advantage is its simplicity of use. It is a Processor Expert plug-in and it does not require any additional components (like for example the Simulink® solution). Moreover, it is based on formal notions, which promises further development (to, for instance, nested automata) and correct code generation/optimization and verification. Many applications can be developed with a state chart without necessity of additional input of a user to the generated code.

4 Conclusion

Processor Expert™ hardware abstraction layers and the Embedded Beans™ have the pivotal value in rapid application development. Embedded Beans™ provide unique interface for the application creation, suitable for modeling.

The model based application development brings new possibilities of rapid application development with possibility of verification and validation of the

system. Out of doubt, this approach has main contribution on field of standardization and formal algorithm description. Moreover, such an approach leads to speed-up of application development while preserving or even increasing quality of the production code. As embedded systems become ubiquitous and the QoS they deliver plus their natural or required safety together with strong need of instant update of such systems to meet actual requirements, usage of formal approaches to application development seems to be the good way to achieve such goals.

References

[1] THE MATHWORKS: 2004. *Simulink – Simulation and Model-Based Design.* USA, The Mathworks, 2004.

[2] BARTOSINSKI, R., STRUŽKA, P., WASZNIOWSKI, L. *PEERT – Blockset for Processor Expert and MATLAB®/Simulink® integration.* Technical Computing Prague, Humusoft, 2005. ISBN 80-7080-57-3.

Grant (in Czech language)

SESAP
Participace na vývoji evropské standardizace vestavného SW pro automobilový průmysl. Poskytovatel: AV ČR, v programu: Informační společnost, kód projektu: 1ET400750406

Internetové vysílání stanic Českého rozhlasu ve velmi vysoké kvalitě

Miloš Wimmer

E-mail: wimmer@civ.zcu.cz

Abstrakt

Příspěvek je věnován popisu systému, který používáme k živému vysílání programů stanic Českého rozhlasu do sítě Internet. Řešení je založeno na technologii vysílání audio streamů v kompresním formátu Ogg Vorbis v rychlostech 128 kb/s a 224 kb/s. Na programové úrovni používáme pouze svobodný software – operační systém GNU/Linux, producenty streamů vls, vlc, oggenc a ices a streamovací server Icecast.

V roce 2003 byla mezi sdružením CESNET a Českým rozhlasem uzavřena dohoda o spolupráci při vytvoření experimentálního systému, který umožní vysílání živého programu stanic ČRo do Internetu ve vysoké kvalitě. Do té doby vysílal Český rozhlas do Internetu živý program svých významných stanic pomocí technologií Real Audio a Windows Media v rychlostech 10–32 kb/s. Tímto způsobem však dosahoval jen nízké kvality přenášeného zvuku.

V rámci výzkumného záměru sdružení CESNET jsme navrhli a realizovali systém, který nepřetržité živé vysílání do Internetu ve vysoké kvalitě umožňuje. Po experimentálním provozu byl systém převeden do produkčního prostředí a tam běží. Protože systém průběžně vylepšujeme, budu se v dalším textu věnovat popisu posledního stavu.

Základními požadavky na celý systém bylo:

- dosažení co možná nejvyšší kvality přenášeného zvuku,
- dosažení co možná nejvyšší stability a odolnosti proti výpadkům,
- možnost zpracovávat vstupní signál z různých zdrojů,
- přijatelná dostupnost a podpora ze strany klientů.

Dosažení těchto požadavků značně předurčilo technologii, na kterou jsme se orientovali. Systém jsme založili na výhradně svobodném programovém vybavení, jehož základ tvoří svobodný kompresní formát Ogg Vorbis, producenti ogg streamů a streamovací server Icecast.

Obecné pojmy

Komprese a datový tok

Pro snížení nároků na objem přenášených audio dat se používají kompresní kodeky (formáty). Komprese je buď bezztrátová, při níž jsou všechna digitalizovaná audio data zachována, anebo ztrátová, při které dochází k částečné ztrátě informace, ale výsledný zvukový signál zní téměř stejně jako originál. V každém ztrátovém kodeku je totiž obsažen tzv. psychoakustický model, což je vlastně matematické vyjádření možností lidského sluchu. Podle tohoto modelu kodek určuje, které informace má člověk největší šanci rozlišit. Ty, které nerozliší nebo které sice rozliší, ale již se nevejdou do komprimované podoby, pak vynechá. Obecně lze říci, že čím menší komprese, tím větší kvalita a současně větší objem dat a obráceně.

Na běžném kompaktním disku CD jsou digitalizovaná audio data uložena bez komprese. Vstupní signál je vzorkován frekvencí 44,1 kHz do 16 bitů a 2 kanálů. Z toho tedy vychází datový tok 1 411 kb/s (což při 60 minutovém záznamu odpovídá 635 MB dat).

Při použití bezztrátového kodeku FLAC lze dosáhnout snížení objemu dat zhruba na dvě třetiny. Při použití ztrátových kodeků se objem dat redukuje významně více, konkrétně vždy záleží na zadané míře komprese a jí odpovídajícímu snížení kvality. Samozřejmě také velmi záleží na kvalitě vlastního kodeku ;-) Pro kvalitní ukládání hudby v ztrátových formátech se používá datového toku 320 kb/s, pro kvalitní vysílání po Internetu se používá datového toku 128 kb/s.

Kvalitní kodeky dokáží vytvářet komprimovaný datový tok v pevné i proměnné rychlosti (tzv. bitrate). Bitrate udává průměrný počet bitů, které jsou použity na jednu sekundu záznamu. Při konstantní rychlosti – Constant Bitrate (CBR) je pro každou část digitalizovaných audio dat použit stejný počet bitů. Při proměnné rychlosti – Variable Bitrate (VBR) se kodér chová tak, že při řidším frekvenčním spektru a malé dynamice aktuálně kódovaného signálu generuje méně dat, zatímco při dynamickém signálu se širokým frekvenčním spektrem může použít většího objemu dat. Režim průměrné rychlosti – Average Bitrate (ABR) představuje střední cestu mezi CBR a ABR.

Nejpoužívanějším bezztrátovým kodekem je zřejmě FLAC, nejpoužívanějšími ztrátovými kodeky jsou zřejmě MP3 a Ogg Vorbis.

MP3

MP3, přesněji ISO-MPEG Audio Layer-3, je nejrozšířenější audio kompresní formát patentovaný německým institutem Fraunhofer-Gesellschaft. Držitel patentu může rozhodovat o licenčních poplatcích i o tom, co všechno lze s formátem dělat. Obliba MP3 je dána především setrvačností a stále ještě lepší podporou

přehrávačů. Formát sám o sobě nic lepšího nenabízí a v dnešní době ho lze považovat za překonaný. K dispozici je řada implementací kodérů, většina z nich podporuje jen CBR.

Ogg Vorbis

Ogg Vorbis je otevřený audio kompresní formát, který nabízí profesionální technologii kódování a streamování. Poměrem komprese/kvalita překonává kompresní formát MP3. Na rozdíl od něho není přitom zatížen žádnými patentovými nároky ani licencemi, které by omezovaly jeho volné a bezplatné používání. Ogg Vorbis je často ne zcela přesně nazýván zkráceně jako ogg. Ve skutečnosti je však ogg označení pro formát transportní vrstvy (tzv. kontejner), který může obsahovat data zakódovaná různými kodeky – např. Ogg Vorbis pro ztrátovou kompresi nebo FLAC pro kompresi bezztrátovou.

Streaming

Význam slova streaming je širší, obvykle se však používá pro vyjádření toho, že uživatel může poslouchat digitalizovaný zvuk (nebo sledovat video) ze současně stahovaných dat namísto toho, aby si musel nejprve kompletní data stáhnout a teprve poté je mohl přehrát.

V Internetu se pro vysílání rádií nejčastěji používá aplikace Shoutcast, která poskytuje streamy ve formátu MP3. Internetová rádia vysílají v naprosté většině svůj program z playlistu – dopředu enkódovaných souborů.

Popis systému

Vstupní signály – zdroje

S Českým rozhlasem jsem se dohodli, že program stanic ČRo 1, 2, 3, 6 a Region budeme přebírat z digitálního vysílání DVB-S a DVB-T. Program regionálních stanic ČRo Regina a ČRo Plzeň budeme přebírat z klasického analogového FM vysílání a program nově vznikajících stanic D-dur, Rádio Česko a Leonardo bude v podobě finálních ogg streamů dodávat Český rozhlas sám.

Kódování bude prováděno se vzorkovací frekvencí 48 kHz, na 16 bitech a 2 kanálech do Ogg Vorbis streamů ve variabilních rychlostech VBR 128 kb/s a 224 kb/s. Pro volbu vzorkovací frekvence 48 kHz hovoří fakt, že přináší jisté zlepšení zvuku oproti 44,1 kHz při nevýznamně zvětšeném datovém toku a pak také to, že streamy získávané z DVB jsou vzorkovány stejnou frekvencí. Streamy o rychlostech 128 kb/s jsou určeny náročnějším posluchačům, streamy 224 kb/s jsou pro posluchače náročné na kvalitu zvuku. Streamy o nižších rychlostech si Český rozhlas zajišťuje jinými prostředky ve vlastní režii.

Základní model

V základním modelu jsme systém pro internetové vysílání rozložili na kódovací servery pro pořizování komprimovaných streamů a na streamovací servery, k nimž se připojují koncoví klienti, kteří chtějí vysílání přijímat a streamy z nich odebírají. K tomuto rozhodnutí nás přivedly především odlišné nároky na proces kódování na straně jedné a na proces poskytování streamů na straně druhé. Kódovací server musí být vybaven značným výpočetním výkonem, protože provádí kódování do Ogg Vorbis formátu v různých rychlostech a pro několik vstupních signálů v reálném čase najednou. Nedostatek času procesoru nebo jiných systémových prostředků by se negativně promítnul do výstupních streamů v podobě zkreslení, výpadků nebo narůstajícího zpoždění.

Na všech zúčastněných serverech běží operační systémem GNU/Linux z distribuce Debian. Na úrovni programového vybavení jsme zvolili na pozici enkodérů aplikace eggenc a ices, na pozici streamovacího serveru aplikaci Icecast. Zmíněné produkty jsou vyvíjeny v rámci projektu Xiph.org pod GNU licencí a lze je volně použít.

Primárním kódovacím serverem je tun2.cesnet.cz umístěný na Západočeské univerzitě v Plzni. Fyzicky jde o server Dell PowerEdge 1800 v konfiguraci:

- 2× procesor P4 Xeon/3,4 GHz, 1 GB RAM,
- síťová karta Intel PRO/1000 Fibre,
- DVB-S karta Hauppauge WinTV NOVA-S-CI vybavená CI slotem,
- zvuková karta Sound Blaster Audigy.

Tento stroj produkuje z programů získávaných z DVB-S karty streamy rozhlasových stanic ČRo 1, 2, 3, 6 a Region. Dále generuje stream stanice ČRo Plzeň získávané ze signálu analogového tuneru připojeného na zvukovou kartu serveru. Všechny výstupní streamy jsou vysílány ve formátu Ogg Vorbis ve variabilních rychlostech (VBR) 128 a 224 kb/s na Icecast server běžící na stroji amp.cesnet.cz. Kromě toho jsou streamy získávané z DVB-S vysílány v nezměněném formátu MPEG-TS multicastem na přidělené skupinové adresy.

Druhý kódovací server tun1.cesnet.cz je umístěn na sálu CESNETu v Praze. Jde o server DELL PowerEdge 2600 v konfiguraci:

- 2× procesor P4 Xeon/2,4 GHz, 1 GB RAM,
- integrovaná síťová karta Intel PRO/1000,
- DVB-T karta Hauppauge WinTV-NOVA-T,
- zvuková karta Sound Blaster Audigy.

Server produkuje dva Ogg Vorbis streamy stanice ČRo Regina, která je získávána ze signálu analogového tuneru připojeného na zvukovou kartu serveru a vysílá je na Icecast server amp.cesnet.cz. Kromě toho produkuje tento stroj redundandní sekundární streamy stanic ČRo 1, 2, 3, 6 a Region, které pořizujeme z DVB-T.

Streamovací server amp.cesnet.cz je umístěn v prostorách CESNETu a jde o běžný server DELL PowerEdge 350 v konfiguraci:

- procesor P-III/1 GHz, 1 GB RAM,
- síťová karta Intel PRO/1000.

Vzhledem k tomu, že z DVB karty potřebujeme zpracovávat několik programů rozhlasových stanic současně, použili jsme pro příjem programů aplikační server vls. Ten dokáže přijímat všechny programy vysílané v daném paketu (v našem případě Czechlink) najednou, oddělit je od sebe a pak je v nezměněném formátu (MPEG-TS) poskytnout k dalšímu zpracování – např. vysílat na určené multicastové nebo unicastové adresy.

V našem řešení vysílá vls server programy ČRo 1, 2, 3, 6, Region a BBC v originálním formátu MPEG Transport Stream na vyhrazené multicastové adresy v síti CESNET. Tím umožňujeme uživatelům, kteří mají k multicastu v síti CESNET přístup, přijímat streamy v nezměněné podobě. Ke zpracování těchto streamů doporučujeme použít klienta vlc, který je vyvíjen pod GNU licencí a lze jej volně použít na řadě dnes dostupných operačních systémů.

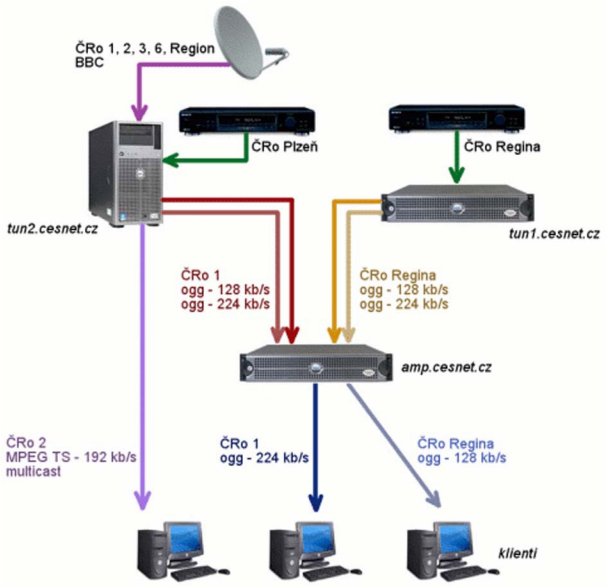
Každý MPEG-TS stream vysílaný aplikací vls přijímá na témže serveru klient vlc, který stream dekóduje do podoby čistého PCM signálu. Ten je následně převáděn kodérem oggenc do Ogg Vorbis streamů v rychostech 128 a 224 kb/s, které jsou předávány aplikaci ices k transportu na stremovací server amp.cesnet.cz.

Pro větší srozumitelnost popišeme rozdíl mezi vysílanými MPEG-TS a ogg streamy. MPEG-TS streamy vysílané z kódovacího serveru tun2.cesnet.cz aplikací vls nejsou poznamenány žádnou následnou konverzí a jsou vysílány v originální podobě získané při příjmu ze satelitu (MPEG-TS, vzorkovací frekvence 48 kHz, datový tok 192 kb/s). vls server vysílá daný stream na určenou konkrétní adresu, takže tento způsob nelze použít pro připojování běžných (unicastových) klientů. S výhodou lze použít vysílání na multicastovou adresu, přes kterou mohou daný stream přijímat všichni klienti, kteří se k ní mohou připojit – v našem případě jde o omezené množství uživatelů sítě CESNET, jejichž počítače mají multicastovou konektivitu.

Ogg streamy vysílané aplikací Icecast ze serveru amp.cesnet.cz prošly transkódovacím procesem z formátu MPEG-TS do PCM a poté do Ogg Vorbis (vzorkovací frekvence 48 kHz, toky 128 nebo 224 kb/s), ale tato konverze je při běžném poslechu a dosažené vysoké kvalitě neznatelná. Ogg streamy nabízené Icecast serverem jsou přitom dostupné běžným protokolem HTTP všem uživatelům

v Internetu. Z koncových klientů (přehrávačů), kteří podporují zpracování ogg streamů, se nejčastěji používají winamp, xmms, zinf, foobar2000, qcd, a další.

Topologii zapojení celého systému ukazuje obrázek 1.



Obr. 1 Schéma zapojení zařízení

Streamovací server amp.cesnet.cz je zapojen do běžné sítě IPv4 i do nastupující IPv6 sítě a své streamy poskytuje klientům v obou těchto sítích.

V síti IPv4 vysílá na adresách

- http://radio.cesnet.cz:8000/mount_point_daneho_streamu
  např. http://radio.cesnet.cz:8000/cro3.ogg

a v síti IPv6 vysílá na adresách

- http://amp-ipv6.cesnet.cz:8006/mount_point_daneho_streamu
  např. http://amp-ipv6.cesnet.cz:8006/cro3.ogg

Klienti se k serveru připojují nejčastěji pomocí odkazů na WWW stránkách živého vysílání Českého rozhlasu, odkazů na indexových serverech internetových rádií nebo přímo na domovské stránce streamovacího serveru. CGI skripty stojící za příslušnými odkazy odešlou WWW prohlížeči odpověď s patřičnou hlavičkou, podle níž se na klientské stanici automaticky spustí přehrávač zvuku, který začne vlastní stream přijímat a přehrávat. Skript nastavuje patřičnou MIME Content-type položku

```
Content-type: audio/x-scpls
```

a určuje adresu streamu v podobě

```
protokol://adresa_serveru:port/jmeno_streamu
```

Na WWW stránce bývá zapsán pod odkazem typu http://radio.cesnet.cz/cgi-bin/cro1-ogg.pls.

Obsah souboru cro1-ogg.pls vypadá takto:

```
#!/bin/sh

cat >> ++++END++++
Content-type: audio/x-scpls

[playlist]
numberofentries=1
File1=http://amp.cesnet.cz:8000/cro1.ogg
Title1=CRo1 - Radiozurnal
Length1=-1
Version=2
++++END++++
```

Kompletní seznam streamů vysílaných naším systémem je uveden na adrese http://radio.cesnet.cz/.

Vysílání nových stanic Českého rozhlasu

Ve spolupráci s kolegy z Českého rozhlasu jsme se podíleli na návrhu a realizaci systému pro vysílání nových stanic Českého rozhlasu D-dur, Rádio Česko a Leonardo. Vysílání těchto stanic bylo spuštěno v internetové podobě ještě před jejich zařazením do vysílání v rámci multiplexu DVB-T. Stanice Český rozhlas D-dur vysílá 24 hodin denně klasickou hudbu od renesance až po tvorbu 21. století, nízkorozpočtový program Rádio Česko je obsahově i formátem založen čistě na zpravodajství a na aktuální publicistice a stanice Český rozhlas Leonardo se zaměřuje na popularizaci vědy, techniky, přírody, historie a medicíny.

Vysílání stanice D-dur se odbavuje pomocí vysílací stanice systému DALET z playlistu, který se sestavuje ve speciálním interním systému ČRo. Systém DALET používá interní zvukový formát MP2, 48 kHz, 256 kbps. Vysílání stanic Rádio Česko a Leonardo je kombinací vysílání živého a záznamů z playlistu. Audio signál je na sál přiveden přes hlavní přepojovač ve formátu MADI a tam se pomocí speciálního zařízení převádí na formát ADAT. Signál v ADAT formátu je přiveden na počítač/enkodér (PC, Pentium 4, 3,0 GHz, Debian, ices-2.0-kh60,

alsa, jack audio server 0.99), který je vybaven 8 kanálovou ADAT zvukovou kartou. Tento stroj vytváří ogg streamy v různých rychlostech a ty se distribuují na vysílací servery. Použitý JACK audio server umožňuje zpracování každého kanálu vstupního signálu ADAT odděleně.

Streamy s nižšími rychlostmi 22 a 48 kb/s jsou posílány na streamovací server Českého rozhlasu stream.rozhlas.cz, streamy 128 a 224 kb/s jsou posílány na streamovací server CESNETu amp.cesnet.cz.

Odstranění zpoždění při streamování živého vysílání

Vnitřní architektura producentů streamů (aplikací, které vytvářejí digitální stream a transportují ho na streamovací server) bývá navržena tak, aby co nejlépe eliminovala potíže vzniklé případnými výpadky ve vstupním signálu, krátkodobým nedostatkem systémových prostředků potřebných pro enkódování nebo transkódování vstupního signálu a výpadky v transportu výstupního streamu na streamovací server. Architektura producentů tak odpovídá požadavkům dnes významně převažujícího streamování z již předem enkódovaných audio dat (tzv. playlistu), jak ho používají internetová rádia.

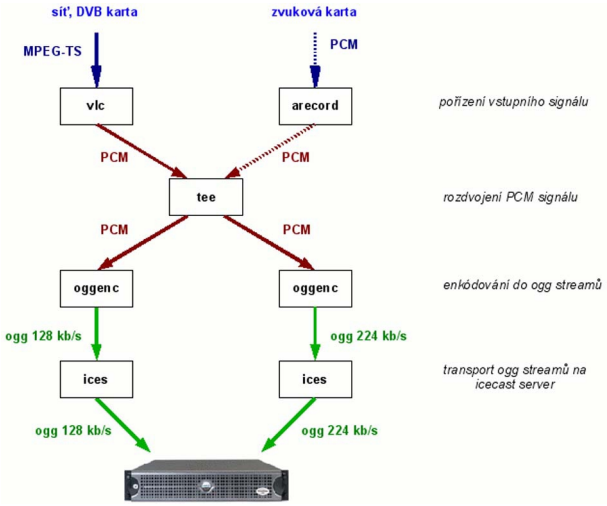
V reálném provozu to znamená, že je producent vybaven velkým vstupním a výstupním zásobníkem, do kterého ukládá vstupní a výstupní data streamu. V situaci, kdy se aplikaci producenta nedostává dostatek strojového času procesoru potřebného pro enkódování vstupního signálu, začne stream zpožďovat a data vstupního signálu začíná hromadit ve vstupním zásobníku. V případě výpadku spojení na streamovací server zase začíná hromadit výstupní enkódovaná data v zásobníku výstupním. V obou případech je cílem, aby posluchač (jeho klient) dostal data pokud možno kompletní a to i za cenu zpoždění přijímaného výsledného streamu oproti času, kdy vstoupil do procesu enkódování. To nepředstavuje problém při vysílání ze záznamu/playlistu. Ovšem při streamingu živého vysílání rozhlasových stanic je to nepříjemné, protože při dlouhodobém běhu aplikace producenta může docházet k postupnému narůstání doby zpoždění (2 sekundy za den znamenají již 1 minutu za měsíc), které může způsobovat nežádoucí opoždění přijímaného programu oproti časovému schématu. Citelně to vadí např. při posunu vysílaného časového znamení, opoždění začátků programů apod. Přítomnost krátkodobého výpadku (který ostatně nemusí být ani slyšitelný) ve streamu živého vysílání je přitom akceptovatelná.

Tento problém se obvykle řeší nočním restartem aplikace producenta, takže zpoždění nemůže narůstat více, než v rámci jednoho dne. To však není elegantní řešení a navíc způsobuje odpojení všech klientů, kteří daný stream přijímají (neuvažujeme-li redundandní systém streamování).

Navrhli jsme proto řešení, které vznik časového zpoždění ve streamech živého vysílání eliminuje. Jeho podstata je založena na vyloučení procesu enkódování vstupního signálu z aplikace producenta (ices), kde ke zpoždění dochází a jeho

přesun do předřazené aplikace enkodéru oggenc. Enkodér pracuje s minimálním vstupním bufferem, takže v případě silně zvýšené zátěže kódovacího serveru začne aplikace vlc svá výstupní PCM data přiváděná na vstup oggenc zahazovat. To je však přijatelné a v běžném provozu jsou tyto výpadky téměř neslyšitelné. Na vstup aplikace producenta ices jsou pak přiváděna data již enkódovaná do finálního ogg streamu, s nimiž producent neprovádí žádné transkódování a přímo je vysílá na cílový streamovací Icecast server.

Celý řetězec pořízení ogg streamu znázorňuje obrázek 2.



Obr. 2 Schéma procesu enkódování

Zdrojový signál je pořízen buď aplikací vlc z dat MPEG-TS streamu anebo programem arecord, který čte data ze zvukové karty. Obě aplikace převádí vstupní signál na čistá PCM data, která jsou přiváděna na vstup oggenc enkodéru. Abychom se vyhnuli následnému transkódování vytvářeného ogg streamu z rychlosti 224 kb/s na 128 kb/s, které by bylo provázeno snížením kvality, rozdvojujeme tok PCM dat do dvou proudů, které přivádíme na dvojici oggenc enkodérů. První provádí enkódování dat do streamu 128 kb/s a druhý generuje nezávislý ogg stream 224 kb/s. Oba streamy jsou pak přiváděny na vstupy aplikací producentů ices, kteří je beze změny transportují na Icecast server(y). Data jsou mezi jednotlivými aplikacemi posílána uvnitř pojmenovaných rour. Tímto způsobem lze zpracovávat zdrojový signál z libovolné aplikace, která jej může převádět do PCM dat.

Uvedenou kompozici producenta ogg streamů používáme na enkódovacích serverech CESNETu tun1.cesnet.cz a tun2.cesnet.cz již několik měsíců. Nežádoucí zpoždění odstraňuje a pracuje spolehlivě.

Zvýšení robustnosti systému a jeho odolnosti proti výpadkům

Provoz systému pro digitální přenos audiosignálu ve vysoké kvalitě představuje nároky nejen na zvukovou kvalitu, ale i na velkou odolnost proti výpadkům. Nejde přitom jen o výpadky způsobené nějakou chybou (těch je minimálně), ale každý server vyžaduje průběžnou údržbu. Zvýšení úrovně odolnosti lze dosáhnout především redundancí a robustností systému a použitím softwarové inteligence.

V původním zapojení jsme měli dva enkódovací servery produkující ogg streamy v rychlostech 128 a 224 kb/s se vzorkovací frekvencí 48 kHz – tun1.cesnet.cz a tun2.cesnet.cz, které vytvářely ogg streamy a posílaly je na Icecast server amp.cesnet.cz, k němuž se pak připojují klienti uživatelů.

Pro zvýšení odolnosti proti výpadkům jsme spustili druhý streamovací server amp2.cesnet.cz, na který jsou vysílány všechny streamy ČRo stejně jako na amp1. Když se klient připojuje k požadovanému streamu, nejčastěji si nejprve stahuje (aniž by to uživatel registroval) z www serveru definiční soubor, který mimo jiného obsahuje informace o jménu a adrese streamu. V tomto souboru lze definovat více adres serverů, které stream poskytují. Klient se poté připojí k prvnímu definovanému streamu v pořadí. Dojde-li pak následně během příjmu k výpadku streamu, klient se automaticky přepojí k streamu definovanému jako druhému v pořadí, atd. Této technologie jsme využili ke zvýšení odolnosti celého systému proti výpadku streamovacího Icecast serveru.

Výpadek zdrojových ogg streamů generovaných enkódovacím serverem jsme eliminovali jejich zduplikováním. K původním streamům ČRo vytvářených na tun2 jsme na stroji tun1 spustili redundandní sekundární streamování vysílání stanic ČRo 1, 2, 3, 6 a Region, které pořizujeme z DVB-T. Sekundární ogg streamy vytváříme v rychlosti 128 kb/s a vysíláme je na oba Icecast servery amp1 a amp2. V konfiguraci Icecastu jsme přitom sekundární streamy definovali jako záložní (fallback), které server použije s automatickým přepojením klientů v případě, že přišel o stream primární. Poté, co je příjem primárních streamů obnoven, přepojí Icecast server klienty automaticky zase zpět. K tomu, aby mohl Icecast server přepojit klienta na jeho původně zvolený stream v situaci, kdy k primárním streamům ogg-224 a ogg-128 kb/s existuje jen jeden záložní stream ogg-128 kb/s, je třeba vytvořit v konfiguraci Icecastu kromě definice fallbacku i rozdvojený lokální mount-point pro každý záložní stream. Tuto techniku ukazuje následující část konfigurace Icecast serveru:

```
<relay>
     <server>127.0.0.1</server>
     <port>8000</port>
     <mount>/z-cro1.ogg</mount>
```

```
        <local-mount>/fall-cro1.ogg</local-mount>
    </relay>
    <mount>
        <mount-name>/cro1.ogg</mount-name>
        <fallback-mount>/fall-cro1.ogg</fallback-mount>
        <fallback-override>1</fallback-override>
    </mount>
    <relay>
        <server>127.0.0.1</server>
        <port>8000</port>
        <mount>/z-cro1.ogg</mount>
        <local-mount>/fall-cro1-256.ogg</local-mount>
    </relay>
    <mount>
        <mount-name>/cro1-256.ogg</mount-name>
        <fallback-mount>/fall-cro1-256.ogg</fallback-mount>
        <fallback-override>1</fallback-override>
    </mount>
```

Nyní tedy provozujeme systém s dvojím zálohováním. Dva nezávislé enkódovací servery posílají stejné streamy ČRo na dva nezávislé streamovací servery. Celý systém je tak odolný proti současnému výpadku jednoho enkódovacího a jednoho streamovacího serveru.

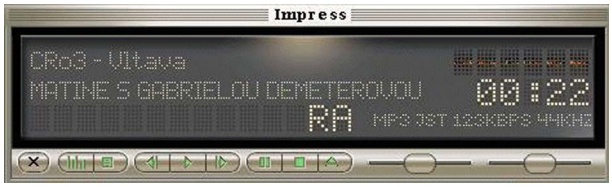
Reálný provoz systému

Provoz celého systému je stabilní. Při zpracování signálů 6 stanic a jejich transkódování do 12 výstupních Ogg Vorbis streamů vykazuje server tun2 zatížení/load kolem 1.5 a idle kolem 65 %. Streamovací server pracuje pod variabilní zátěží, která však nepřekračuje hodnotu 0.5 při 200 současně přihlášených klientech.

Kvalitu zvuku považujeme za vysokou. Bez pochybností nejvyšší kvalitu zaručují ogg streamy 224 kb/s, které se zvukově velmi blíží CD, jejich zvuková scéna je čistá. U streamů 128 kb/s je zvuková scéna trochu užší a méně přehledná. Tyto rozdíly jsou patrné zejména při soustředěném poslechu na kvalitnějším zařízení, běžné počítačové „bedničky“ tyto rozdíly částečně stírají. Ze strany posluchačů je kvalita všech streamů hodnocena výborně.

Časové zpoždění přijímaného streamu je velmi malé (1–3 sekundy) a je dáno především nastavením velikosti vyrovnávací paměti (bufferu) koncového klienta. Nastavení větší velikosti eliminuje občasné výpadky nebo zpoždění přenášených dat na méně propustných linkách a logicky zvyšuje zpoždění. V případě přijímání streamu 224 kb/s doporučujeme zvýšit velikost vyrovnávací paměti, protože přijímaných dat je více a buffer udržuje data na méně než 1 sekundu dopředu.

Vysílané streamy doplňujeme doprovodnou textovou informací (tzv. metadaty) o právě vysílaném programu. Podkladem nám jsou popisné soubory s minutáží získávané exportem ze systému denní programové skladby vysílání Českého rozhlasu do formátu xml.



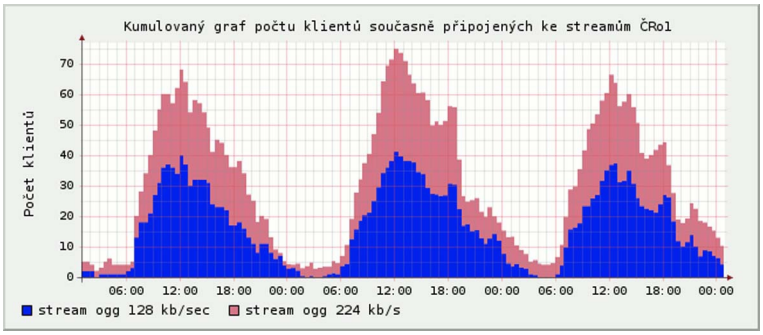
Obr. 3 Zvukový klient Quintessential Player a metadata

Obr. 4 Zvukový klient Freeamp a metadata

Podle původního předpokladu jsme chtěli vstupní audio signály přebírat přímo z linkového výstupu odbavovacího pracoviště Českého rozhlasu a uvažovali jsme o tom, že by kódovací server byl umístěn v budově ČRo. Při tomto řešení bychom museli nepřetržitě vysílat ze sítě ČRo na streamovací server v CESNETu 16 streamů, na což však není současný kanál připojení počítačové sítě ČRo do Internetu připraven. Proto jsme jako alternativu použili získávání digitálního signálu z DVB a analogového signálu z FM tunerů a kódovací servery jsme tak mohli ponechat na sálech CESNETu a ZČU. Obávali jsme se, že příjem z FM tuneru bude mít značný negativní vliv na zvukovou kvalitu příslušných streamů, protože frekvenční rozsah zvukového signálu z FM tuneru (resp. vysílače) je zúžen do pásma 30–15 000 Hz oproti pásmu 2–20 000 Hz používaného u CD. Technici z ČRo nám pro srovnání poskytli zvukový záznam pořízený přímo z výstupu odbavovacího pracoviště. Zvuková kvalita signálu získaného z DVB je s ním srovnatelná, signál získávaný z tuneru je o poznání horší. Připojením kódovacího serveru přímo k odbavovacímu pracovišti ČRo bychom tedy v současnosti dalšího významného zvýšení kvality nedosáhli.

Statistiky

Ze statistik streamovacího serveru amp.cesnet.cz je vidět zřetelný trend rostoucího zájmu o nejkvalitnější streamy enkódované v rychlosti 224 kb/s. Zatímco před rokem byl poměr posluchačů streamů 128 kb/s vůči 224 kb/s přibližně 3 : 1, dnes je už téměř vyrovnaný. To potvrzuje oprávněnost našeho záměru soustředit se právě na přenosy audia ve velmi vysoké kvalitě.



Obr. 5 Kumulovaný graf počtu klientů současně připojených k ogg streamům 128 a 224 kb/s ČRo 1

Průměry denních statistik ukazují, že streamy odebírá 5 000 uživatelů, délka poslechu je 30 minut a server přenese 150 GB dat.

Zajímavosti a technické podrobnosti

Naše původní představy o podobě experimentálního systému byly založeny na tom, že rozdělíme proces primárního zpracování vstupního signálu a proces jeho transkódování mezi dva vzdálené stroje. Chtěli jsme použít málo výkonný server s DVB-S kartou, který by aplikací vls posílal MPEG-TS streamy přijaté ze satelitu multicastem do sítě – k tomu není potřeba velkého výpočetního výkonu ani jiných systémových prostředků. Ze sítě by pak stream mohli přijímat libovolní klienti – tedy i klient vlc na našem kódovacím serveru vybaveném patřičným výkonem CPU potřebným na transkódování streamu.

Zjistili jsme však kritické nároky na zajištění zcela rovnoměrné datové propustnosti mezi serverem s aplikací vls a serverem s klientem vlc. Přestože byly oba servery připojeny na gigabitovou páteřní síť do stejné VLAN, objevovaly se v PCM datech klienta vlc občasné poruchy v podobě „frekvenčního uklouznutí“ zvuku. Slyšitelé to bylo např. při hře smyčcových nástrojů nebo zpěvu, kdy se zdálo, že hudebníci zahráli falešně. K těmto potížím docházelo zřídka a jen na několik sekund, takže odhalit příčinu nebylo snadné. Problém částečně

eliminovalo zařazení několikasekundového bufferu na vstup klienta vlc a použití protokolu rtp, ale nebyl odstraněn zcela. Po provední různých měření jsme dospěli k závěru, že na problému se podílí chvilkové vyšší zatížení propojovacích aktivních prvků (jinak však dostatečně dimenzovaných). Experimentálně jsme ověřili, že by vyhovovalo vzájemné propojení obou serverů kříženým kabelem, ale při tom by musely být oba servery ve stejné lokalitě a blízko sebe, takže oddělení části primárního zpracování (vls) od produkce ogg streamů by pozbylo smyslu. Proto jsme se rozhodli umístit obě části na stejný server s dostatečným výkonem s tím, že do sítě vysílá originální MPEG-TS i transkódované ogg streamy současně. Přenos ogg streamů z kódovacího serveru na Icecast server již není na případné nerovnoměrnosti datové propustnosti kritický a výsledný zvuk je bez chyby.

Dále jsme se setkali s tím, že při výpadku trasy mezi kódovacím serverem a Icecast serverem na dobu delší než několik sekund nenaváže aplikace ices se serverem nové spojení, přestože v systému zůstává v běhu. K těmto situacím ztráty vzájemné konektivity téměř nedochází, ale vyloučit zcela je nelze.

Použili jsme proto upravenou neoficiální verzi aplikace ices-2.0-kh59, která se chová tak, že se při přerušení spojení s Icecast serverem ukončí. My spouštíme celý proces transkódování v shellu ve smyčce, takže v případě ukončení běhu ices počkáme několik sekund a proces opět spustíme. Tím jsme zajistili automatické obnovení funkce celého systému po znovuzískání konektivity mezi oběma servery.

Odkazy

http://www.cesnet.cz/doc/techzpravy/2003/icecast/

http://www.cesnet.cz/doc/techzpravy/2004/audio/

http://www.cesnet.cz/doc/techzpravy/2005/audio/

http://radio.cesnet.cz/

http://www.rozhlas.cz/

http://www.icecast.org/

http://mediacast1.com/∼karl/

http://www.videolan.org/

http://darkice.sourceforge.net/

http://www.debian.org/

http://www.alsa-project.org/

Fixed Mobile Convergence from an IP network perspective

Pavel Křižanovský

E-mail: pkrizano@cisco.com

Fixed Mobile Convergence (FMC) is a term with lots of meanings and is used very often today. This article should show which technologies and concepts can be found behind this term and what are the motivations for service providers (SPs) to implement such mechanisms in their networks.

Last couple of years brought major changes in SP business. Fixed line operators ("classical" telcos) are facing problems of declining PSTN voice revenues and a loss of fixed access lines caused by fixed-to-mobile line substitution. On the other hand mobile operators growing exponentially in the end of nineties and in the beginning of this millennium are now facing the mobile market saturation and therefore slowdown of the classical mobile business revenues. In several countries including Czech Republic we can even see mobile penetration more then hundred percent.

These trends are forcing both kinds of service providers to seek for some new services to offer to attract their customers which can help them to generate more revenue.

Wireline operators found their chance in wider range offerings of voice and data services including high speed broadband access and VoIP based voice services. Their near future can be also found in video services, namely Video-on-Demand and IP-TV.

Mobile operators can find new possibilities in Fixed Mobile Convergence technologies.

Theoreticians tell us that FMC basically consist of three layers:

- **Network Convergence**

  The Network Convergence layer is focused on reducing operation expenses by converging multiple fixed and mobile networks onto a common IP/MPLS core supporting a range of access options, including PSTN, DSL, leased lines, Metro Ethernet, WLAN and RAN for Mobile networks.

- **Services Convergence**

  The Service Convergence layer provides session control function and is therefore enabling new IP services such as mobile data access, rich media conferencing, voice or messaging. The session awareness enables to make service available on any terminal and over any access even allowing for seamless handoff of active sessions between different access types. In addition the service convergence layer ensures that any IP services is allocated appropriate resources and enable services to be billed, enabling high value IP Services.

- **Application Convergence**

  The application convergence layer includes the actual services operators are going to market with. Examples are seamless data services available over any access, or a dual mode business voice service.

In the rest of the article will be dedicated mainly to the network convergence and also the application convergence, namely dual mode voice services.

Network convergence

Most of the mobile operators are already using IP/MPLS technology in their core network, namely MPLS VPN. This enables them to use one network as a transport infrastructure for independent data traffic types, but how about mobile voice?

Speaking about european countries, ninety nine percent of mobile voice is based on GSM technology (2 or 2.5 generation networks) and now 3G networks, namely UMTS are on the way (but the arrival of it is not as fast as.expected in the beginning of millenium).

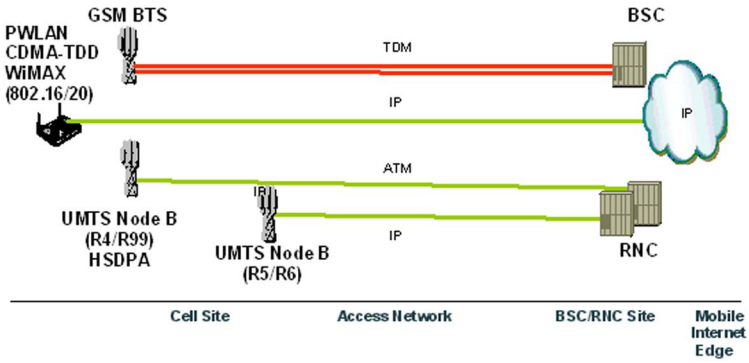
GSM is based on TDM (Time Division Multiplex) and TDM technologies have usually problems to be migrated onto IP networks mainly because of synchronization problems.

UMTS is different. Current (3GPP R99/R4) implementations are usually based on ATM technology, but this is also not easy to be seamlessly converged onto one IP/MPLS network.

These are actual basic voice services, but we have to play also with data services like wifi hotspots and/or mobile broadband data services, such as CDMA2000, TDD UMTS or OFDM services. The data part of the services is easily integratable to the core IP/MPLS, but the services have to be delivered not only to the MPLS core sites.

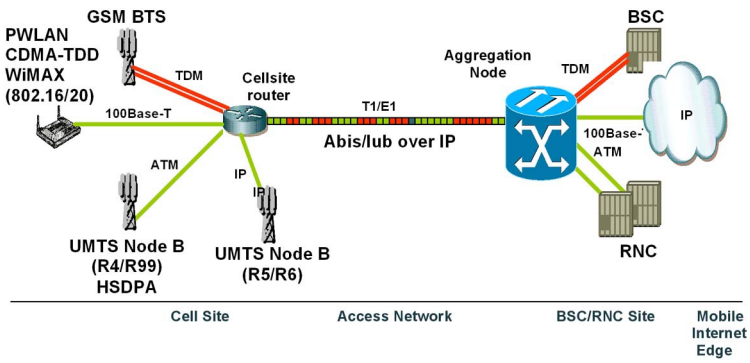
This means, that mobile operators are typically using several totally independent technologies in their Radio Access network (RAN): TDM, ATM and IP.

They usually have at least one TDM E1 line for GSM, one ATM E1 for UMTS andsome IP link for other data services (including their management) The possibility to go is to converge all these technologies onto one common platform – IP network.



Unfortunately there are no widely usable complex standards yet to do it efficiently (to save the total number of links from cellsites) and therefore vendors are using say about proprietary solutions to adopt these technologies onto one IP network.

For example Cisco is using sampling technique to adopt GSM traffic into pure IP. GSM uses Abis protocol between BTS and BSC devices in the RAN. The technique rely on lossless compression which is periodically taking samples from the TDM link and sending only the bits samples changed during the last sampling period. The samples are packaged into IP packets using PPP Multilink encapsulation The source of compression are mainly idle channels (no calls) and silence frames On the other side the TDM bit stream is regenerated from the incomming IP traffic.



Between Node-B and RNC devices UMTS uses Iub protocol over ATM (AAL2/AAL5 PVCs). The important information from ATM cells is again en-

capsulated into IP, sent through the backhaul line and on the other side the ATM PVC is regenerated from the IP stream. The sources of compression in this case are mainly uninteresting ATM cell and AAL layer headers.

As we can see on the following picture all the compressed traffic can be sent through the IP backhaul line(s) together with other backhaul traffic.

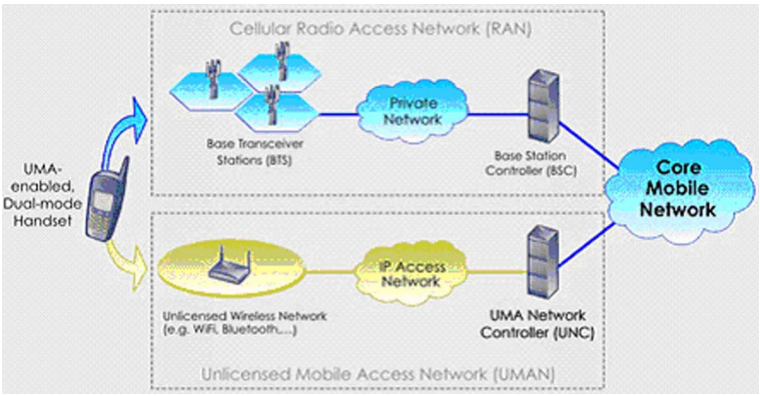
There other mechanisms and standards like Any Transport over MPLS (AToM) or Circuit Emulaiton over IP, which can be combined for further integration of nonIP TDM/ATM traffic into IP/MPLS networks but this was just one example of fixed mobile convergence in network layer.

Application convergence

One of the best FMC examples on application layer are GAN (Generic Access Network, formerly UMA – Unlicensed Mobile Access) and IMS (IP Multimedia Subsystem).

GAN

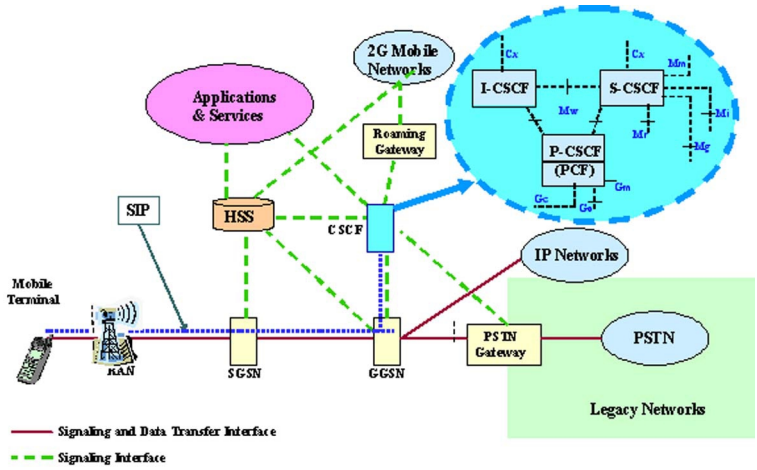
The concept of GAN is indeed simple and attractive: tunnel GSM/GPRS traffic from a GAN client, through an IP network to a GAN Network Controller in order to provide seamless GSM/GPRS service via alternative IP based access technologies such as WiFi. The reality of GAN implementation means that operators need to understand the impacts of GAN on existing and new security threats, IP backbone dimensioning and approaches to IP transit and peering together with important considerations as to the quality of experience for services delivered over an IP access network.



Source: UMA consortium and 3GPP

IMS

The IP Multimedia Subsystem (IMS) is a service delivery framework for providing SIP based multi-media services to users over IP networks. Such a framework approach to service delivery allows those common functionalities to be deployed once and then leveraged by all IMS applications. These common functionalities include user authentication, identity management, security, charging, QoS interactions, service creation and roaming. Whilst IMS standardization has been driven by the cellular community, because it is based on IETF protocols, IMS concepts are now being considered for adoption by other market segments. Specifically ETSI's TISPAN project is adopting IMS architecture for a fixed network environment.



IMS is more futureproof solution but its complexity makes it very difficult to fully develop and take it into commercial life. The concept of GAN is much simpler to be implemented in existing mobile operators' networks and we expect it to be the one who will become the reality first.

Fixed-mobile konvergence z pohledu dodavatele telekomunikační infrastruktury

Jiří Gogela

E-mail: Jiri.Gogela@nokia.com

Historie a současnost telekomunikačního trhu

Mobilní telekomunikační sítě vznikající v poslední dekádě 20. století představovaly paralelní telekomunikační infrastrukturu k zavedeným fixním sítím. Tato infrastruktura byla považována za velice specifickou a nepříliš kompatibilní s tradičními systémy fixních sítí. Intergrace obou prostředí se tak, až na vzácné výjimky, omezovala pouze na přepojování telefonických hovorů.

Z ekonomického hlediska znamenalo zavedení mobilních služeb značné zvýšení konkurenčního tlaku na provozovatele fixních sítí. Paradoxně v mnoha případech docházelo k vnitřní konkurenci ve firmách, které poskytovaly oba druhy služeb. Počátkem 21. století se – díky klesajícím cenám služeb – staly mobilní sítě skutečnou ekonomickou hrozbou pro tradiční operátory. V mnoha případech hospodářské výsledky mobilních divizí ‚zachraňovaly' ekonomickou situaci mateřských telekomunikačních společností. V tomto prostředí dochází k situaci, kdy spojení mobilního operátora s poskytovatelem fixních služeb není již chápáno jako konkurenční výhoda, ale naopak jako jistá přítěž.

V několika posledních letech se ovšem objevují nové technologické ‚hrozby', které mohou mít dramatický dopad na doposud dominující mobilní operátory. Jedná se o služby založené na komunikaci přes Internet, které pro uživatele představují dramatické snížení nákladů. Tyto internetové aplikace pokrývají značnou část existujících služeb, např. hlasová komunikace – Skype, instant messaging – ICQ atd. Stále rozšířenější mobilní terminály z kategorie smartphone založené na operačních systémech Symbian, Windows Mobile apod. jsou schopny podporovat klienty zmíněných aplikací, a mobilní operátoři tak stojí před stejnou hrozbou jako jejich konkurence v oblasti pevných linek. Jsou v reálném nebezpečí, že v horizontu několika let budou degradováni z poskytovatelů hlasových a datových služeb pouze na poskytovatele internetového připojení.

Fixed-mobile konvergence (FMC) z pohledu operátora

Výše uváděné faktory vedou k hledání cest k novým službám, které by umožnily zachování současného ekonomického modelu. Jejich hlavní ideou je sbližování fixních a mobilních služeb, většinou je tento přístup označován jako konvergence fixních a mobilních sítí (fixed-mobile convergency FMC).

FMC technologie ‚vrací do hry' fixní operátory, propojení mobilního a fixního operátora se stává silnou konkurenční výhodou, neboť ADSL připojení domácností je klíčovým faktorem pro rozšíření FMC. Pokud operátoři začnou nabízet výrazně výhodnější tarify pro volání prostřednictvím VoIP přes ADSL linky, bude tato služba pravděpodobně silnou motivací pro růst poptávky po ADSL připojení. Pro mobilní operátory znamená využití ADSL značný offload linek a snížené náklady na kapacitu sítě, zejména v rezidenčních oblastech.

Na určitých specifických trzích, např. v USA, přináší FMC podstatné zlepšení dostupnosti sítí mobilních operátorů (v řadě oblastí je nesmírně obtížné zajistit indoor pokrytí radiovým signálem z důvodu nesouhlasu majitelů pozemků s výstavbou základnových stanic).

Fixed-mobile konvergence (FMC) z pohledu uživatele

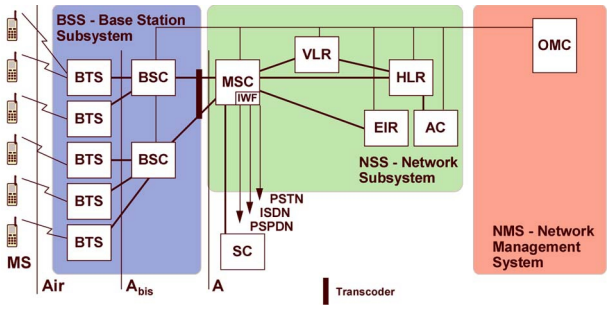
Hlavním bodem FMC je zachování mobilního telefonu jako komunikačního prostředku s tím, že uživatel bude moci pro přístup ke službám sítě využít v podstatě libovolný kanál. Jinými slovy: se stejným telefonem a stejným telefonním číslem se uživatel dovolá jak prostřednictvím standardní GSM sítě, tak i přes WiFi hotspot připojený k Internetu. Bude-li např. uživatel využívat pro volání WiFi síť připojenou prostřednictvím ADSL ze svého domova, bude mu účtována výrazně nižší cena.

Pro firemní zákazníky představuje FMC možnost využít firemní WiFi síť pro interní hlasovou komunikaci bez nutnosti používat více telefonních terminálů.

Technologie pro FMC

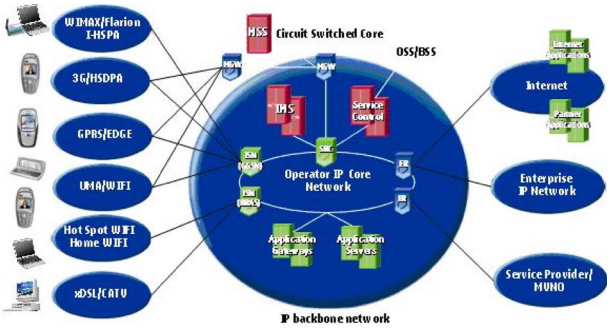
Problematika FMC je poměrně široká – zasahuje oblasti od účtování služeb až po zabezpečení sítí. V příspěvku se zaměříme především na systémy přístupových technologií.

Tradiční architekturu mobilní sítě (GSM) znázorňuje schéma na obr. 1:



Obr. 1

Jak bylo uvedeno, nové technologie přinášejí požadavek na univerzální přístup – to znamená vývoj architektury sítě směrem k multi-access prostředí (viz obr. 2:



Obr. 2

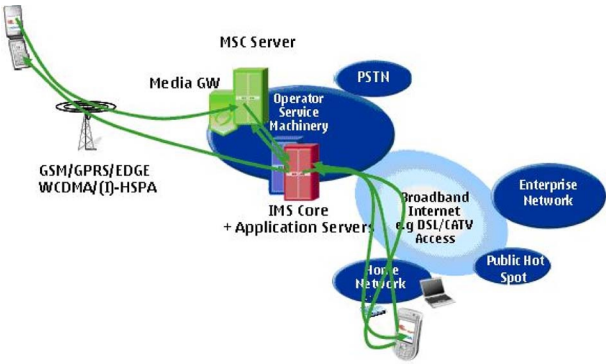
Základem rozšíření FMC technologií je pokrytí domácností a firemních pracovišť vhodným prostředkem pro bezdrátovou komunikaci. V současnosti jsou nejdostupnější tyto dvě alternativy:

- picocell – jedná se o miniaturní GSM základnové stanice, vybavené IP rozhranním. Tato základnová stanice o nízkém výkonu zajistí pokrytí signálem v okruhu několika metrů (eventuálně desítek metrů). Nesporným kladem tohoto řešení je naprostá kompatibilita se všemi existujícími telefony, hlavním záporem je relativně obtížná integrace těchto stanic do stávajících sítí.
- IP over WiFi/BlueTooth – ověřené a rozšířené technologie, problémem zde je omezené množství mobilních terminálů, které je podporují.

Pro využití hlasových služeb mobilních operátorů prostřednictvím IP sítí jsou upřednostňovány systémy UMA (Unlicenced Mobile Access) a ‚native‘ VoIP

(Voice over IP). Zatímco UMA zachovává podstatné principy GSM komunikace (autentikace prostřednictvím SIM, standardní A-interface pro integraci do MSC, atd.), native IP přináší zcela novou koncepci založenou na spíše IT standardech.

Základní princip zpracování VoIP telefonního hovoru v síti mobilního oprátora je zřejmý ze schématu na obr. 3:



Obr. 3

Jádrem řešení na straně operátora je IMS (IP Multimedia System), jedná se o zařízení zajišťující autentizaci uživatelů. S využitím protokolu SIP iniciuje uživatel spojení, pro přenos hlasu v síti slouží protokol RTP, který je zpracován ‚ústřednou' MSC (Mobile Switching Center) stejně jako ostatní běžné hovory v síti.

Dopad FMC na existující systémy

Jak je patrné, zavádění prvků FMC přináší mnohem větší otevřenost do poměrně striktního a kontrolovaného světa telekomunikační infrastruktury. Přístup ke službám prostřednictvím Internetu přináší výzvy v podobě podstatně zvýšených nároků na zabezpečení sítě. Otevírají se otázky ochrany samotných uživatelů před zneužitím jejich identity (SIP) a samozřejmě se zvyšuje tlak na standardizaci v oblasti IP telefonie. Dosavadní proprietární řešení typu Skype apod. nejsou dostačující, neboť fakt, že v oblasti telekomunikací působí značné množství výrobců infrastruktury i terminálů, vyžaduje poměrně detailní standardizaci.

Prameny

http://www.3gpp.org/

http://www.umatechnology.org/

http://www.nokia.com/A482005