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

36. konference

Sborník příspěvků

Hotel Maxičky
Děčín
16.–19. května 2010

Programový výbor

Vladimír Rudolf
Jan Kynčl
Václav Pergl
Václav Matyáš

Sborník příspěvků z 36. konference EurOpen.CZ, 16.–19. května 2010

Plzeň 2010. První vydání.
Editor: Vladimír Rudolf
Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis, Plzeň
e-mail: servis@vydavatelskyservis.cz

Tisk: Typos, tiskařské závody, s. r. o.
Podnikatelská 1160/14, Plzeň

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-19-8

Obsah

Fedora a Red Hat Enterprise Linux

Miloslav Trmač

E-mail: mitr@redhat.com

1 Operační systém Linux

Linux je plnohodnotný operační systém Unixového typu[1], který lze použít jako běžnou uživatelskou stanici, síťový server, nebo v mnoha specializovaných prostředích (např. clustery pro náročné výpočty, WiFi routery, mobilní telefony).

„Běžný“ uživatel na Linuxu ocení široký záběr dostupného software: součástí operačního systému je kromě základního prostředí celá řada aplikací včetně kancelářského balíku OpenOffice.org, schopného používat souborové formáty MS Office. Linux byl od začátku koncipován tak, aby mohl být používán několika uživateli zároveň; tato koncepce (např. striktní oddělení role běžného uživatele a správce systému) vede k vysoké úrovni stability a bezpečnosti.

Správcům počítačů pomáhá to, že Linux je složen z velkého množství poměrně oddělených komponent, což usnadňuje hledání příčin problémů a dává správci poměrně velké možnosti na rozhraní těchto komponent systém přizpůsobovat svým potřebám. Linux obsahuje velmi silné prostředky pro psaní skriptů (ad hoc programování podle potřeby uživatele, např. automatizace často prováděných činností). Psaní skriptů usnadňuje i to, že většina dat v systému je uložena v textovém formátu, který lze zpracovávat univerzálními nástroji.

Pro organizace nasazení Linuxu obecně nabízí flexibilitu, nezávislost na jediné firmě dodávající software, a nižší náklady na správu.

2 Vývojový model Open Source

Linux je mezi mainstreamovými operačními systémy unikátní svým vývojovým modelem: Zatímco většina softwarových produktů je nabízena firmami, které plně kontrolují vývoj produktu a vyžadují platbu za každou nasazenou kopii, Open Source software lze volně kopírovat, nasadit a dále šířit zcela zdarma.

Bezplatně dostupné nejsou jen spustitelné kopie software, ale – a to hlavně – i jeho *zdrojový kód*, forma programu, se kterou pracují programátoři. To každému

[1] Dříve bylo možné mluvit o klonu Unixu, dnes spíše přebírají Unixové operační systémy technologie z Linuxu.

umožňuje zkoumat, jak program funguje, a provádět v něm úpravy podle vlastních potřeb. Tyto úpravy pak uživatel může poslat původním autorům (nebo spíše správcům) programu. Program tedy není vyvíjen nějakou uzavřenou organizací, ale širší komunitou zainteresovaných uživatelů.

Bezprostředním důsledkem tohoto modelu je relativně vyšší kvalita a spolehlivost software (uživatelé mohou pomoci s hledáním a opravou i těch chyb, na které původním autorům nestačí kapacita), a jeho přizpůsobení pro potřeby (obzvláště programujících) uživatelů. Software také není existenčně závislý na svých původních autorech; v případě, že původní autor práci na programu z libovolného důvodu ukončí, dřívější uživatelé jej mohou dále udržovat a vyvíjet.

Model Open Source je výhodný i pro neprogramující uživatele: Protože zdrojové kódy software jsou k dispozici, uživatel není vydán na milost dodavateli. Kdyby dodavatel přestal z libovolného důvodu uživateli vyhovovat (nedostatečná kvalita služeb, prudké zvýšení cen, vynucený upgrade na novější verzi), může si uživatel nechat software spravovat a vyvíjet jiným dodavatelem. Firmy, které se živí tvorbou a úpravami Open Source software, tedy nemohou „usnout na vavřínech“ a jsou nuceny každodenně o své klienty bojovat kvalitou svých služeb.

Vysoké školy a jiné vzdělávací instituce mají v Open Source software k dispozici výborný nástroj pro praktické vzdělávání studentů. Studentům lze na Open Source software ukázat, jak jsou v praxi implementovány a používány technologie, o kterých se ve škole učí (např. jádra operačních systémů nebo překladače), bez složitého vyjednávání o zpřístupnění zdrojovým kódům software.

Open Source je také užitečné pro zadávání praktických studentských prací (ročníkových projektů, bakalářských nebo diplomových prací). Student může svůj program po odevzdání místo odložení „do šuplíku“ dát pod Open Source licencí k dispozici uživatelům na celém světě, čímž získá cenné zkušenosti s praktickými potřebami uživatelů a komunikaci s nimi. V případech, kdy by napsání samostatného většího programu bylo pro studentskou práci příliš rozsáhlé, mohou studenti místo toho pracovat na vylepšeních existujících Open Source programů.

3 Fedora a Red Hat Enterprise Linux

Fedora a Red Hat Enterprise Linux jsou *distribuce* operačního systému Linux. Linux není jednolitý projekt vyvíjený jednou komunitou programátorů: existují stovky drobnějších komunit, z nichž se každá věnuje jednomu balíku software. Úkolem „distribuce“ je z takových balíků software vybrat ty nejdůležitější a nejužitečnější a sestavit z nich integrovaný, dobře spolupracující operační systém. Každá distribuce např. nabízí jednotný mechanismus pro instalaci aktualizací, takže se základní knihovny, ovladače zařízení, serverové aplikace, kancelářský

software nebo zásuvné moduly do WWW prohlížečů všechny aktualizují jednotným mechanismem.

Fedora je distribuce Linuxu zaměřená na rychlý vývoj a inovace Linuxové platformy; nové verze vychází pravidelně dvakrát ročně a jsou udržovány přibližně 13 měsíců. Nové technologie vyvíjené v Linuxu nebo pro Linux (např. SELinux, PulseAudio, IcedTea, NetworkManager, SystemTap) jsou do distribuce zařazovány v době, kdy jsou užitečné pro většinu uživatelů, i když vývoj není úplně dokončen nebo když nová technologie není schopna dokonale nahradit předchozí implementaci: Fedora funguje jako platforma, kde jsou tyto technologie v praxi nasazeny u velkého množství uživatelů a kde jsou dokončeny a upraveny podle zkušeností z těchto nasazení.

Vývoj distribuce Fedora je sponzorován společností Red Hat; kromě zaměstnanců firmy Red Hat se ale na vývoji podílí více než dvacet tisíc jiných programátorů, autorů dokumentace, překladatelů a jiných spolupracovníků.

Red Hat Enterprise Linux (dále „RHEL“) je operační systém založený na distribuci Fedora, určený pro běžné nasazení v podniku, včetně „mission-critical“ aplikací. Vývojový cyklus je podstatně delší (18–24 měsíců mezi hlavními verzemi), a každá hlavní verze je podporována alespoň 7 let. Nová hlavní verze RHEL vznikne z vybrané verze Fedory výběrem důležitých a spolehlivých komponent, a prochází více než ročním testováním, laděním výkonu, a dalšími úpravami. Red Hat nové funkce pro RHEL primárně vyvíjí v distribuci Fedora: uživatelé Fedory tedy získají nové technologie, a Red Hat může využít zkušenosti uživatelů pro jejich zdokonalení. RHEL je pro zákazníky dostupný bez licenčních poplatků; uživatelé platí jen pravidelné roční poplatky za podporu, přístup k aktualizacím a další služby.

Fedora a RHEL obsahují plně integrovanou virtualizaci založenou na technologii KVM: virtuální stroje lze spouštět, spravovat a migrovat přímo z prostředí operačního systému, bez potřeby instalace odděleného virtualizačního řešení.

Jednou z nejdůležitějších oblastí, ve které Red Hat soustavně pracuje na vylepšování Linuxové platformy, je zvyšování bezpečnosti systému. Dříve šlo o mechanismy ztěžující zneužití bezpečnostních děr, souhrnně nazývané Exec Shield (non-executable data, randomizace adres, automatická kontrola přetečení bufferu nebo paměti na haldě). V dnešní době je primární oblastí SELinux, „Mandatory Access Control“ mechanismus původně vyvinutý americkou NSA. SELinux přiřazuje každému objektu (soubor, proces, socket, ...) informace o „typu“ objektu a definuje pravidla pro to, jak spolu mohou interagovat objekty různých typů.

Zásadním rozdílem tohoto mechanismu oproti obvyklým oprávněním u souborů je, že pravidla SELinuxu jsou závazná pro všechny uživatele; uživatel tedy nemůže (nechtěně ani úmyslně) zpřístupnit objekt jiným uživatelům nebo objektům způsobem, který odporuje nastaveným pravidlům. Tak lze např. zabránit

zásuvným modulům WWW prohlížeče číst uživatelům soukromý SSH klíč, nebo omezit programy běžící jako uživatel `root` tak, aby mohly měnit jen ty aspekty systému, které jsou pro jejich práci nezbytně nutné. I kdyby se útočníkovi podařilo převzít kontrolu nad některým procesem, pravidla SELinuxu útočníka omezují (útok na WWW server by například mohl umožnit útočníkovi změnit obsah nabízený přes WWW, ale nenechá jej vytvářet další síťová spojení přímo přistupovat k hardware).

4 Novinky v Red Hat Enterprise Linux 6

Nová hlavní verze RHEL 6, se blíží dokončení; 21. dubna byla vydána první veřejná beta verze. Uživatelům dává přístup k novým funkcím vyvinutým na Linuxové platformě za poslední 3 roky (některé z nich již byly zpětně integrovány do RHEL 5). RHEL 6 bude nabízen pro 32-bitové a 64-bitové architektury x86, IBM System z a 64-bitové IBM Power.

V jádře Linuxu obecně došlo k úpravám pro lepší využití velkého množství procesorů a paměti. „Tickless kernel“, modifikace, kdy pro časování není třeba zbytečně pravidelně budit procesor, snižuje spotřebu napájení a zvyšuje efektivitu virtualizovaných systémů. Spotřebu napájení dále šetří podpora ALPM (dočasného vypínání SATA spojení). Jádro také podporuje „control groups“, které umožňují správci systému rezervovat určitou část výkonu (CPU, paměti, síťové kapacity) pro konkrétní procesy, nebo naopak spotřebu konkrétních procesů omezit.

Jako výchozí systém souborů se nově používá ext4, nabízející např. zvětšení limitů pro velikost jednotlivých souborů a celého systému souborů, používání „extentů“ a rychlejší kontrolu konzistence systému souborů. Jako alternativa pro velmi velké systémy souborů je nově k dispozici XFS.

RHEL 6 používá pro virtualizaci technologii KVM, která je dostupná i pro nejnovější verze RHEL 5. Nově je podporováno přidávání PCI zařízení za běhu virtuálního stroje a sdílení paměti mezi běžícími virtuálními stroji. RHEL 6 nadále může běžet jako virtuální stroj v Xen.

Z hlediska bezpečnosti systému došlo k dalšímu rozvoji SELinuxu: nová je možnost uzavřít konkrétní proces do „sandboxu“, kde má velice malé možnosti interagovat s okolím, integrace SELinuxu do X11 (XACE, umožňuje omezit komunikaci mezi X klienty), a použití SELinux pro ochranu virtuálních počítačů mezi sebou navzájem (sVirt). Nová je také podpora pro „key escrow“ klíčů používaných pro šifrování disků.

Pro nasazení uživatelských stanic s centrálně spravovanými účty je užitečný nový démon `sssd`, který zprostředkovává přístup k centrálním databázím informací o účtech, a je schopen tyto informace uložit do cache pro případy, kdy centrální databáze není dostupná.

Vývojáři ocení mimo jiné novou verzi debuggeru `gdb`, která je rozšiřitelná v jazyku Python; takové rozšíření se automaticky používá pro výpis hodnot standardních tříd jazyka C++ (např `string`, `list`).

„Automatic Bug Reporting Tool“ sbírá informace o pádech aplikací a umožňuje uživateli předat potřebné informace vývojářům. Standardní nastavení posílá informace do systému pro sledování chyb u společnosti Red Hat, správce systému ale může místo toho tyto informace sbírat a vyhodnocovat lokálně v rámci firmy.

Grafické uživatelské prostředí bylo samozřejmě aktualizováno na nejnovější verze X11, GNOME a KDE.

Prohlídka systému Android

Michal Zbortek

E-mail: michal@zbortek.cz

Abstrakt

Tento příspěvek představí hlavní vlastnosti nového operačního systému Android od společnosti Google. Bude ukázáno, s čím se potýká běžný uživatel ve většině případů, jaké výhody spatřuje a zároveň, které funkce mu chybí. Ukáže se, zda půjde většina těchto nedostatků doplnit aplikacemi. Dále bude nastíněna vnitřní architektura systému a možnosti vývoje aplikací pro tuto platformu.

1 Úvod

Android je poměrně nová mobilní platforma vznikající za účasti společnosti Google. Systém je postaven na jádře Linux a je naprogramován v jazyce Java. Zatím se vyskytuje pouze v mobilních telefonech, ale je plánován i do netbooků. Android byl vydán pod licencí Apache 2.0.

2 Historie

V červnu v roce 2005 Google koupil společnost Android Inc., která nastartovala základy systému. V listopadu 2007 byla založena Open Handset Alliance a současně ohlášena platforma Android. V tomto měsíci byla vydána první předváděcí verze vývojového balíku, obsahovala základní nástroje včetně emulátoru, dokumentace a ukázkových programů. První ostrá verze 1.0 byla představena v září 2008 a v říjnu byl Android uvolněn jako open source.

Prvním telefonem se systémem Android se na konci září roku 2008 stal T-Mobile G1, který vyráběla firma HTC. Následovaly ho telefony HTC Magic a HTC Hero. Ostatní společnosti jej začali také zařazovat do svých produktů, mezi ně patří Samsung, Motorola, Sony Ericsson a další. V dnešní době je již uvolněno celkem 34 zařízení.

Android se neustále vyvíjí. Na konci dubna 2009 vznikla aktualizace 1.5 (kódové jméno Cupcake), která nově obsahovala softwarovou klávesnici, lepší podporu bluetooth a různé vylepšení a zjednoduššení na ovládání. Verze 1.6 (kódové jméno Donut) přinesla v polovině srpna možnost nahrávání videa a podporu pro vyšší rozlišení WVGA. Aktuální verze je 2.1 (kódové označení Eclair) dodala lepší optimalizace pro využití hardware, změnu uživatelského rozhraní, podporu blesku fotoaparátu, a jiné.

3 Z pohledu obyčejného uživatele

Po vybalení telefonu a prvním spuštění nás čeká průvodce nastavením. Můžeme spustit nabízený tutoriál, který nás provede základními činnostmi, jako je například zadávání textu. Následně máme možnost nastavit přihlášení k účtu na serverech Google, je to nepovinný krok a získáme tím možnost synchronizace kontaktů, kalendáře, a dalších služeb. Na posledním dialogu se systém zeptá na povolení sběru dat pro statistické účely, po jehož zavření se dostáváme na hlavní domovskou obrazovku.

Domovská obrazovka je aplikací, která se spouští při stisknutí tlačítka Home na telefonu. Slouží pro spouštění aplikací a pro organizaci plochy, na kterou můžeme vkládat zástupce, zkratky do konkrétních sekcí, složky a živé součásti nazývané jako widgety. Ty běží jako samostatné programy a instalují se úplně stejně jako obyčejné aplikace. Těchto ploch je několik a přepínáme mezi nimi gesty doprava a doleva. Od verze 2.1 má android novou verzi této domovské aplikace, která navýšila počet ploch ze tří na pět a obsahuje drobné vylepšení uživatelského rozhraní.

Home v Android 1.1

Home v Android 2.1

První verze Androidu neměly softwarovou klávesnici, protože tehdy jediný model telefonu měl vlastní QWERTY klávesnici. Po uvolnění přistrojů bez ní android vyřešil tuto situaci klavesnicí zobrazenou na displeji.Výhodou je, že běží jako samostatná aplikace a můžeme ji vyměnit instalací jiné. Zadávání textu vyvoláme stiskem do prostoru editačního pole.

Pro účel snadné instalace programů a správy aktualizací, Google vytvořil databázi aplikací, na kterou přistupujeme přes aplikaci Market. Systém automaticky kontroluje všechny naše nainstalované součásti a upozorňuje na nové aktualizace. Bohužel v České republice máme k dispozici pouze aplikace, které jsou zdarma, protože Google stále nemá zařízené podmínky pro jejich nabízení na českém trhu.

V Androidu byl od počátku nativně implementován multitouch, avšak kvůli sporům ohledně patentu společnosti Apple byl zakázán v jádře systému. Od verze 2.0 je multitouch v systému plně podporován a je na aplikacích, zda jej začlení do svého ovládání.

4 Z pohledu zkušeného uživatele

Zkušenější uživatelé (geekové) ocení, že Android je založený na systému Linux se všemi jeho výhodami. Mezi největší výhodu bude patřit konzole, kde máme možnost přes příkazy některé procedury provést efektivnějším způsobem.

Ovšem v základu bez práva superuživatele (root) bychom toho moc nedokázali. Naštěstí téměř každý přístroj se dá tzv. „rootnout“, rozdíly jsou v náročnosti tohoto úkonu. První model T-Mobile G1 ve svém prvním buildu obsahoval chybu, která způsobila, že cokoliv bylo napsáno na klávesnici, bylo vykonáno pod uživatelem root. Bylo tedy potřeba provést downgrade na tuto verzi a později znovu aktualizovat. Nexus One má tento proces o dost snadnější, stačí pouze spustit příkaz, který umožní instalace libovolného buildu Androidu.

Mezi další výhody patří:

- **Aplikace uložené na SD kartě** – u aktuální verze systému stále ještě neexistuje možnost ukládání aplikací mimo hlavní paměť. Za pomoci skriptů, které většina buildů již obsahuje, můžeme tento nedostatek vyplnit.

- **Tethering** – je sdílení připojení pro další zařízení, nejčastěji počítač. Modifikovaný systém umožňuje sdílení přes Bluetooth, Wi-Fi i USB kabel.

- **Optimalizace výkonu** – upravené buildy systému často obsahují vlastní jádro, které mívá úpravy pro lepší využití operační paměti a celkově hardware.

5 Z pohledu vývojáře

Většina operačního systému je napsána v jazyce Java. Přístroje, na kterých běží, mají podporu Javy přímo v procesoru.

Google nabízí vývojový balíček SDK, který je k dispozici pro 3 největší desktopové operační systémy Windows, Linux a Mac OS X. Jako nejvhodnější vývojové prostředí nám nabízí platformu Eclipse, pro kterou existuje plugin, který mnohem zesnadňuje práci. Obsahuje grafický editor GUI a zajišťuje spoustění aplikací v emulátoru nebo přímo v zařízení.

Protože Java může být na některé kritické operace pomalejší, Google začal pracovat na dalším vývojovém balíku, které má označení NDK. Umožňuje vytváření knihoven do nativního kódu v jazyce „C“.

6 Závěr

Operační systém Android je stále ve vývoji, má plno nedostatků, ale jeho vývoj jde neustále kupředu a podle těchto tendencí by měl zanedlouho obsahovat všechny důležité součásti, které jsou uživateli vyžadovány.

Literatura

[1] *Android Developers.* http://developer.android.com/

[2] *Forum XDA Developers.* http://forum.xda-developers.com/

Synchronizace s klientskými aplikacemi

Pavel Dobrý

E-mail: pdobry@kerio.com

Abstrakt

Klientské aplikace tvoří pro uživatele základní pracovní prostředí. V následujících kapitolách si ukážeme jaká úskalí a problémy se mohou vyskytnout při synchronizaci e-mailů, kalendářů a kontaktů ze serveru do těchto klientů. Objasníme si základy synchronizační teorie a stručný přehled používaných protokolů. Zaměříme se také na aktuální situaci na poli synchronizace s mobilními telefony.

Úvod

Při vývoji groupwarového serveru je jednou z nejdůležitějších vlastností podpora klientských aplikací, ve kterých uživatel pracuje se svými daty. Jako groupware označujeme veškerá data a programy sloužící k efektivní a produktivní spolupráci více uživatelů. Zahrnuje e-mailové zprávy, kalendářové události, plánování úkolů a sdílení všech těchto informací mezi uživateli. Každého hned jistě napadne, že přístup k grouwarovým datům je možné realizovat online např. prostřednictvím webového klienta. Zřejmou nevýhodou této možnosti je podmínka trvalého spojení se serverem. Naproti tomu samostatné klientské aplikace, které fungují jako e-mailový, kalendářový nebo adresářový klient nabízí větší možnosti, včetně režimu práce off-line, tedy bez okamžitého přístupu na server. Je ale třeba data ze serveru na klienta a opačně přenášet, tj. synchronizovat.

Synchronizace je opakující se proces výměny informací mezi dvěma aplikacemi, jehož výsledkem je stav, kdy obě strany (aplikace) obsahují shodná data. Tohoto ideálního stavu dosáhnout pouze v případě, že obě strany jsou stejné aplikace a data se v podstatě replikují. Tolik strohá definice. V případě synchronizace groupwarových dat (e-mailů, kalendářů nebo kontaktů) se ale množina synchronizovaných dat liší podle možností klienta, způsobu práce s ním nebo s ohledem na technická omezení při přenosu dat.

Synchronizační metody

Zaměřme se nyní na možné způsoby, jak lze informace synchronizovat. Základním požadavkem na proces synchronizace je schopnost přenášet změny provedené na serveru nebo klientovi na opačnou stranu. Ne vždy je ale nutné přenášet změny oběma směry. Jednosměrná synchronizace (one-way sync) se využívá při vytváření archívu dat nebo pro klienty sloužící k prezentování informací. Příkladem může být třeba veřejně přístupný webový kalendář. Server představuje jediný a primární zdroj a role klienta neumožňuje data jakkoliv modifikovat. Tato metoda synchronizace je viditelně jednodušší na implementaci ovšem její využití je omezené.

Naproti tomu obousměrná synchronizace (two-way sync) je mnohem rozšířenější. Uživatel zcela přirozeně očekává, že může svá data v klientovi měnit, posílat e-maily a pozvánky na schůzky. A že všechny změny budou na server uloženy.

Groupwarové servery používají nejčastěji přímou obousměrnou synchronizaci s klienty. Topologie spojení má tvar hvězdy, server slouží jako centrální bod pro synchronizaci všech klientů. Zároveň také obsahuje všechna data v kompletní podobě. Klienti pak synchronizují jen určité podmnožiny dat z uživatelských schránek. Podmnožina může být definována časovým intervalem (jen emaily došlé v minulých 3 dnech), typem synchronizovaných dat (pouze kalendář) nebo jejich organizačním uspořádáním (jen konkrétní e-mailová složka).

Synchronizace je vždy spouštěna klientem a klient také řídí její průběh, zejména rychlost a pořadí v jakém se přenášejí jednotlivé bloky položek. Samotný stav synchronizace pak může být kontrolován jednou nebo i oběma stranami. Stav synchronizace určuje, jaká data už byla na klienta (server) odeslána a které změny je třeba v dalším synchronizačním požadavku ještě poslat.

Aktuálnost dat zajišťuje opakování synchronizace. Další synchronizace může být buď pravidelná podle definovaného časového intervalu, nebo okamžitá, při jakékoliv změně v datech. Nastane-li změna na klientovi, může klient spustit synchronizaci ihned. O něco složitější je situace, pokud změna nastane na serveru. Pro tyto případy navazují synchronizační protokoly tzv. zpětný kanál, který umožňuje serveru poslat na klienta notifikaci o nastalé změně (Push notifikace). Klient pak na základě notifikace může zahájit synchronizaci. Zpětným kanálem bývá nejčastěji trvale navázané HTTP spojení.

Synchronizační konflikty

Konflikty jsou přirozenou součástí synchronizačního procesu. Konflikt nastává, pokud došlo ke změně konkrétní synchronizované položky na obou stranách a tato změna není shodná. Např. pokud se změní telefonní číslo v kontaktu

uloženém na serveru a uživatel mezitím ve svém klientu provede tutéž změnu a zadá jiné telefonní číslo. Při následné synchronizaci se tyto změny potkají a vzniká konflikt.

Synchronizační konflikty musí být vždy vyřešeny, aby byla synchronizace úspěšná.

Existuje několik různých strategií pro řešení konfliktů:

- Dát uživateli výběr, nechť rozhodne, která změna platí.
- Server vždy vyhrává.
- Klient vždy vyhrává.

Jak je vidět, žádné z těchto řešení není ideální. Uživatel je obtěžován rozhodováním a potvrzováním. Toto lze akceptovat, pokud data patří pouze jednomu uživateli. V případě dat sdílených mezi uživateli nebo informací umístěných ve veřejných složkách ale uživatel nemusí být schopen určit, která změna je vlastně „správná“. Strategie „jedna strana vždy vítězí“ je výhodnější, protože zaručuje konzistentní přístup k řešení konfliktů.

Používané protokoly

Pro synchronizaci groupwarových dat existuje několik protokolů. Některé protokoly jsou standardizované a některé hojně rozšířené. Na projektu našeho groupwarového serveru jsme postupně implementovali všechny používané protokoly.

V e-mailových klientech se používá nejčastěji POP3 a IMAP. Tyto protokoly nelze považovat za čistě synchronizační, nicméně jejich rozšíření a všeobecná podpora je řadí do kategorie povinných v každém groupware serveru.

Asi nejznámějším představitelem otevřeného synchronizačního protokolu je **SyncML** [2] (Synchronization Markup Language). Protokol je standardizovaný pod hlavičkou Open Mobile Alliance. Umožňuje synchronizovat prakticky libovolný typ dat a je velmi univerzální. Jako transportní protokol se používá nejčastěji HTTP. Relativní nevýhodou SyncML je jeho slabší rozšíření a podpora. Používají jej zejména mobilní telefony, nicméně se velmi různí v tom, jaká data lze se serverem synchronizovat. Často je možné synchronizovat pouze PIM data (kalendář, kontakty, poznámky) V podnikové sféře má velmi silné konkurenty, za kterými pokulhává, i když je v mnoha ohledech univerzálnější. Na vině je zejména nedostatečná podpora pro správu vzdálených zařízení.

Dominantní postavení má v tomto směru firma Microsoft a její proprietární protokol **Exchange ActiveSync** [1]. Díky obrovské základně serverů v podnikovém prostředí se firmě podařilo tento protokol pasovat na „standard“ v oblasti synchronizace mobilních zařízení.

SyncML ani ActiveSync se příliš nehodí pro synchronizaci desktopových aplikací nebo velkých objemů dat. Proto se firmy spojené v organizaci CalConnect rozhodly vypracovat nové protokoly založené na standardu WebDAV, které by umožnily synchronizaci kalendářů a kontaktů. Protokoly **CalDAV** a **CardDAV** [3] jsou otevřené a je možné je použít i pro synchronizaci mobilních zařízení. Průkopníkem v této oblasti je firma Apple a její produkty iPhone, iPod a iPad.

Otevřený nebo uzavřený protokol?

Největší nevýhodou uzavřených protokolů je fakt, že bývají patentovány a s jejich používáním jsou spojeny licenční poplatky. Pro komerční produkty je rozsáhlost ekosystému a množství zařízení, které tyto protokoly nativně podporují, velkým lákadlem, které dokáže vyvážit náklady.

Ze zkušenosti lze říci, že z hlediska implementační složitosti nebo problémů s kompatibilitou zde není mezi protokoly žádný rozdíl. Vše se odvíjí od poctivosti výrobců software pro mobilní telefony a serverových řešení. Ani otevřený standard s detailním popisem protokolu automaticky nezaručuje plnou kompatibilitu se všemi klienty.

Desktopové aplikace

Typickým příkladem groupwarové desktopové aplikace je klient Microsoft Outlook nebo Microsoft Entourage.

Klientské aplikace běžící na desktopu prakticky nemají omezen diskový prostor pro ukládání dat. Tato výhoda představuje paradoxně zároveň i problém. Chceme-li synchronizovat všechny e-mailové složky se všemi zprávami nebo kompletní kalendáře, dostává se do hry časový faktor. Získat a přenést několik GB dat ze serveru na klienta může trvat desítky minut, někdy řádově i hodiny, pokud server obsluhuje několik desítek či stovek klientů najednou. Synchronizace mimo lokální síť bývá navíc zpomalována kapacitou internetového připojení. Během této doby uživatel nemůže plnohodnotně pracovat, protože nemá e-maily nebo kontakty dostupné.

Možné řešení si můžeme ukázat na praktickém případě vývoje nového klienta. Pro každého vývojáře serveru je taková situace lákavá, neboť má pod kontrolou obě strany a může vytvořit skutečně efektivní řešení. Jaké možnosti tedy máme?

- Přednostní synchronizace důležitých položek. Nejpotřebnější data, která uživatel vyžaduje pro práci, synchronizujeme přednostně. Např. kontakty, kalendář a složka s doručenou poštou.

- Částečná rychlá synchronizace. Při synchronizaci e-mailových zpráv hlavičky e-mailu a jeho t extovou část synchronizujeme jako první a objemné přílohy až poté.
- Synchronizace dat podle stáří. Nejnovější e-maily se stahují jako první.

V praxi se ukazuje, že tento problém je vhodné řešit jen u té části groupwarových dat, která představuje největší objem. Jedná se hlavně o e-mailové zprávy, které spolu s přílohami zabírají bezkonkurenčně největší prostor. Kalendáře, poznámky, úkoly nebo seznamy kontaktů jsou v porovnání s e-maily mnohem menší a synchronizují se najednou.

Mobilní zařízení

Synchronizace mobilních telefonů, PDA, tabletů a dalších přenosných zařízení zažívá v současné době boom. Zásluhu na tom má i nevídané rozšíření „chytrých“ mobilních telefonů. I když se jedná o technicky velmi výkonné přístroje, ve srovnání s desktopovými počítači je zde pár podstatných odlišností:

- Omezená velikost paměti.
- Pomalejší datové připojení.
- Provoz na baterie.

Tyto specifické vlastnosti se přímo dotýkají i požadavků na jejich synchronizaci a strukturu synchronizačního protokolu.

Limitovaná velikost flash paměti na mobilním zařízení omezuje množství dat, která může klient uložit. Pro synchronizaci je tedy nutné nějakým způsobem definovat pouze podmnožinu všech dat dostupných na serveru, které se budou přenášet. Toto základní nastavení provádí uživatel přímo v klientovi při definování přístupu na serverový účet. U e-mailových složek je možné vybrat, které se mají synchronizovat, a také jak staré zprávy chce uživatel mít na svém telefonu. Kontakty se synchronizují vždy kompletní, neboť zároveň slouží jako telefonní seznam v telefonu. Pro kalendář je k dispozici omezení na stáří událostí podle jejich výskytu, např. pouze jeden měsíc zpátky.

Dalším ovlivňujícím faktorem je rychlost připojení k serveru. Pokud se nenacházíme v dosahu volné Wi-Fi sítě, je jedinou další možností datové připojení mobilních operátorů. Jeho rychlost je značně kolísavá, od pomalého GPRS až po UMTS. Někdy je dokonce takové připojení zpoplatněno podle přenesených dat. V praxi se proto snažíme o maximální snížení objemu přenášených dat. Některé synchronizační protokoly s tím počítají, jako např. Exchange ActiveSync, který přenáší data v tokenizované podobě zakódované do WBXML[4]. Další úspory lze

dosáhnout využitím možností transportního protokolu HTTP, který se takřka výhradně používá. Zapojením gzip komprese přenášených dat a využitím keep-alive HTTP spojení se sníží množství přenesených dat. U zabezpečených HTTPS spojení navíc odpadá režie SSL/TLS vrstvy při navazování každého spojení.

Mobilní telefony prakticky

Asi největší problém při synchronizaci mobilních telefonů představuje jejich velká variabilita. Každý výrobce má svoji představu o funkcích telefonu a toho co je možné synchronizovat. To často vede ke zklamání na straně uživatele i na straně vývojáře serveru.

Slýcháme pak často podobné otázky:

- Proč nevidím u kontaktu fotku?
- Jak to, že tam vůbec nemám kalendář?
- Můžu vidět i jinou složku než Inbox?

Příčinou je ve všech případech omezení na straně mobilního telefonu. Ani ne tak hardwarové, jako softwarové.

Pro synchronizaci jsou telefony obyčejně vybaveny jedním z klientů podporujících SyncML nebo Exchange ActiveSync. SyncML má v podnikovém prostředí zatím velmi slabou pozici, slouží především k zálohování kontaktů a kalendářů. Existuje i několik hostovaných služeb v Internetu, které nabízí zálohování zdarma. ActiveSync naproti tomu umožňuje synchronizovat vše včetně e-mailu v rámci jednoho snadno nastavitelného účtu. Navíc podporuje i další služby jako vyhledávání kontaktů ve veřejném adresáři na serveru, plánování událostí na serveru, nastavování automatické odpovědi v nepřítomnosti nebo vzdálené vymazání zařízení při jeho ztrátě nebo odcizení.

Pojďme se nyní zaměřit na nejpoužívanější mobilní telefony, a co všechno umí.

Windows Mobile

Platforma Windows Mobile má nejdelší historii v oboru synchronizace. Postupem času se spektrum synchronizovaných informací rozšiřuje a v nejnovější verzi 7 zahrnuje např. i ukládání SMS na server. Windows Mobile si stále zachovává v oblasti synchronizace status etalonu, ke kterému se snaží ostatní mobilních zařízení dostat. I tak zde nalezneme jistá omezení jako např. zobrazení pouze jednoho kalendáře nebo adresáře s kontakty.

Apple iPhone

Mobilní telefon iPhone, stejně jako iPod Touch a nebo současná novinka iPad, nastavil zcela nové standardy v ovládání mobilních zařízení. Podpora protokolu ActiveSync byla do iPhone přidána ve verzi 2.0 a doplnila tak stávajícího e-mailového klienta, který uměl pouze IMAP a POP3. S verzí 3.0 představenou loňský rok přišla i očekávaná změna a příchod podpory protokolů CalDAV a iCalendar pro synchronizaci kalendářů.

iPhone umí synchronizovat jakoukoliv e-mailovou složku a jako jediný je schopen také synchronizovat i více kalendářů nebo kontaktových složek ze serveru. Plně podporuje plánování událostí a vyhledávání e-mailů a kontaktů na serveru. iPhonu chybí aplikace pro správu úkolů.

Nokia

Nokia má velké spektrum mobilních telefonů s různými verzemi operačních systémů Symbian a Maemo. Pro většinu z nich dodává zdarma aplikaci Mail for Exchange, která synchronizuje groupwarová data s ActiveSync serverem. Díky velkému rozptylu ve verzích a modelech telefonů se ale jedná o nejvíce problémovou platformu. Verze aplikace Mail for Exchange se liší pro každý model telefonu a stejně tak i jejich funkcionalita. Vydávání updatů pak více než co jiného připomíná chaos.

Problém s kompatibilitou, který se u Nokií často vyskytuje, je zapříčiněn hlavně ledabylou implementací protokolu s fixací na funkčnost jediného serveru – Microsoft Exchange. Pro vývojáře serveru je to hotová noční můra.

S čerstvě vydanou verzí Mail for Exchange 3.0 konečně přichází možnost výběru e-mailových složek, které se mají synchronizovat. Kalendář může být jen jeden, je ale možné synchronizovat úkoly. Kromě synchronizovaných kontaktů je možné vyhledávat v globálním adresáři na serveru a také nastavit automatickou odpověď v nepřítomnosti.

Palm

Nový operační systém WebOS (nyní ve verzi 1.4.1) přinesl na Palm konečně spolehlivou a stabilní verzi klienta pro Exchange ActiveSync. Chybí pouze synchronizace úkolů. Vcelku bezproblémový a velmi dobře kompatibilní klient.

Android

Až do verze 2.1 byl Android omezen pouze na podporu IMAPu a POP3 nebo synchronizaci s Gmail účtem. Výrobci mobilních telefonů v čele s HTC tento nedostatek vyřešili po svém a chybějícího klienta pro ActiveSync synchronizaci

dopsali sami. Spolu s tím nahradili i výchozí aplikace pro e-maily a kalendář. Příkladem je HTC Hero nebo HTC Magic s Android 1.5 a 1.6.

Od verze 2.1 již Android umí synchronizovat poštu a kontakty s ActiveSync serverem. Chybějící synchronizace kalendáře (umí jen Google Calendar) je opět řešena v režii výrobce telefonu. Příkladem může být Motorola Droid nebo Motorola Milestone. Nevýhodou tohoto stavu je fakt, že synchronizace e-mailů a kalendáře probíhá nezávisle dvěma různými aplikacemi. Ani přidané aplikace zatím neumějí synchronizovat úkoly.

Závěr

Synchronizace groupwarových dat s klienty představuje pro vývojáře sadu zajímavých problémů k řešení. Ať už se jedná o snahu přenášet data co nejefektivněji nebo zajistit co nejlepší kompatibilitu s existujícími klienty. V konečném důsledku se bude vždy jednat o kompromis mezi rychlostí a objemem přenášených dat. Naopak synchronizace mobilních telefonů už dávno není jen příjemnou funkcí a stala se již základní vlastností moderních groupwarových řešení.

Literatura

[1] *Exchange Server Protocol Document.* http://msdn.microsoft.com/en-us/library/cc425499(EXCHG.80).aspx

[2] Wikipedia – SyncML. http://en.wikipedia.org/wiki/SyncML

[3] *The Calendaring and Scheduling Consortium.* http://www.calconnect.org/

[4] *WAP Binary XML Content Format.* http://www.w3.org/TR/wbxml/

Trendy v oblasti čipových platebních karet

Martin Chlumský

E-mail: martin.chlumsky@hp.com

1 Posun v zabezpečení proti kopírování karet

Pokud platba čipovou kartou probíhá online, má autorizační centrum banky možnost zkontrolovat tzv. aplikační kryptogram, což je forma zabezpečení transakčních informací algoritmem 3DES MAC (Message Authentication Code) a klíčem sdíleným mezi kartou a autorizačním centrem vydavatele. Čipové karty ovšem umožňují také provádění offline transakcí, kdy je celý průběh platby z důvodu urychlení uskutečněn bez spojení s autorizačním centrem. Kryptogram s nezbytnými informacemi je pro potřeby vyúčtování odeslán do banky později, např. ve večerních hodinách. V takovém případě musí být k dispozici alternativní metoda, která dovolí terminálu již v průběhu transakce nějakým způsobem zjistit, zda na kartě byly provedeny neoprávněné změny nebo zda jde o falešnou kartu. Na základě výsledku takové kontroly může terminál zabránit neoprávněnému schválení transakce.

Standard EMV definuje tři různé metody offline ověření karty. Nejjednodušší je statická autentizace (SDA – Static Data Authentication), kde je kontrolována případná modifikace vybraných datových objektů karty. Pokročilejší dynamická autentizace (DDA – Dynamic Data Authentication) umožňuje zjistit, zda jde o duplikát, nejvyspělejší metoda Combined DDA and Application Cryptogram Generation (CDA) pak navíc dovolí terminálu ověřit, že rozhodnutí karty o schválení transakce není podvrženo útočníkem.

- SDA metoda je založena na principu verifikace podpisu vybraných statických dat karty. Vydavatel vlastní certifikát podepsaný certifikační autoritou platební asociace. Při přípravě dat jsou důležitá data karty podepsána privátním klíčem vydavatele (banky), přičemž podpis dat a příslušný certifikát vydavatele jsou uloženy na kartu. Každý terminál obsahuje sadu veřejných klíčů asociací s různými délkami. Terminál nejprve zkontroluje vhodným klíčem certifikát vydavatele na kartě a posléze získaným klíčem vydavatele i digitální podpis načtených dat. Tak získá informaci o tom, zda nebyla důležitá data karty od její výroby pozměněna. Karta v procesu SDA nijak neparticipuje, proto je možné pro tento druh karet využít levné

čipy bez podpory RSA. Není žádným tajemstvím, že z tohoto prozaického důvodu byly SDA karty doposud nejrozšířenější.

Takovou kartu lze ovšem snadno zkopírovat včetně platného digitálního podpisu. Terminál pak není schopen nijak zjistit, že jde o kopii. Jelikož nelze z karty kopírovat žádné klíče, tj. offline PIN ani klíče pro generování kryptogramů, nelze takovou kartu použít pro online platby, kde by autorizační systém zjistil nesoulad v hodnotě aplikačního kryptogramu. Při vhodném výběru obchodu může ale útočník provádět s duplikátem menší offline platby. Hodnota offline PIN je na takové kartě pochopitelně nastavena podle vlastních preferencí podvodníka.

- DDA poskytuje dodatečnou záruku, že karta nebyla zkopírována. K tomu je ale potřeba, aby každá karta byla vybavena svým unikátním RSA párem a schopností podepisovat data. DDA karty tedy nevystačí s 3DES technologií a vyžadují dražší čipy s podporou asymetrické kryptografie. Poznamenejme, že tyto karty navíc nabízejí šifrovaný způsob verifikace PIN mezi terminálem a kartou.

Během procesu DDA terminál prověří, zda důležitá data karty nebyla od personalizace změněna a zda je karta originálem, tj. má-li svůj privátní klíč s patřičným certifikátem podepsaným vydavatelem. Ověření autenticity dat karty probíhá podobně jako u SDA pouze s tím rozdílem, že statická data jsou verifikována současně s certifikátem karty – hash důležitých dat karty je součástí certifikátu karty. Terminál navíc požádá kartu o vygenerování digitálního podpisu vybraných dat, který je následně terminálem ověřen prostřednictvím dříve zkontrolovaného certifikátu karty. Procesu Dynamic Data Authentication se v tomto případě účastní aktivně karta i terminál.

Přestože je terminál schopen zjistit, zda je karta originálem nebo duplikátem, nemá žádnou garanci, že vygenerovaný aplikační kryptogram s rozhodnutím o schválení či zamítnutí transakce byl vygenerován skutečně kartou a nikoliv útočníkem, který může vstoupit do komunikace mezi kartou a terminálem. Obdobně jako u SDA platí, že terminál nemá žádné prostředky k verifikaci aplikačního kryptogramu, neboť verifikace kryptogramu může být provedena pouze v autorizačním centru.

- CDA je vylepšenou verzí DDA, přičemž je obvykle využit stejný typ karet jako u metody DDA odlišující se pouze nastavením. Tato metoda zahrnuje do podpisu generovaného kartou i spočtený aplikační kryptogram, čímž terminál získává garanci, že aplikační kryptogram (MAC transakčních dat) s patřičným rozhodnutím o schválení byl vygenerován skutečně kartou. Proces verifikace je velmi podobný DDA, je pouze přesunut v průběhu transakce až na dobu generování aplikačního kryptogramu.

CDA je nejvyspělejší metodou zabezpečení statických dat i dynamických dat generovaných kartou. Nevýhodou tedy může být pouze výpočetní náročnost,

která je spojena s asymetrickou kryptografií. Vzhledem k tomu, že u CDA je digitální podpis a jeho verifikace spojena s generováním kryptogramu, je možné, že bude tato časově náročná operace provedena v jedné transakci dokonce dvakrát (první kryptogram spojený s požadavkem o online zpracování a druhý kryptogram obsahující finální rozhodnutí karty). To může mít teoreticky dopad na prodloužení doby trvání transakcí.

Doposud nebyl asociacemi činěn žádný tlak na používání některé z výše uvedených metod. Banky nejčastěji inklinovaly k levnému řešení SDA karet, které pochopitelně oproti nečipovým kartám přineslo významné zvýšení bezpečnosti. Od roku 2011 již bude ale povinné vydávat nové karty bez podpory SDA. Jelikož jsou produkty umožňující provedení DDA nebo CDA shodné, zdálo by se, že banky sáhnou rovnou po nejvyspělejší metodě. Realita může být ovšem překvapivě jiná. Ukazuje se, že malý podíl CDA karet i terminálů v současném platebním prostředí přináší množství chyb v implementacích, zejména na straně terminálů. Významným kritériem pro rozhodování vydavatele karet může být i rychlost transakce. Jestliže v ověření DDA najdeme jednu verifikaci podpisu karty, u CDA transakce může terminál ověřovat i dva podpisy. To může negativně ovlivnit rychlost transakce na starých terminálech, na moderních platformách je zpoždění z pohledu průběhu celé platby neznatelné.

Dalším zajímavým faktorem je zvolená délka RSA párů karty. Pro délky klíčů vydavatele jsou sdružením EMV pravidelně vydávána závazná pravidla. V současné době se běžně používají klíče s délkou modulu 1 152 nebo lépe 1 408 bitů. Klíče karet mohou mít délku maximálně shodnou s klíčem vydavatele. Jelikož současné karty disponují dostatečným výkonem, bylo by tedy možné vydávat karty s klíči 1 408 bitů. Ovšem ani tady není situace příliš růžová. Délka klíče se obvykle odráží na délce zabezpečených dat přenášených mezi kartou a terminálem. K reprezentaci délky dat je používáno BER kódování, takže pro klíče s délkou od 1 024 bitů (128 B) je potřeba dvou bytů. Mnoho terminálů ovšem očekává délku pouze v jediném bytu, což způsobuje vážné problémy. Asociace tedy doporučují používat klíče kratší než 1 024 bitů, např. 896 bitů.

Hlavní příčinou těchto problémů je nedůsledné testování výrobců, ale zejména i nedostatečné testy asociací při provádění certifikací. Je až zarážející, že za těchto okolností tlačí asociace vydavatele karet k migraci na novější technologie, přičemž je zřetelné, že banky nemohou za investované peníze získat to nejlepší možné zabezpečení.

Využití karet k autentizaci držitele

Asociace MasteCard přišla s programem CAP (Chip Authentication Program), který by měl přispět k zabezpečení e-banking, e-commerce a telebanking transakcí prostřednictvím vícefaktorové autentizace založené na současné čipové tech-

nologii EMV. Jednoduché kapesní zařízení (PCR – Personal Card Reader) vygeneruje po úspěšné verifikaci PIN jednorázový kód (token), který může být použit k zabezpečení elektronické transakce nebo ověření držitele různými kanály (Internet, telefon). Asociace Visa adoptovala tuto specifikaci pod názvem DPA – Dynamic Passcode Algorithm.

Celý koncept je založen na dvojici základních schopností EMV karet: ověřit držitele prostřednictvím offline PIN a generovat aplikační kryptogram. Aplikační kryptogram je hodnota MAC (Message Authentication Code), která zabezpečuje 3DES klíčem sdíleným mezi kartou a bankou základní transakční informace: pořadové číslo transakce, částku, měnu transakce, náhodné číslo generované terminálem, rozhodnutí karty o schválení či zamítnutí transakce a příznaky reprezentující její interní stav. Nejjednodušší příklad použití CAP může vypadat následovně: po zasunutí karty do kapesní čtečky je držitelem zvolena požadovaná operace. Následně je uživatel vyzván k zadání PIN. Po úspěšné verifikaci PIN je kartou vygenerován kryptogram, který je společně s některými informacemi vhodnou formou prezentován na displeji kapesního zařízení. Tato zobrazená hodnota představuje autentizační token, který je držitelem přepsán do formuláře internetového bankovnictví.

Specifikace CAP definuje několik módů: jednorázové heslo, výzva–odpověď, podpis nebo volitelně i podpis transakčních dat. Z pohledu karty jde v principu vždy o offline transakci. Vstupy do výpočtu aplikačního krytpogramu jsou v závislosti na zvoleném módu vloženy uživatelem na klávesnici kapesního zařízení nebo jsou jednoduše nahrazeny nulami. Například pro režim vygenerování jednorázového hesla není kromě PIN uživatel nucen zadávat žádné další informace, v režimu výzva–odpověď pak uživatel vkládá také výzvu zaslanou bankou a tato hodnota je ve vstupních datech algoritmu aplikačního kryptogramu použita jako náhodné číslo generované terminálem.

Specifikace CAP byla koncipována tak, aby bylo možné použít již dříve vydané platební karty. Generování autentizačních tokenů má ale v takovém případě dopady na risk management karty, protože ovlivňuje její interní čítače offline plateb. To může v důsledku znamenat, že karta nemusí být schopna po několika vygenerovaných tokenech provést skutečnou transakci, pokud se např. terminál v obchodě nemůže spojit s autorizačním centrem a interní čítač počtu po sobě jdoucích offline transakcí je již překročen. K nápravě dojde až při transakci, kdy je karta v kontaktu s autorizačním centrem a kdy jsou opět resetovány interní čítače karty. Jelikož je navíc sdílení klíčů platební aplikace se systémem pro autentizaci v rozporu se základními bezpečnostními praktikami, je vhodnější na platební kartu personalizovat oddělenou CAP aplikaci s vlastním klíčem.

V běžné platební transakci je verifikace aplikačního kryptogramu celkem jednoduchou záležitostí. Zpráva zasílaná z terminálu do autorizačního centra obsahuje základní transakční data (částka, datum, příznaky terminálu a karty, čítač transakcí karty apod.) a vlastní kryptogram zabezpečující tato data. Systém si

odvodí na základě čísla karty, popř. i čítače transakcí, jedinečný klíč, pomocí něhož spočte vlastní hodnotu kryptogramu nad obdrženými transakčními informacemi. Ta je srovnána s hodnotou zaslanou v autorizační zprávě. To ovšem nelze realizovat v transakci CAP. Pokud by měl uživatel přepisovat autentizační token, který by v sobě přenášel veškeré potřebné informace, byl by natolik dlouhý, že by při jeho přepisování nebo diktování po telefonu s velikou pravděpodobností docházelo k chybám. Naštěstí lze část informací na straně autentizačního serveru v bance získat jiným způsobem nebo predikovat. Vydavatel zná verzi použitých algoritmů nebo klíčů na vydané kartě, zná hodnotu prováděné transakce apod. Aby byla délka tokenu pro uživatele přijatelná, je jeho obsahem pouze část aplikačního kryptogramu společně s údaji, které nelze jinou cestou získat. Výběr těchto informací při sestavování tokenu je na straně čtečky řízen bitovou mapou IPB (jakýsi filtr). Ta je uložena na kartě a je optimalizována vydavatelem, který přesně zná chování svých karet. Pokud tato mapa na kartě chybí, použije terminál default hodnotu. Ta ovšem postrádá optimalizaci a přenáší tak větší množství informací. Tokeny pak mají obvykle délku větší než dvanáct znaků.

Pokud je karta vybavena samostatnou autentizační aplikací, je situace mnohem příznivější. Vnitřní příznaky autentizační aplikace nejsou ovlivňovány platební aplikací, jsou tedy statické a není tedy nutné přenášet jejich stav. Token pak přenáší pouze část aplikačního kryptogramu a interního čítače transakcí. Takové tokeny je možné v závislosti na požadované úrovni bezpečnosti stlačit do sedmi až devíti číslic.

Zejména při využití platební aplikace pro generování autentizačních tokenů je nutné brát v potaz, že ačkoliv se samotný token může zdát dostatečně dlouhý, jeho velkou část tvoří čítač transakcí a interní příznaky karty. Pokud útočník odchytí takový token, může významnou část dalšího tokenu celkem snadno predikovat. Např. využití implicitní hodnoty bitové mapy IPB karet VISA dává útočníkovi i při zdánlivě velké délce tokenu s dvanácti číslicemi šanci 1 ku 65 535, protože z aplikačního kryptogramu je přenášeno pouze 16 bitů a ostatní informace lze celkem snadno odhadnout z dříve zachycených dat (čítač transakcí se inkrementuje a interní příznaky budou patrně stejné). Vezmeme-li v úvahu také fakt, že platební asociace specifikaci CAP veřejně nepublikovaly a nebyla tedy nijak prověřena širší skupinou bezpečnostních expertů, je poněkud diskutabilní, zda může takové řešení skutečně dosáhnout požadované úrovně bezpečnosti. Přesto se řady i poměrně kritických odborníků shodují na tom, že proti statickým heslům jde o výrazný krok kupředu.

Bezkontaktní platby

MasterCard PayPass a Visa payWave, to jsou loga, která během tohoto nebo příštího roku začneme i u nás vídat na místech, kde jsme doposud platili běžnými

kartami. Česká republika je jedním z ostrůvků pomyslné mapy vyspělejší části světa, kde tato technologie doposud nebyla implementována a spuštěna alespoň v pilotním provozu.

Bezkontaktní transakce vycházejí převážně z EMV. Jelikož ale existuje velká část trhu, která tento standard doposud neakceptovala, existuje i varianta MSD (Magnetic Stripe Data), která je vhodná pro prostředí spoléhající na karty s magnetickým proužkem (např. USA).

Hlavní výhodou bezkontaktních transakcí má být pohodlí a rychlost. Tomu je podřízeno vše – bezkontaktní platby v našem regionu budou probíhat převážně offline. To znamená, že karta nebude čekat na schválení získané online komunikací s bankou. Navíc nebude pro tyto platby vyžadována žádná verifikace držitele, tj. PIN nebo podpis. Je tedy patrné, že půjde především o platby menších částek v řádu několika set korun. Bezkontaktní platby ovšem nejsou omezeny pouze na offline režim, mohou být provedeny i online, přičemž je možné spoléhat také kontrolu online PIN nebo podpisu. Příslušné nastavení není závislé pouze na rozhodnutí banky, ale je regulováno pro každý region platebními asociacemi.

Online komunikace karty bude vyžadována tehdy, pokud bude částka přesahovat stanovený limit, nebo se karta sama rozhodne, že již bude lepší kontaktovat banku. Online platba může proběhnout v bezkontaktním režimu nebo bude držitel vyzván, aby kartu zasunul do standardní kontaktní čtečky, kde proběhne běžná online EMV transakce. Pouze kontaktní transakce ale umožní, aby došlo k vynulování interních čítačů a karta byla opět nachystána k provádění dalších bezkontaktních offline plateb na místech, kde připojení není možné.

Zajímavostí je, že bezkontaktní transakce nemusejí být prováděny nezbytně kartou. Asociace představily i jiné formáty: přívěšky na klíče, hodinky, mobilní zařízení apod. V nejbližší době se zřejmě setkáme pouze s běžným formátem karet vybavených duálním rozhraním.

Bezkontaktní technologie ovšem může vyvolat i obavy klientů. Je taková technologie bezpečná? Komunikuje karta skutečně na vzdálenost do 10 cm nebo je možné informace z karty získat i na větší vzdálenost. Lze pak takovou kartu zkopírovat, aniž by držitel něco zjistil? Nebude tedy lepší vlastnit starou, vyzkoušenou kontaktní kartu? Z tohoto důvodu nelze vyloučit, že bude mít klient banky i nadále možnost sáhnout po kartě vybavené pouze kontaktním rozhraním.

Nedostatky EMV řešení

Univerzitní týmy v poslední době několikrát poukázaly na slabiny řešení postavených na standardech EMV. Ve zkratce si představíme tři nejznámější studie.

Autentizace CAP

V předchozí části jsme si stručně popsali využití EMV karet ke generování autentizačních tokenů. Vědci se bez znalosti specifikace CAP vrhli do analýzy tohoto

řešení a poukázali na několik více či méně závažných nedostatků. Některé z nich jsou zajímavé pouze pro paranoidní jedince, z některých ovšem plyne, že při nevhodné implementaci může být zamýšlená úroveň bezpečnosti poněkud degradována. Uveďme si několik odlišných typů nedostatků, aby bylo patrné, z jak rozdílných pohledů lze koncept posuzovat.

Specifikace CAP definuje několik různých módů – jednorázové heslo, challenge-response, podpis dat apod. Z pohledu karty jde ale vždy o pouhé vygenerování aplikačního kryptogramu. Jelikož vybraný režim není nijak zakomponován do vstupních dat, nelze zpětně zjistit, k jakému účelu byl token vygenerován. Ve vstupních polích pro výpočet kryptogramu jako jsou částka, náhodné číslo, popř. měna, jsou buď nuly, nebo potřebná informace. Útočník tedy může požádat přítele, aby vygeneroval token v režimu výzva–odpověď s výzvou ze samých nul. To přítel patrně nebude považovat za problém, protože banka by takovou výzvu určitě nikdy neposlala. S tímto kódem se ovšem útočník můžete jít zákeřně přihlásit na účet důvěřivce, jelikož aktuální platné jednorázové heslo by bylo spočteno zcela shodným způsobem (náhodné číslo – výzva – je v takovém případě vyplněno nulami).

Řešení rovněž nijak nezamezí některým útokům Man-in-the-middle. Pokud útočník vstoupí do komunikace mezi uživatelem a bankou, může zadržet platný token, nahlásit uživateli chybu aplikace a poté se s tokenem přihlásit k bankovní aplikaci místo uživatele. Za nedostatek je považována i absence časového omezení platnosti tokenu.

Bylo rovněž poukázáno na to, že na bankomatu nebo platebním terminálu je s ohledem na četné používání mnoha lidmi jen těžko zjistitelné z otisků na klávesnici, jaký PIN máte. Pokud vám ovšem někdo zcizí kartu včetně čtečky, budou s nejvyšší pravděpodobností omačkané především ty klávesy, které používáte pro zadávání PIN. To může zvýšit šanci na uhádnutí PIN během tří běžně dostupných pokusů na 3/24. Doposud také neměl zloděj karet k dispozici žádné běžně dostupné zařízení, které by jasně prokázalo, zda je hodnota PIN správná či nikoliv. CAP čtečka, která generuje token pouze za předpokladu, že vložíte správný PIN, umožňuje útočníkovi ověřit, že hodnota PIN, kterou z vás násilím získal, je platná, a to bez rizika zkoušení na bankomatu pod dohledem kamer. Za zmínku stojí, že použití násilí k získání PIN není zcela ojedinělou záležitostí.

YES-Cards

YES cards – to je název pro upravené duplikáty SDA karet. Tým univerzitních pracovníků poukázal na možnost zhotovení duplikátů SDA karet, jejichž nastavení definované bankou preferuje offline platby a verifikaci offline PIN. Toto nastavení je typické pro většinu současných čipových karet. Falešné karty pochopitelně nemají správné klíče pro generování aplikačních kryptogramů, mohou mít ovšem vlastní hodnotu offline PIN a dostatečně nastavené limity pro provádění

offline plateb. K provedení podvodné transakce pak stačí najít vhodný obchod s podporou offline plateb. SDA karty obsahují podpis vybraných dat, který je společně s ostatními datovými objekty správně zkopírován na falešné karty. Terminál tedy nezjistí žádnou nesrovnalost a nemá z tohoto pohledu žádný důvod posílat transakci k online ověření. Pokud nákup nepřesáhne jeho limity pro provedení offline transakce a verifikace offline PIN proběhne úspěšně (vlastní hodnotu PIN není těžké zadat ;-), požádá terminál kartu obvykle o schválení. A karta odpoví ANO. Kartou spočtený aplikační kryptogram sice není správný, ale to terminál nemá možnost zjistit. Tento stav může odhalit banka až ve chvíli, kdy offline transakce dorazí do jejího centra (např. během noci). Tato platba již ale byla schválená a jediné, co může banka učinit, je případně zakázat tuto kartu. Zaslání online skriptu na kartu s příkazem zablokovat aplikaci ovšem také nijak nepomůže, protože falešná karta nemá nikdy snahu komunikovat s autorizačním centrem. Navíc by takový příkaz velmi pravděpodobně ignorovala. Pak je tedy nutné dát kartu na jakousi černou listinu, aby terminály číslo této karty neakceptovaly. Poznamenejme, že banky v některých případech offline kryptogram již schválené transakce dokonce ani nemohou kontrolovat, protože nedostávají vždy dostatek informací k jeho verifikaci.

Chip and PIN is broken

Poslední a asi nejvíce diskutovaný výsledek zvídavých hochů z univerzit poukázal na možnost přesvědčit terminál o tom, že zadaná hodnota PIN je správná a i při kontaktu s autorizačním centrem, tj. při nejbezpečnějším způsobu platby, může dojít ke schválení takové transakce. Útočníci využili toho, že komunikace mezi terminálem a kartou nevyužívá žádné zabezpečení. Požádá-li tedy terminál kartu o verifikaci offline PIN, není těžké do této komunikace zasáhnout, příkaz na kartu vůbec nezaslat a terminálu vrátit odpověď, že verifikace PIN proběhla bez chyby. Příznaky risk managementu karty a terminálu obsahují řadu zajímavých informací, ale obvykle jde spíše indikace chyb, neúspěšného provedení nějaké operace apod. V tomto případě karta nedostala příkaz verifikace PIN, není tedy nastaven žádný příznak, který by indikoval, že zadaná hodnota PIN byla špatně. Není rovněž nastaven příznak provedené verifikace PIN. Jelikož karty často podporují ověření držitele podpisem a existují i terminály, které verifikaci nevyžadují (parkovné apod.), nelze takový stav jednoduše označit za chybný. EMV bohužel nedefinuje žádný mechanizmus, který by umožnil vzájemnou domluvu karty a terminálu na způsobu verifikace držitele a následnou kontrolu, že k tomuto ověření došlo. Přestože lze považovat tyto nedostatky za zásadní slabiny v návrhu, nemusela by situace být tak vážná, pokud by autorizační centrum provádělo dostatečné kontroly před odesláním odpovědi se schválením transakce (karta tvrdí, že neproběhla verifikace PIN, terminál tvrdí, že verifikace PIN proběhla úspěšně). Některé banky ale neprovádějí důslednou kontrolu jednotlivých

příznaků a jejich vzájemné porovnání, druhým faktem je, že provozovatelé POS terminálů neodesílají vždy dostatek informací, které by pro takové testy byly potřebné. Z toho je ovšem není možné zcela vinit, jelikož asociace přenos některých informací nevyžadují. Přestože demonstrace zneužití karty v televizi BBC vypadala bombasticky, není dle mého soudu situace tak závažná. Především jde o karty, které byly odcizeny, přičemž majitelé nezajistili jejich zablokování. Druhý fakt je ten, že na čip je nutné přiletovat kabel vedoucí k notebooku, který vstupuje do komunikace mezi kartou a terminálem. Pro útočníky je tedy i nadále jednodušší provádět podvodné transakce prostřednictvím magnetických proužků, které jsou stále součástí platebních karet.

Zajímavé mohou být úvahy o zneužití těchto slabin v bezkontaktních platbách. Nejpalčivějším problémem této nastupující technologie je možnost vzdáleného čtení obsahu karet. Pak je tedy ale teoreticky možné bez nutnosti krádeže karty vytvořit kopii, pochopitelně bez originálních klíčů a PIN. To by mohlo vést ke snadné výrobě dříve zmíněných YES cards. Naštěstí je éra SDA technologie u konce, a u duplikátů karet s autentizací DDA nebo CDA terminál snadno zjistí, že nejde o originály. Také výše zmíněný problém s verifikací offline PIN není pro bezkontaktní svět významný. Bezkontaktní platby jsou určeny pro transakce s malými částkami, a proto budou obvykle odbaveny bez jakékoliv verifikace držitele. Pokud bude verifikace nezbytná, je k dispozici pouze online PIN nebo podpis držitele. Verifikace online PIN neprobíhá dříve zmiňovanou komunikací mezi terminálem a kartou, nýbrž přímou verifikací zašifrované hodnoty PIN u vydavatele karty (terminál komunikuje s bankou). Útočník do této komunikace nemůže rozumně zasáhnout.

Závěr

Na závěr malé bilancování. Je patrné, že asociace by měly při přípravě specifikací nechat prověřit navrhované řešení širší odbornou veřejností. Udržování specifikací v tajnosti nemusí být dostatečným opatřením k zajištění požadované bezpečnosti. Ukazuje se, že významnou roli hraje i konkrétní implementace. Zejména snaha o přílišné zjednodušování může přinést skryté zranitelnosti systému. Rovněž nedůsledné testování vede výsledně ke zpomalení nasazování bezpečnějších řešení. Přesto je zřetelné, že EMV standard přispěl k bezpečnějším platbám. Pravdou ale zůstává, že dokud budou karty vybaveny magnetickým proužkem, bude pro organizované skupiny nejjednodušší zneužívat tuto historickou technologii. Teprve po odstranění tohoto magického proužku lze očekávat, že se skutečně zvýší podíl a úroveň útoků na čipovou technologii.

Kalendářový server SOGo

Miloš Wimmer

E-mail: wimmer@civ.zcu.cz

Abstrakt

Příspěvek je věnován popisu výběru aplikace kalendářového serveru pro zaměstnance i studenty Západočeské univerzity, jeho vlastnostem i zkušenostem s provozováním. Zvolený svobodný software SOGo nabízí služby kalendářů, úkolů a kontaktů a integruje je s externím poštovním systémem a s adresářovou službou. Společně s nimi tak tvoří groupwarový systém. Umožňuje přístup přes vlastní prostředí webmailu i z externích klientů podporujících standardní protokoly CalDAV nebo SyncML, jakými jsou např. Mozilla Thunderbird, iCAL, PDA, apod.

Období před SOGem

V prostředí počítačové sítě Západočeské univerzity se od jejího vzniku v roce 1990 používala elektronická pošta. Služby poštovního systému umožňující přístup přes standardní protokoly IMAP a POP a přístup přes webové prostředí Horde uživatelům plně vyhovovaly a o nadstavbové služby groupwarových systémů neprojevovali dlouho reálný zájem. Poptávka po nich se objevila ze strany organizačních struktur školy po roce 2000.

S ohledem na podporu vytváření e-learningových kursů, pro možnost provozovat systém v operačním systému Linux i z dalších důvodů byl tehdy vybrán komerční groupwarový systém Lotus Notes. Kvůli licenční politice Lotusu byl groupwarový systém poskytnut omezenému okruhu lidí v řádu několika desítek uživatelů. Až na výjimky se však mezi nimi nesetkal s přívětivým přijetím a to ani mezi těmi, kteří po groupwarových službách volali. Hlavním důvodem zřejmě bylo nezvyklé uživatelské prostředí Lotus Notes a také interní systém elektronické pošty, který byl oddělený od centrálního poštovního systému ZČU.

V konečném důsledku se tak systém Lotus Notes používal mezi vybranými uživateli ZČU jen v malé míře a nenaplnil očekávání, která měl přinést.

Období hledání

Coby správce centrálního poštovního systému ZČU a zastánce svobodného software jsem byl na jaře 2008 požádán, abych se pokusil najít alternativní nekomerční groupwarový systém – v zásadě kalendářový server. Groupwarovou kalendářovou službu považuji za užitečnou pro většinu uživatelů, chápu ji jako součást „messagingových“ služeb a jsem přesvědčen, že by měla být integrována s poštovním systémem, proto jsem souhlasil. Navíc zde byla výzva dokázat, že svobodný software nabízí lepší řešení i na tomto výsostném poli komerčních systémů...

Nejprve jsem si chtěl ujasnit, jaké požadavky má kalendářová služba splňovat. Podmínky nutné:

- bude dostupná pro všechny zaměstnance i studenty (20 000+ kont)
- umožňuje přístup z webového rozhraní i z nativních desktopových klientů
- lze ji integrovat se stávajícím (externím) poštovním systémem
- webové rozhraní integruje poštu i kalendář
- podporuje řízený sdílený přístup
- je v naší správě
- umožňuje přihlášení v duchu systému Orion a integraci s ním. Uživatelé tedy budou používat své existující jméno/heslo, kterým se ověřují u všech služeb v síti ZČU.
- je lokalizovaná nebo lokalizovatelná

Bylo by dobré, kdyby:

- podporovala zpřístupnění rozvrhů studentů a vyučujících generovaných ze studijní agendy
- podporovala plánování sdílení zdrojů (projektor)
- podporovala synchronizaci s PDA
- bylo možné ji integrovat s námi podporovaným poštovním klientem Thunderbird

Doufal jsem, že v této aplikační oblasti bude situace obdobná jako u webmailových systémů, tedy že bude na výběr mezi několika nejčastěji používanými a osvědčenými systémy, které se budou lišit provedením, funkcemi a možnostmi integrace s okolními službami, ale základní kalendářové funkce budou podporovat standardně. To byl hluboký omyl. Díval jsem se na řadu systémů a bezmála deset z těch, které měly papírově splňovat mé požadavky, jsem se pokusil zprovoznit.

Výsledek byl tristní. Chybějící funkčnost, nekomfortní uživatelské rozhraní, problematická instalace, fatální problémy při běhu systému, nespolehlivost, potíže s integrací s okolními službami, špatná podpora kalendářových standardů a s tím související špatná interoperabilita s kalendářovými klienty, to všechno byly převažující průvodní jevy. Po více než měsíci únavné práce jsem byl nucen přiznat, že rozumný alternativní groupwarový systém ze světa svobodného software jsem nenašel.

Po roce jsem byl osloven se stejným požadavkem. Tentokrát jsem už měl zkušenosti z předchozího hledání, takže zjišťování aktuálního stavu kalendářových systémů probíhalo rychleji. Podrobněji jsem se díval na tyto kalendářové systémy:

- Bedework
- Zimbra
- Open-Xchange
- Horde
- OBM
- Oracle Calendar
- Webcalendar
- CalendarServer
- Chandler
- SOGo (OpenGroupware)

a srovnával jsem je s:

- Lotus Notes
- Google Calendar
- MS Exchange

U některých systémů došlo ke zlepšení, některé zůstaly stát beze změny. Pro náš záměr se použitelnými zdály Horde a SOGo. Oba systémy jsem tedy uvedl do testovacího provozu a ve skupině asi 30 lidí jsme je 3 měsíce používali.

Horde

Systém Horde je webová aplikace vyvíjená v PHP pod GNU licencí. Obsahuje řadu modulů, které lze kombinovat. Na našem produkčním webmailu používáme moduly Horde, IMP a Turba pro prostředí klasického webmailu. Přidání modulu Kronolith, který systém rozšiřuje o kalendářovou službu, by tedy pro nás znamenal evoluční krok, protože celé prostředí webmailu by zůstalo stejné, rozšířila by se jen jeho funkčnost. Pro ukládání dat se používá databáze MySQL.

Obecnou nevýhodu Horde vidím v tom, že je díky PHP těžkopádnější, pomalejší a nelze dobře škálovat. Jeho prostředí nevyužívá moderní dynamické webové technologie jako např. AJAX, takže je pro uživatele méně komfortní. V úvahu je třeba vzít i jeho hodně pomalý vývoj (v posledním roce téměř stagnoval) a pomalé reakce vývojářů.

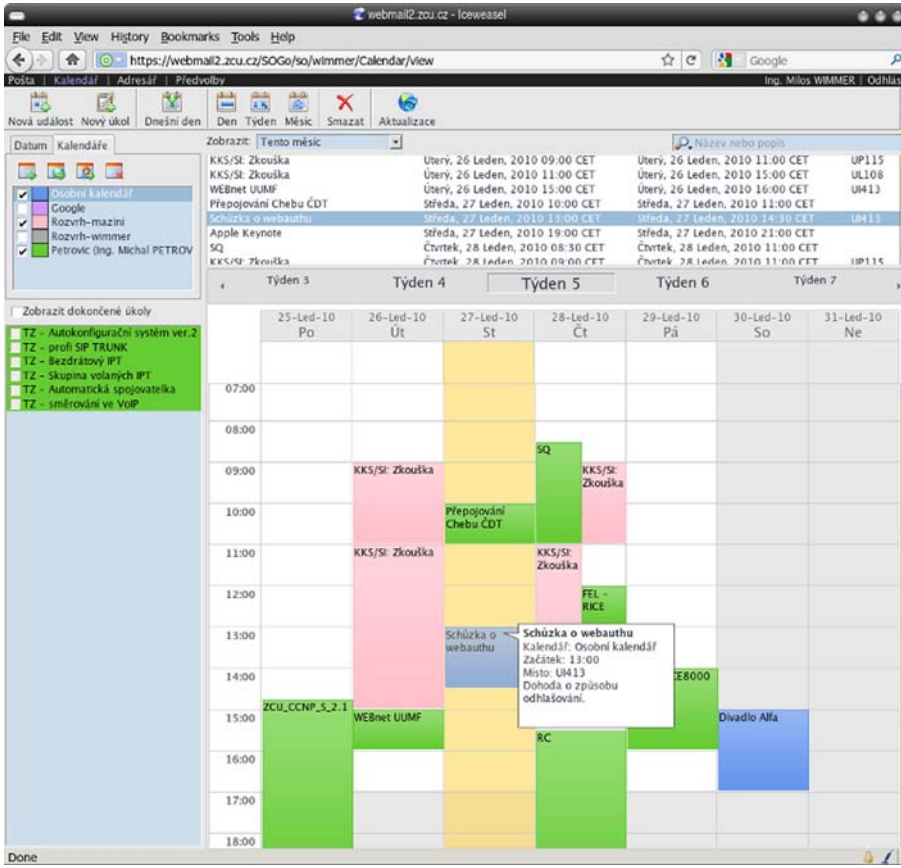
Díky otevřenosti PHP kódu a připravené koncepci tzv. hooků lze do prostředí snadno vložit automatické připojování externě dostupných kalendářů, což jsem s výhodou využil pro vložení kalendáře s osobním rozvrhem každého studenta a vyučujícího generovaným serverem studijní agendy.

Během testovacího provozu jsme ověřili, že základní kalendářové služby Horde jsou při práci ve webovém prostředí funkční a koncepčně jsou dobře integrované s prostředím webmailu. Podpora externích klientů je problematická. Horde nemá žádného „svého“ externího kalendářového klienta. Pro přístup ke kalendářové službě z prostředí desktopu jsme proto používali klienta Mozilla Thunderbird s rozšířením Lightning. Kalendáře se musely definovat ručně v podobě ICS souborů dostupných na vzdáleném HTTP serveru. Synchronizace kontaktů nebyla možná. Některé funkce jsou z externího klienta nedostupné, což je citelné zejména u free/busy informací potřebných pro plánování schůzek s jinými lidmi anebo při delegování přístupových práv jiným uživatelům. Problematická je i synchronizace s PDA. Na nich jsme používali SyncML klienty Funambol a Synthesis.

SOGo (Scalable OpenGroupware.org)

Systém SOGo je založen na projektu OpenGroupware.org. Využívá prostředí GNUstep a webový aplikační framework SOPE. Je napsaný v Objective-C a vyvíjen pod GNU licencí. Poskytuje službu webového prostředí pro poštu, kalendář a kontakty a současně vzdálený přístup ke kalendářové službě. Pro ukládání dat se nejčastěji používá databáze PostgreSQL.

SOGo běží v systému jako binární program, který si spouští určený počet instancí. Je rychlý, byl navržen i pro velké instalace a jeho koncepce dovoluje velmi dobrou škálovatelnost. Klienti k němu nepřistupují přímo, ale prostřednictvím libovolného HTTP serveru. Webové prostředí SOGo používá moderní dynamické technologie (AJAX) a je velmi komfortní. Koncepce prostředí kopíruje prostředí klienta Mozilla Thunderbird. Opticky i způsobem ovládání jsou obě prostředí



Obr. 1 Ukázka webového prostředí kalendářového serveru SOGo

téměř shodná, což považujeme za výhodu, protože uživatelé Thunderbirda pracují i při webovém přístupu k poště ve známém prostředí.

SOGo je aktivní projekt, který se dynamicky rozvíjí. Vývojáři aktivně spolupracují s komunitou uživatelů, jejich reakce jsou většinou velmi rychlé. Komunita má přístup k vývojovému stromu, takže lze používat nejčerstvější zdrojové kódy obsahující opravené chyby i nové funkce.

SOGo podporuje kalendářové standardy a externím klientům umožňuje přístup přes protokoly CalDAV a CardDAV. Pro plnohodnotný přístup ke všem službám a funkcím kalendářového serveru z prostředí poštovního klienta Thunderbird udržuje vývojový tým upravenou verzi rozšíření Lightning-Inverse Edition spolu s dvojicí dalších rozšíření.

Během testovacího provozu jsme ověřili, že základní kalendářové služby SOGo jsou při práci z webového prostředí i z externích klientů funkční a koncepčně jsou dobře integrované s elektronickou poštou. Podpora externích klientů je vyhovující. PDA zařízení s podporou CalDAV komunikují se SOGo serverem přímo, ostatní se mohou synchronizovat přes protokol SyncML. Pro SyncML synchronizaci jsme použili aplikační server Funambol, který je se SOGem propojen jeho vlastním konektorem. Na PDA jsme používali volně dostupné klienty Funambol.

Při provozu jsme objevili také nějaké chyby, ty významnější vývojáři rychle odstraňovali.

Po vyhodnocení zkušeností a zvážení všech aspektů jsme se rozhodli nasadit kalendářový server SOGo jako centrální službu pro všechny zaměstnance a studenty ZČU a to nejprve do širšího testovacího režimu.

Období se SOGem

Kalendářový server SOGo je vyvíjen pod GNU licencí, je určen pro operační systém Linux, lze ho provozovat bezplatně a bez potřeby licencí pro jednotlivé uživatele. K dispozici je ve formě zdrojových textů i balíčků pro nejrozšířenější linuxové distribuce.

Popis služeb SOGo serveru

Kalendářový server nabízí tyto základní služby:

- hlavní osobní kalendář uživatele
- libovolné množství jeho dalších osobních kalendářů
- přístup ke kalendářům jiných uživatelů, kteří k nim nastavili příslušná práva
- read-only přístup ke vzdáleným kalendářům – ICS souborům dostupných přes vzdálené HTTP servery jako např. rozvrh, státní svátky, Google kalendáře, ...
- osobní seznam kontaktů i veřejný seznam kontaktů, což jsou kontakty poskytované LDAP serverem
- seznam úkolů
- přidělování přístupových práv k osobním kalendářům pro jiné uživatele serveru
- funkce free/busy dostupná ověřenému uživateli kalendářového serveru

- pozvánky na události jsou automaticky zapisovány do osobních kalendářů pozvaných účastníků a navíc je jim elektronickou poštou zasílána zpráva obsahující ICS soubor s danou událostí
- import a export událostí i kontaktů
- přístup k těmto zdrojům je možný lokálně přes www rozhraní i vzdáleně přes standardní přístupové protokoly CalDAV a SyncML

Návaznost na jiné služby

Všechny události, úkoly a osobní nastavení jsou na kalendářovém serveru ukládány do SQL databáze. Základním databázovým serverem je PostgreSQL, alternativně lze použít i MySQL nebo Oracle. Účet uživatele na SOGo serveru vzniká automaticky až při jeho prvním přihlášení.

Webové rozhraní serveru integruje kalendářovou službu s poštovním účtem přihlášeného uživatele, k němuž se připojuje přes standardní IMAP protokol. Poštovní server tak může běžet na nezávislém stroji.

Pro plánování schůzek i adresování zpráv využívá SOGo tzv. Veřejné kontakty, což jsou kontakty poskytované LDAP serverem. Další nezbytnou komponentou je tedy jakýkoliv standardní LDAP server.

Zprávy elektronické pošty odesílané z webového prostředí mohou být předávány lokálně běžícímu SMTP serveru anebo na libovolný vzdálený SMTP server.

Obr. 2 Vazba SOGo serveru na potřebné okolní služby

Pro možnost synchronizace externích klientů nepodporujících protokol CalDAV bych seznam potřebných okolních služeb rozšířil ještě o Funambol server, který funguje jako převodník nebo proxy server mezi SyncML klienty a SOGo serverem. Funambol server je nezávislý aplikační SyncML server vyvíjený pod GNU licencí v Javě. Do něj lze vložit komponentu konektoru pro spojení se SOGo serverem.

Jak jsem uváděl, klienti přistupují k běžícímu démonu SOGo serveru přes lokálně běžící HTTP server. Na jeho pozici lze s výhodou použít Nginx nebo Apache.

Pro ověření přistupujícího uživatele se používají služby LDAP, C.A.S. nebo SQL. V prostředí sítě ZČU používáme technologii jednotného ověření Single Sign-on (SSO) WebAuth. Tato služba je podporovaná modulem do HTTP serveru Apache a proto jsem použil právě tento server. S vývojáři SOGa jsme se domluvili na doplnění kódu, který akceptuje ověření WebAuthu a tím jsme mohli SOGo server zařadit mezi ostatní naše servery používající SSO.

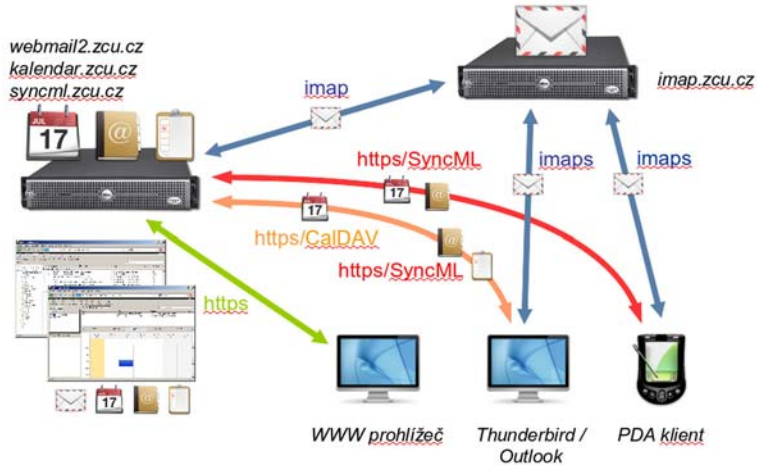
Topologie groupwarové služby

Celá topologie groupwarové služby v prostředí sítě ZČU je tvořena těmito servery:

- *předřazený antispamový/antivirový poštovní server*
  - operační systém Linux Debian-amd64
  - postfix, postgrey
  - MailScanner, clamav, spamassassin, ...
- *centrální poštovní server*
  - operační systém Linux Debian-amd64
  - IMAP a POP server dovecot
  - postfix
  - doručování do uživatelských složek formátu Maildir na souborovém systému xfs
- *kalendářový + webmail server*
  - operační systém Linux Debian-amd64
  - apache+webauth modul, postgresql
  - sogo, funambol
  - postfix
- LDAP server

Do webového prostředí webmail/kalendářového serveru přistupují uživatelé prostřednictvím WWW klientů protokolem HTTPS. Z prostředí nativních klientů přistupují k poštovnímu účtu protokoly IMAP nebo POP a ke kalendářovému serveru protokoly CalDAV (např. Thunderbird + Lightning, Apple iCAL) nebo SyncML (MS Outlook, PDA).

Webové prostředí SOGo podporuje různé jazykové mutace. S původní českou lokalizací jsme nebyli úplně spokojeni, proto jsem ji přepracoval, aby se co nejvíce shodovala s českou lokalizací Thunderbirdu. Tyto úpravy jsem samozřejmě vrátil zpět do projektu.



Obr. 3 Přístupy klientů

Zkušenosti s reálným provozem

SOGo server nyní běží v testovacím provozu, kdy se k němu může přihlásit každý uživatel počítačové sítě ZČU. Na serveru má vytvořený účet zhruba 1 200 uživatelů, denně k němu přistupuje průměrně 500 uživatelů.

Aplikační software má překvapivě malé nároky na systémové zdroje a server (2× CPU E5430 @ 2.66 GHz) příliš nezatěžuje. S vlastním během aplikace nemáme problémy, je stabilní.

Podstatné je přijetí služby ze strany uživatelů. To je zatím velmi dobré. Uživatelům se líbí komfortní dynamické webové prostředí v duchu Thunderbirdu a oceňují i jeho rychlé odezvy. Stejně tak příznivé jsou ohlasy uživatelů, kteří s kalendářovou službou pracují v prostředí Thunderbird/Lightning.

SOGo je relativně mladý projekt, který se opravdu dynamicky vyvíjí. S rozšiřováním jeho funkcí a souvisejícími úpravami kódu občas dojde k tomu, že se objeví chyba. V případě citelnějších chyb není problém vrátit se zpět k předchozí verzi (v provozu je to otázkou asi 3 sekundového výpadku služby). Chyby se hlásí vývojářům přes jejich bug tracking systém a jejich reakce považuji za uspokojivé.

Opensource jako zaměstnání

Vývojový tým tvoří několik málo lidí z Montrealu. Na vývoji SOGa a dalšího známého síťového programu PacketFence pracují ve své firmě Inverse Inc. na plný úvazek. Nabízejí službu placené podpory pro svůj software a umožňují také sponzorovat vývoj. Toho jsme využili když jsme zaplatili vývoj funkcí a úprav,

které jsme v serveru postrádali a které se nám hodí v našem prostředí - jako např. podporu vzdálených kalendářů, automatické vložení odkazu na kalendář s rozvrhem do kalendářů každého uživatele a řadu dalších.

Vývojáři mají svůj strategický plán budoucích prací, ale kdokoliv si může objednat vytvoření nové funkcionality nebo upřednostnění realizace určité práce finanční úhradou. I tyto placené úpravy se stávají součástí celého programového balíku, takže jsou k dispozici ostatním.

Vývojáři samozřejmě přijímají i úpravy a návrhy na zlepšení a na další směřování vývoje od členů komunity.

Monolit nebo skládanka

Je zřejmé, že popsaný groupwarový systém je tvořen složením několika na sobě nezávislých komponent, které jsou i nezávisle vyvíjeny. Jejich vzájemná kooperace je založena na prostém faktu dodržování standardů. Tato koncepce je v protikladu s monolitickými systémy.

Jakkoliv nemám v úmyslu zpochybňovat výhody monolitických řešení, jsem příznivcem těch složených. Výhodu vidím v možnosti rozvíjet systém evolučně, po menších krocích a bez rizika, že úzké místo jedné části systému bude paralyzovat celý systém. A současně, v případě objevení úzkého místa během provozu, mám možnost ho eliminovat nasazením jiné, vhodnější komponenty, aniž by to zásadně ovlivnilo ostatní části celého systému.

V tomto konkrétním případu bych měl např. obavy nahradit centrální poštovní server poštovním systémem, který by byl nedílnou součástí uceleného groupwarového serveru. Náš systém s více než 20 tisíci aktivních kont si bez větších problémů poradí i s doručením hromadné zprávy do několika tisíců kont současně během několika minut a to za plného provozu při přihlášení více než tisíce uživatelů.

V případě nasazení systému pro menší a obvyklé instalace je toto riziko samozřejmě menší.

Odkazy

[1] http://www.scalableogo.org

[2] http://www.funambol.com

[3] http://www.horde.org

[4] http://www.mozillamessaging.com

[5] http://thunderbird.mozilla.cz

[6] http://support.zcu.cz/index.php/Kalendář

Interoperabilita formátů pro groupware

Otakar Leopold

E-mail: oleopold@kerio.com

1 Úvod

Pojem interoperabilita dnes slýcháme stále častěji. Je jedno, zda se jedná o různé organizace, které spolupracují nebo o softwarové produkty. Skoro se dá říci, že je toto slůvko módním trendem a ten, kdo by jej ignoroval, bude znevýhodněn oproti konkurenci. Dokonce ani ty největší softwarové firmy si nemohou dovolit ignorovat interoperabilitu a vyvíjet uzavřená, byť kompletní softwarová řešení. Pojďme se podívat na úroveň interoperability v oblasti groupwarových řešení. Nejdříve si představíme dnes nejrozšířenější standardy. Potom se zamyslíme nad tím, zda stávající standardizace formátů postačuje pro úspěšnou interoperabilitu, a podíváme se na základní problémy dnes existujících groupwarových programů. Abychom si mohli jednoduše ukázat některé problémy, uvedeme si pro ně příklad na modelové situaci. Zamyslíme se nad příčinou problému, a pokud to bude možné tak si řekneme, jestli existuje způsob řešení.

2 Standardy

V současné chvíli existují dva hlavní standardy definující formát uložení groupwarových dat. Prvním je iCalendar popsaný v RFC 5545 [2] a druhým je vCard popsaný v RFC 2426 [3].

Formát iCalendar byl navržen skupinou Calendaring and Scheduling Working Group ze sdružení IETF. Založen je na starším formátu vCalendar, který definovali v Internet Mail Consortium(IMC) [4]. Přijat byl v roce 1998 jako RFC 2445 a loni v srpnu byl aktualizován na dnešní podobu. Definuje uložení základních kalendářových objektů, jako jsou kalendářové události, úkoly, poznámky, free-busy, popis časové zóny a alarmu. Všechny tyto objekty musí být ještě vloženy do hlavního objektu, který nese informaci o groupwarovém klientovi a o verzi formátu použitého pro uložení dat. Tento objekt si lze představit jako kontejner, do kterého se vkládají jednotlivé groupwarové objekty.

Standard vCard byl původně navržen Versit Consorciem v roce 1995 a o rok později přešlo jeho vlastnictví pod IMC [4]. Slouží k ukládání elektronických vizitek, které mohou být snadno sdíleny napříč internetem.

Oba zde uvedené standardy používají pro ukládání dat jednoduchý textový formát, ve kterém jsou hodnoty ukládány ve tvaru „název;parametry:hodnota“. Kódování znakové sady je utf-8 a pro zakončení řádek je použit dvojznak CRLF. Řádky by neměly být delší než 75 znaků a proto je ve formátech definována možnost jejich zalomení, kdy na začátku pokračujícího řádku musí být mezera nebo tabulátor.

Příklad kalendářové události:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
UID:19970610T172345Z-AF23B2@example.com
DTSTAMP:19970610T172345Z
DTSTART:19970714T170000Z
DTEND:19970715T040000Z
SUMMARY:Bastille Day Party
END:VEVENT
END:VCALENDAR
```

Příklad kontaktu:

```
BEGIN:VCARD
VERSION:3.0
N:Gump;Forrest
FN:Forrest Gump
ORG:Bubba Gump Shrimp Co.
TITLE:Shrimp Man
TEL;TYPE=WORK,VOICE:(111) 555-1212
ADR;TYPE=WORK:;;100 Waters Edge;Baytown;
 ;30314;United States of America
LABEL;TYPE=WORK:100 Waters Edge\nBaytown,
 30314\nUnited States of America
EMAIL;TYPE=PREF,INTERNET:forrestgump@example.com
REV:20080424T195243Z
END:VCARD
```

3 Standardy v praxi

Po seznámení se standardy můžeme nabýt dojmu, že napsat vlastního klienta podporujícího groupware je v podstatě jednoduché. Že stačí implementovat ukládání dat do přesně definovaného formátu a náš program bude plně funkční s kterýmkoliv klientem, který používá stejný standard. Bohužel náš svět není perfektní a tak velice brzo začneme narážet na problémy při implementaci a tes-

tování našeho programu. Pojďme se podívat na některé problémy na které můžeme během tvorby klienta narazit.

Představme si modelovou situaci, kde tým vývojářů dostal za úkol napsat groupwarový program podporující standardy iCalendar a vCard. Jedním z cílů projektu je spolupráce s co největším množstvím programů. Všichni vývojáři jsou zkušení, a proto první věc, na kterou se zaměří, je studium standardů. Standard pro iCalendar má přibližně 170 stran a pro vCard je dlouhý 42 stran. Asi nemá cenu číst standard jako celek, ale přečíst pouze základní části pro všeobecný přehled a zbytek aktuálně vyhledávat. Když se tak stane, přijde na řadu vývoj vlastního programu, který je doprovázen neustálým pročítáním standardu. Pro náš případ neřešíme vůbec problém komunikace, ale pouze práci s jednotlivými groupwarovými objekty.

Jakmile je k dispozici funkční verze, tak se skupina pustí do testování vzájemné spolupráce s jinými programy. Ačkoliv náš program plně podporuje standardy, tak testování s ostatními programy rozhodně není bezproblémové. Pojďme se podívat na jednotlivé problémy blíže.

4 Problémy

4.1 Čas konce události

Jeden z testovaných klientů špatně zobrazuje dobu trvání události vytvořené v našem programu, ale událost vytvořená v problémovém klientovi se zobrazí dobře i v našem programu. Podrobným prozkoumáním přijdeme na to, že klient nemá plně implementovaný standard. Ten umožňuje zapsat do události buď čas počátku a čas konce, nebo čas počátku a délku jejího trvání. Klient ale dokáže pracovat pouze s druhou variantou. Takže pokud je v události uložen čas konce, je délka události vyhodnocena jako nulová a jako taková je také zobrazena. Pokud výrobce testoval pouze v uzavřeném prostředí, tak mu bez problémů vše fungovalo a nikdy nepotřeboval implementovat podporu i pro druhý způsob uložení.

Tady je těžké určit, kde leží podstata problému. Někdo bude tvrdit, že to je nedostatek standardu, který by měl být striktnější. My ale máme dvě možnosti řešení, buď se přizpůsobíme a budeme všude striktně požívat dobu trvání, nebo kontaktujeme výrobce a upozorníme ho na nedostatek v jeho implementaci. První způsob řešení je snazší, ale asi méně správný, a druhý je komplikovaný a bude trvat nějakou dobu, než bude chyba opravena. Nevhodný je i z toho pohledu, že některé firmy nejsou ochotny chyby aktivně opravovat a i kdyby je opravili, stejně se vyskytne skupina uživatelů, kteří neprovedou update například kvůli ceně.

4.2 Organizátor

Při testování jsme si všimli, že někteří klienti zobrazují zakladatele události dvakrát. Po chvíli zkoumání této záhady dojdeme k zajímavému zjištění. Naše implementace nepovažuje organizátora události automaticky za jejího účastníka, a proto jej přidáme i mezi účastníky. Druhá aplikace ale organizátora automaticky řadí mezi účastníky, takže jej zobrazí dvakrát přesně podle uložených dat. Sama při založení události organizátora mezi účastníky nepřidá, takže naše aplikace jej nezobrazí jako účastníka.

Po konzultaci se standardem zjistíme, že ani jedna interpretace mu neodporuje. Ale co je tedy správně? Jedna strana namítne, že organizátor by se měl vždy účastnit, ale druhá to tak nevidí. V tomto případě je standard nedostačující a tento případ by měl definovat.

4.3 Celodenní události

Během zkoušení naše vývojáře zarazilo chování celodenních událostí v různých časových zónách. Ačkoliv je událost založena v jedné časové zóně, v jiné je zobrazena od začátku do konce dne lokálního času. Podle RFC se čas začátku takovéto události ukládá bez informace o časové zóně a pouze jako datum. Takže je to v pořádku.

To jestli je správné, že celodenní událost je nezávislá na časové zóně je spíše filozofická otázka. Jedním z případů, kdy je takovéto chování smysluplné jsou narozeniny. Své narozeniny budete chtít mít v kalendáři vždy nastavené na konkrétní datum bez ohledu na to, kde se budete právě nacházet. Ale to vede k problémům při výpočtu free-busy. To podle standardu obsahuje časové údaje v GMT a server při vytváření odpovědi nemá šanci vytvořit správnou odpověď. Chybí mu vazba události na časovou zónu. Jeden z klientů to řeší po svém, událost ukládá s lokálním časem a do objektu uloží údaj o celodennosti do vlastní položky. Tomu ostatní programy nerozumí a zobrazují události jako necelodenní.

4.4 Časové zóny

Na časové zóny si naši vývojáři stěžovali už při implementaci programu, protože převádění časů mezi různými zónami je komplikované. A u testování tomu není jinak. Pokud chceme uložit čas s konkrétním časovým pásmem, tak k datům musíme zapsat i plnou definici časové zóny. To je dokonale přenositelné, ale také nepraktické. Každý klient musí implementovat algoritmy pro práci s definicí časových zón. Díky složitosti definice zón tak mnoho klientů nemá plnou podporu pro standard, a to samozřejmě způsobuje potíže.

Příklad časové zóny:

```
BEGIN:VTIMEZONE
TZID:America/New_York
LAST-MODIFIED:20050809T050000Z
```

```
BEGIN:STANDARD
DTSTART:19671029T020000
RRULE:FREQ=YEARLY;BYMONTH=10;
 BYDAY=-1SU;UNTIL=20061029T060000Z
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
TZNAME:EST
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19760425T020000
RRULE:FREQ=YEARLY;BYMONTH=4;
 BYDAY=-1SU;UNTIL=20060402T070000Z
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
TZNAME:EDT
END:DAYLIGHT
BEGIN:DAYLIGHT
DTSTART:20070311T020000
RRULE:FREQ=YEARLY;BYMONTH=3;
 BYDAY=2SU
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
TZNAME:EDT
END:DAYLIGHT
BEGIN:STANDARD
DTSTART:20071104T020000
RRULE:FREQ=YEARLY;BYMONTH=11;
 BYDAY=1SU
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
TZNAME:EST
END:STANDARD
END:VTIMEZONE
```

V definici zóny může být zapsána kompletní historie pravidel pro přechody mezi letním a zimním časem, ale v praxi drtivá většina klientů podporuje pouze jedno pravidlo pro přechod na letní čas a jedno pro přechod na zimní. To je podpořeno i některými operačními systémy, které se chovají také tak.

Dalším problémem je změna pravidel pro změnu letního a zimního času. Pokud se pravidla změní, budou po jejich aktualizaci v programu všechny již existující události posunuty. Posun se projeví v úseku mezi počátkem původního a nového letního času a koncem původního a nového zimního času. To je způsobeno právě ukládáním definice časové zóny do události.

Někteří klienti se to snaží řešit mapováním časových zón v události na systémové časové zóny. Ani to bohužel není řešením, protože takoví klienti mají

problémy s událostmi z jiných operačních systémů, kde jsou naprosto jinak definované stejné časové zóny.

Může nás napadnout ukládat všechny časy v GMT, ale ani to nám nepomůže právě kvůli přechodům mezi zimním a letním časem. Takové události by v letním čase začínali o hodinu později než v zimním. Ke správnému zobrazení události v cílové časové zóně musíme k časům přičíst posun oproti GMT, a ten je v letním čase o hodinu větší.

4.5 Složitá opakovací pravidla

Pokud potřebujeme vytvořit pravidelně se opakující událost, definuje standard takzvaná opakovací pravidla. Každé pravidlo potom definuje množinu výskytů vytvořené události.

Během testování opakovaných událostí jsme narazili na obtíže při editaci opakovacího pravidla v různých klientech. Samotné výskyty události jsou zobrazeny korektně, ale v detailech je špatně zobrazeno opakovací pravidlo. Porovnáme uživatelské rozhraní programů a zjistíme, že některá pravidla pro opakování jdou zadat pouze v jednom z klientů.

Zamysleme se nad tím, jak tento problém vznikl. Zapisování opakovacích pravidel je ve standardu velice komplexní. Jdou tedy zapsat všechna myslitelná opakovací pravidla a některá dokonce více způsoby. Díky tomu je standard univerzální, ale dnešní programy se snaží vytvořit co nejjednodušší uživatelské rozhraní. Pokud by uživatel dostal grafický nástroj na plnou podporu pravidel, dozajista by nebyl jednoduchý a uživatelsky přívětivý. Vývojáři to řeší poskytnutím uživateli pouze rozumné podmnožiny opakovacích pravidel. Například jen málokdo bude potřebovat zadat pravidlo každou sudou středu každý třetí měsíc, proto jej klienti ani neumožňují. To ale nutně vede k výše zmíněnému problému. Opakování zadané v jednom programu nelze zobrazit v programu druhém. Každý z vývojářských týmů považoval za rozumnou jinou podmnožinu opakovacích pravidel a tyto dvě skupiny se ne zcela překrývají. Tento problém je zase řešitelný pouze dohodou mezi výrobci programů nebo přizpůsobením jednoho z nich.

4.6 Výjimky z opakovaných událostí

Další věcí, kterou každý program řeší trochu jinak jsou výjimky z opakovaných událostí. Pokud máme událost pravidelně se opakující každou středu, tak může nastat situace, kdy jeden výskyt budeme chtít přesunout na čtvrtek. Na to je standard připravený a definuje jak takovou výjimku uložit. Ale už neříká, jak má být s výjimkou naloženo, pokud dojde ke změně původní události.

Takže část klientů se chová tak, že do výjimky uloží pouze povinné a změněné položky, a zbylé vezme z původní události. Tento způsob je výhodný pokud dojde ke změně položky, která není výjimkována a změna se projeví i do výjimky.

U těchto klientů je problém například odebrat účastníky z jedné výjimky. Položka účastníka je vícehodnotová a není způsob jak ve výjimce říci, že z původní množiny hodnot jednu odeberte.

Zbytek klientů udělá kompletní kopii výskytu. Díky tomu je výjimka nezávislá. Tyto programy mají problém s nekompletními výjimkami z ostatních klientů. Ale zase mohou snadno například odebrat jednoho účastníka z jednoho výskytu. Na druhou stranu pokud někoho trvale přidáte, tak se jeho jméno neobjeví ve výjimce.

Oba způsoby jsou v souladu se standardem, ale navzájem nejsou plně kompatibilní a mají svoje nevýhody.

5 Závěr

Nakonec naše skupina potřebovala na odladění programu do použitelné podoby několikanásobně více času než na vlastní implementaci. Samozřejmě zde jsme si neuvedli všechny problémy, na které bychom narazili při testování. Ale z uvedených příkladů je zřejmé, že pouhý standard není postačující k vytvoření groupwarového řešení, které by bylo univerzálně použitelné. Ani mohutné testování nám nezajistí plnou interoperabilitu, protože jsou chyby klientů, které nemůžeme ošetřit na naší straně. Takže jedinou možností, jak vyřešit uspokojivě všechny nedostatky, je udržovat komunikaci s ostatními softwarovými výrobci. Za tímto účelem vznikl CalConnect [1]. Jedná se o konsorcium výrobců groupwarových řešení. Jeho cílem je právě zlepšení interoperability kalendářových a plánovacích dat napříč všemi programy a platformami. Součástí pravidelných setkání je takzvaný interop, na kterém se testuje právě vzájemná kompatibilita groupwaru. Samozřejmě ani CalConnect nevyřeší všechny problémy, ale napomáhá vzájemné komunikaci mezi jednotlivými firmami.

Literatura

[1] *CalConnect – The Calendaring and Scheduling Consortium.* http://www.calconnect.org

[2] *RFC 5545 – Internet Calendaring and Scheduling Core Object Specification.* http://www.ietf.org/rfc/rfc5545.txt

[3] *RFC 2426 – vCard MIME Directory Profile.* http://www.ietf.org/rfc/rfc2426.txt

[4] *Internet Mail Consortium.* http://www.imc.org/pdi/

Protokoly pro synchronizaci groupwareových dat

Štěpán Potyš

E-mail: spotys@kerio.com

1 Úvod

Přestože se internet dnes stává dostupným i tam, kde to před pár lety nebývalo zvykem, je jedním ze základních požadavků uživatelů na groupwarové nástroje možnost offline přístupu k datům. Tzv. online klienti, jako jsou např. různé webové aplikace, vyžadují dostupné připojení ke groupwarovému serveru a jsou proto velice vhodné spíše v situaci, kdy uživatel přistupuje na server z různých počítačů. Dnešní uživatelé však zpracovávají e-maily, plánují svůj čas a pracují s adresářem kontaktů i na cestách, kde je připojení k internetu nedostupné. Nástroje, které k práci používají tedy uchovávají kopii všech, nebo alespoň nějaké rozumné podmnožiny, dat lokálně na klientském počítači. K naplnění této představy, jak si ukážeme dále, je zapotřebí robustní mechanizmus, tzv. synchronizační protokol, zajišťující celou řadu operací, o kterých cílový uživatel velice často vůbec neví a ani by podobnými detaily být zatěžován neměl. O tom jaké protokoly jsou v současnosti k tomuto účelu k dispozici, o jejich schopnostech, jednoduchosti a kompatibilitě si povíme v následujícím textu.

2 Základní principy synchronizace

Nejdříve se zaměřme na to, jak celý synchronizační proces probíhá. Zcela základními operacemi, které musí každý synchronizační protokol umožňovat, je stažení dat ze serveru a naopak jejich uložení na server. V praxi se operace stažení dat ještě rozděluje na tzv. kompletní download a inkrementální download. Kompletním downloadem, nebo též plnou synchronizací, se rozumí stažení všech dat ze serveru. Potřeba inkrementálního downloadu, nebo též inkermentální synchronizace, je vzhledem ke stále rostoucí velikosti synchronizovaných dat zcela logickým krokem ve vývoji synchronizačních protokolů. Klient při ní stahuje jen ta data, která ještě nemá. Pokud se klient synchronizuje poprvé, je plná a inkrementální synchronizace totožná, protože na klientovi ještě nejsou žádná data.

Každá další inkrementální synchronizace však ušetří spoustu času a síťového provozu. Plná synchronizace se tedy používá jen v případech, kdy dojde k nějaké závažné chybě např. ke korupci dat na klientovi. Upload je ve své podstatě vždy inkrementální, protože prováděné změny se synchronizují průběžně a nedávalo by smysl ukládat na server data, která už tam nepochybně jsou. Slovem synchronizace budeme dále označovat obojí, download změn ze serveru i upload změn na server a budeme předpokládat, že se mohou při jedné synchronizaci dít vždy obě operace.

Úkolem synchronizace je tedy udržení konzistentních dat na obou komunikujících stranách. V ideálním případě je na obou stranách shodná kopie dat. K narušení konzistence dochází ve chvíli, kdy na jedné straně, řekněme na klientovi, dojde ke změně. V takovém případě je zapotřebí přenést tuto změnu i na server. Čím déle tuto operaci budeme odkládat, tím více změn se nahromadí a tím větší bude pravděpodobnost, že dojde ke změnám na obou stranách zároveň – jiný klient provede změnu a uloží ji na server. Rovněž se zvýší pravděpodobnost, že nastane změna na obou stranách ve stejných datech (tzv. konflikt), tj. případ, kdy jiný klient změní stejná data a uloží je na server dříve než náš klient. Množství změn, ani změny na obou komunikujících stranách, v praxi nebývají problém, avšak řešení konfliktů je jeden z nejhůře uchopitelných problémů při synchronizaci, protože v obecném případě nikdy nelze zcela s jistotou říci, jakým způsobem konflikt vyřešit. Způsob řešení konfliktů je tedy potřeba volit s ohledem na aktuální situaci až za provozu. Tento fakt je jistou výhodou pro vývojáře serverů, neboť zodpovědnost za řešení konfliktů přenáší na klienta. Ten umožní nastavit defaultní strategii, případně nechá rozhodnout uživatele pokaždé, když se konflikt vyskytne. Existují v zásadě tři možnosti:

1. client wins – přepsat konfliktní údaj na serveru verzí z klienta,
2. server wins – přepsat konfliktní údaj na klientovi verzí ze serveru,
3. merge – zkombinovat hodnoty obou verzí.

První dvě možnosti vždy zahazují nějaká data a v případě špatné volby dojde ke ztrátě dat. Merge zase není vždy pro daná data použitelný. Pokud bychom například změnili telefonní číslo zákazníka, bylo by v případě konfliktu možné zachovat obě verze telefonního čísla. Je zřejmé, že v případě čísla bankovního účtu by toto řešení nebylo příliš vhodné.

3 Rozšířené funkce

Kromě standardního stažení a uložení nabízejí protokoly často speciální funkce, které umožňují klientům snáze získávat a ukládat data, která potřebují.

- Vyhledávání – Některé protokoly využívají vyhledávací dotazy, aby omezily množství dat přenášených po síti na klienta. Někdy je například potřeba se dotázat jen na kontakty s určitou kategorií nebo události, které nezačínají příliš daleko v budoucnosti ani nekončí dávno v minulosti.

- Souhrnné dotazy – Umožňují stahovat data z více URL najednou nebo třeba rekurzivní procházení podsložek. Zjednoduší se tak dotazování klienta, neboť místo dotazu na každou složku v hierarchii je možné poslat jeden souhrnný dotaz.

- Zpětný kanál – V případech, kdy více klientů sdílí stejná data, je zapotřebí, aby se po uložení změny na server ostatní klienti dozvěděli, že ke změně došlo a aktualizovali své lokální kopie. To je možné zajistit periodickými dotazy klienta na server, kde se klient ptá, zda došlo k nějaké změně. Druhým způsobem je tzv. zpětný kanál, kterým server může na změny aktivně klienta upozornit. Někdy se tato upozornění posílají prostřednictvím UDP paketů ke klientovi, jindy klient použije speciální dotaz, na který server neodpoví okamžitě, ale až v případě, kdy dojde na serveru ke změně, nebo uplyne předem domluvený timeout. Tato technika je mnohem spolehlivější neboť UDP pakety se ze serveru na klienta nemusí dostat. UDP zpětný kanál je tedy vhodné kombinovat s periodickými dotazy.

- Synchronizace celých objektů vs. synchronizace po propertách – Někteří klienti využívají možnosti uložit na server např. jen tu část kontaktu, která se změnila. Místo posílání celého kontaktu tak klient uploaduje jen např. telefonní číslo, které uživatel právě změnil.

- Přístupová práva – Některé protokoly spoléhají na to, že klient, který se aktuálně na server připojuje, dostane od serveru jen ta data, která smí vidět. Tyto protokoly však klientům neumožňují, aby uživatel mohl nastavit sdílení dat s jinými uživateli. Protokoly postavené na WebDAVu tuto možnost poskytují.

- Informace o uživatelích – Sdílení dat s jinými uživateli často vyžaduje, aby uživatel, který sdílení nastavuje, měl možnost získat základní informace o ostatních uživatelích, kterým chce dát přístup ke svým datům.

- Speciální dotazy – Protokoly navržené pro práci jen s jedním typem groupwarových dat, jako např. CalDAV a CardDAV, mívají některé speciální dotazy optimalizované právě pro daný typ dat. Např. v případě protokolu CalDAV je možné stáhnout ze serveru událost, včetně jejího opakovacího pravidla a konkrétní výskyty dopočítat na klientovi, nebo využít speciální dotaz, který klientovi dá tyto výskyty spočítané. Tím se zjednodušuje logika klienta.

4 Existující standardy

V současnosti jsou nejpoužívanější protokoly pro synchronizaci groupwarových dat postaveny na technologiích HTTP a XML. Výhodou HTTP je zejména jeho průchodnost přes firewally, XML je zase rozšířeným standardem pro serializaci dat.

- Microsoft WebDAV

  Je postaven na protokolu WebDAV (RFC2518) a je použit např. v Microsoft Entourage k synchronizaci e-mailů, kontaktů a kalendářových událostí. Umožňuje vyhledávání pomocí speciálních SEARCH requestů obsahujících dotazy obdobné SQL. Stahování dat je realizováno buď po celých objektech a to HTTP metodou GET, nebo po jednodlivých položkách metodou PROPFIND. Obdobně je tomu u uploadu metodami PUT a PROPPATCH. Souhrnné dotazy jsou realizovány dávkovými metodami SEARCH, BPROPFIND, BPROPPATCH, atd. Zpětný kanál řeší pomocí HTTP UDP paketů, které však mohou být filtrovány firewallem a nejsou příliš spolehlivé. Proto klient používá ještě periodický POLL request ke zjištění, zda došlo na serveru k nějakým změnám v uživatelském mailboxu. Protokol pracuje s přístupovými právy, avšak ta jsou velice úzce uzpůsobena implementaci práv v Microsoft Exchange.

- Microsoft Exchange ActiveSync

  Tento protokol je používán na mnohých mobilních zařízeních a byl vyvinut firmou Microsoft. Data se přenášejí ve WBXML. Je v některých ohledech rovněž přizpůsoben architektuře Microsoft Exchange např. předpokládá existenci Global Address Listu. Dotazuje se vždy po složkách a přístupová práva nechává vyřešit serverovou stranu. Zpětný kanál realizuje trvale navázaným HTTP spojením. Protokol umožňuje sychnronizovat kontakty, kalendáře, úkoly a emaily.

- Exchange Web Services

  Poslední protokol z dílen Microsoftu. Je implementován např. v posledí verzi Microsoft Entourage for EWS. Tento protokol je uzpůsoben pro synchronizaci klienta s Microsoft Exchange serverem a poskytuje funkce známé z MAPI (Messaging API – rozhraní pro přístup k datům na MS Exchange používané ve všech verzích MS Outlooku). Umožňuje synchronizaci všech typů groupwarových dat.

- CalDAV

  Standard definovaný v RFC 4791. Je speciálním protokolem navrženým k synchronizaci kalendářů a kalendářových událostí. Tomu odpovídají

i jeho speciálně formulované dotazy např. na stažení výskytů všech událostí v daném časovém intervalu. Protokol umožňuje vyhledávání uživatelů na serveru, přístupová práva ke sdíleným složkám (převzato z rodičovského protokolu WebDAV). Jednotlivé události jsou do XML serializovány do bloku CDATA ve formátu VCALENDAR popsaném v RFC5545.

- CardDAV

  Stále ještě ve formě draftu. Je protokol navržený pro synchronizaci kontaktů. Podobně jako CalDAV je postaven na WebDAVu a poskytuje speciální dotazy pro práci s kontakty. Ty jsou rovněž do XML serializovány do bloku CDATA ve formátu VCARD popsaném v RFC2426. Ze své podstaty je tento protokol mnohem jednodušší než CalDAV.

- SyncML

  Podobně jako ActiveSync je protokol SyncML nejčastěji používán na nejrůznějších mobilních zařízeních. Data jsou přenášena ve WBXML. Umožňuje synchronizaci různých typů groupwarových dat, nejčastěji kontaktů, kalendářů, úkolů a emailů.

5 Závěr

V předchozích odstavcích jsme si vyjmenovali možnosti některých současných synchronizačních protokolů, které ve skutečnosti řeší totéž, jen je každý z nich navržen za různých okolností, pro různé typy zařízení či za účelem optimalizace některých operací. Přestože jsou všechny postaveny na HTTP a XML, jejich struktura je zcela odlišná, což je činí vzájemně nekompatibilními. Situace s kompatibilitou je však ještě složitější. Díky komplikovanosti některých protokolů je jejich plná implementace prakticky nemožná. Jako příklad uveďme protokol WebDAV, který je základem protokolů CalDAV a CardDAV. Kdybychom se podívali na RFC4791 standardu CalDAV a základní dokumenty potřebné k implementaci protokolu – RFC2445, RFC2518, RFC3744,RFC3253, dostaneme definici standardu čítající více než 500 stran textu. Přitom samotné RFC3744 definující přístupová práva ve WebDAVu je natolik obecné a komplikované, že neexistuje snad žádná implementace, která by jej plně zahrnovala. Při takovéto komplexnosti není divu, že se stále setkáváme s klienty, kteří si různé části standardů vykládají po svém, což způsobuje mnohdy zcela zásadní problémy při synchronizaci. Těmto potížím se snaží předejít různá sdružení, pořádající testy interoperability, jako je například IOP-testing pořádané sdružením CalConnect. Nutnost účasti na podobných testováních však na druhou stranu zhoršuje dostupnost standardu těm, kteří se těchto akcí z nějakého důvodu účastnit nechtějí, nebo nemohou.

Literatura

[1] *CALCONNECT.* http://www.calconnect.org

[2] *OASIS.* http://www.oasis-open.org

[3] *The Internet Engineering Task Force (IETF).* http://www.ietf.org

Kerberos v prostředí Microsoft Windows

Ondřej Ševeček

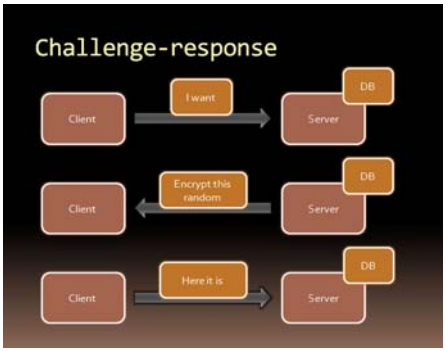
E-mail: ondrej@sevecek.com

Autentizační protokoly

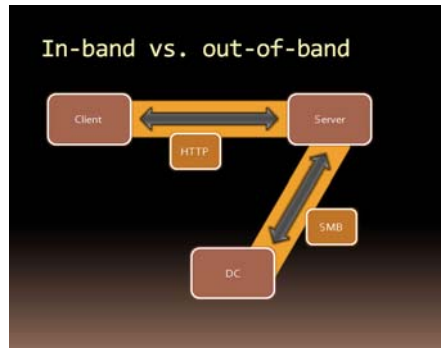
Od roku 2000 kdy firma Microsoft vydala operační systém Windows 2000 je základem jejich počítačových sítí LDAP databáze Active Directory. Ukládá primárně uživatelské účty a tedy i jejich autentizační a některé autorizační informace. Jsou to loginy, hesla (obvykle jen jejich heše), uživatelské skupiny a jejich členství a některé speciální další autorizační parametry jako jsou parametry Kerberos delegace apod. Tato databáze nahradilo dřívější SAM (Security Accounts Manager) databázi přítomnou ve Windows NT a postavenou na technologii registrových souborů.

Víceméně integrální součástí databáze je služba LSA (Local Security Authority) poskytující dva síťové autentizační (a v případě Kerberosu i autorizační) protokoly Kerberos v5 a LM/NTLM/NTLMv2 (tři vývojově následující protokoly, nadále označované jako rodina NTLM). Oba protokoly používají databázi účtů/heší a mají do této databáze důvěryhodný přístup právě k heším (loginy může číst libovolný uživatel i vzdáleně). Protokol Kerberos v5 přišel také teprve s Windows 2000 a je podporován právě pouze v Active Directory doménách a na operačních systémech Windows 2000, XP, 2003 a novějších (rodina NT). V takovém prostředí je výchozí a preferovanou autentizační metodou, rodina NTLM je však stále k dispozici kvůli zpětné kompatibilitě a v některých speciálních případech, kdy není možné použít Kerberos.

Kryptografické, ale i infrastrukturní kvality Kerberosu jsou lepší než v případě NTLM. Cílem tohoto příspěvku je seznámit posluchače s těmito principiálními rozdíly a ukázat jim některé zajímavosti, které Kerberos umožňuje právě v prostředí s Microsoft Windows. V následujícím textu je mnoho kryptografických a technických a infrastrukturních informací zjednodušených kvůli jednoduchému pochopení, avšak ne nepravdivých. Není cílem protokoly definovat, ale poskytnout rychlé porozumění a porovnání.



Obr. 1



Obr. 2

Single-Sign-On

V podnikových sítích se autentizační protokoly používají s cílem poskytnout uživatelům tzv. single-sign-on (SSO). Myslí se tím cíl, aby uživatel služeb nebyl nucen zadávat přístupové informace (login/heslo) vícekrát než jednou při prvním přístupu k systému.

Vzhledem k tomu, že Microsoft nabízí tyto dva protokoly, všechny služby dodávané firmou také tyto dva protokoly podporují. Jako příklad je možné uvést sdílené soubory (SMB), RPC/DCOM, RDP (Terminal Services), ale i HTTP (Microsoft IIS web server, ISA Server firewall apod.), SMTP, POP3, IMAP4 (Microsoft Exchange Server mail server), DNS (DNS dynamic update client a Microsoft DNS Server) a IPSec.

Rodina protokolů NTLM

Jedná se historicky o tři verze protokolu vyvinutého firmou Microsoft pro potřeby IBM OS2 a Microsoft Windows v polovině 80. let. Implementace byla firmou chráněna až do roku cca 2005 kdy firma detaily oficiálně zveřejnila.

Jedná se o protokoly LM, NTLM a NTLMv2. Protokoly fungují na principu challenge-response kdy autentizační server generuje náhodné číslo jako challenge a porovnává výsledek jejího zašifrování heší uživatelského hesla s tím, co má uloženo v databázi (viz obr. 1).

Tabulka 1 sumarizuje algoritmy a parametry všech tří protokolů.

Z pohledu infrastrukturní bezpečnosti je zajímavý přenos autentizační informace skrz službu, ke které se klient připojuje, tzv. Pass-Through (viz obr. 2). To je zásadní problém pro least-privilege prostředí.

Tab. 1

Protokol	Algoritmus	Heš	Poznámky
LM	DES	DES, heslo rozděleno na 2×7 znaků, US-ASCII, jen velká písmena	tragicky slabé
NTLM	DES	MD4, heslo 64 znaků Unicode	neexistuje vzájemná autentizace, slabá ochrana proti replay útoku
NTLMv2	HMAC-MD5	MD4, heslo 64 znaků Unicode	za určitých podmínek vzájemná autentizace, ochrana proti replay útokům pomocí kontroly času client/server v rozsahu 30 minut

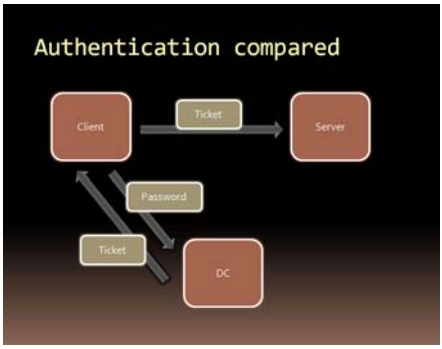
Pokud vynecháme z úvah možnost odposlechu síťové komunikace, privátní informace (i když v zašifrované podobě) zbytečně prochází rukama správce služby, která k nim vůbec nemá mít přístup. Druhý zásadní problém je absence vzájemné autentizace, takže klient neví, zda služba, se kterou komunikuje je ta, kterou očekává.

Kerberos

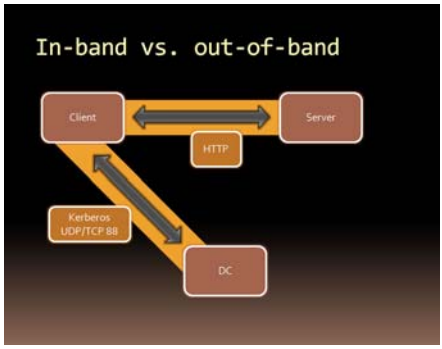
Implementace protokolu používá MD4 k výpočtu heše uživatelského hesla a RC4, 3DES, nebo AES 256 (od Windows Vista/2008) k šifrování přenosu. Pokud uživatel požaduje přístup k nějaké službě, a tato služba má svůj účet (heslo/heš) uloženou v důvěryhodné databázi, uživatel si pro tuto službu vyžádá přístupový tiket od autentizačního serveru. K vydání tiketu pro službu je (velmi zjednodušeně řečeno) potřeba použít uživatelovy přihlašovací údaje.

Na rozdíl od NTLM však jejich přenos probíhá pouze mezi klientem a autentizačním serverem. Protože služba musí mít v databázi také svůj účet, jsou následné transakce vzájemně ověřeny (viz obr. 3 a 4).

Pro zajímavost se zmíníme o jedné vlastnosti implementace, která je obvykle špatně chápána. Na rozdíl od NTLM, které používá časová razítka, aby omezilo pouze pravděpodobnost replay útoku, Kerberos používá časová razítka tak, že replay útok (vůči jedné stejné autentizační databázi) není možné provést. V této souvislosti existuje politika, kdy autentizační databáze vyžaduje synchronizaci času s klientem (ve výchozím stavu v rozsahu ± 5 minut).



Obr. 3



Obr. 4

Těchto 5 minut je chápáno jako bezpečností míra, tedy pokud bychom čas prodloužili, snížili bychom bezpečnost (zvýšili pravděpodobnost replay útoku). Ale jedná se ve skutečnosti pouze o výkonnostní charakteristiku. Autentizační server si totiž ve skutečnosti pamatuje všechna časová razítka přicházejících od klientů. Musí si je ale pamatovat právě jen na dobu oněch 2×5 minut.

Podstatnou infrastrukturní schopností této implementace Kerberosu je ale tzv. delegace. Starší NTLM ji totiž neumožňuje a omezuje tak možnosti SSO pro složitější prostředí síťových služeb.

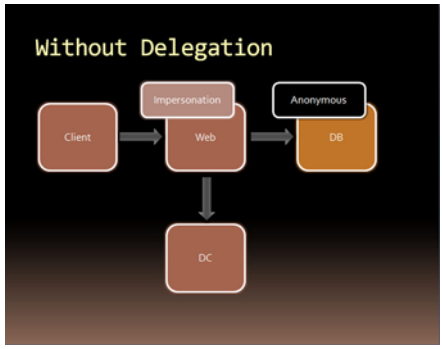
Delegace

Delegaci je možno popsat na následujícím příkladu. Klient se přihlašuje na nějakou síťovou službu, například web server s webovou aplikací. Aplikace potřebuje dále přístup na další zadní služby, jako jsou například databázové servery. Přístup dozadu ale opět vyžaduje přihlášení původního uživatele, protože databáze má implementovánu autorizaci právě na bázi identit uživatelů bez ohledu na to, zda do ní přistupují přímo, nebo přes nějakou službu (viz obr. 5 a 6).

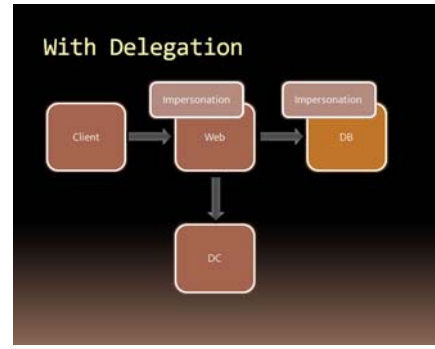
Protokol NTLM toto neumožňuje, protože přední služba nezná přístupové informace uživatele – ty jsou zašifrovány a pouze přeposílány mezi klientem a autentizačním serverem. Ten také nemá co by mohl poskytnout službě, aby to mohla sama použít směrem na zadní databázový server.

V případě kerberosu je to jinak. Z normálního principu funkce umí autentizační server vydávat tikety pro služby. Pokud je to tedy následně povoleno, jedna služba, která dostala tiket od uživatele, si může na jeho základě vyžádat tiket pro službu jinou (podle obr. 7).

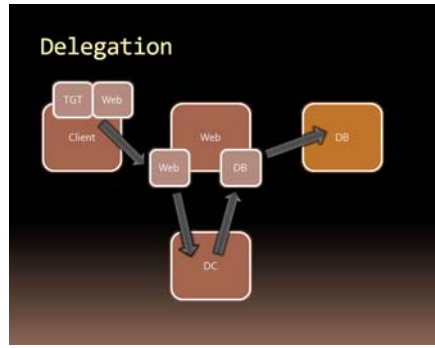
Vydání tiketu je tedy možné na základě uživatelova autentizovaného požadavku, nebo v tomto případě na základě jiného služebního tiketu (samozřejmě



Obr. 5



Obr. 6

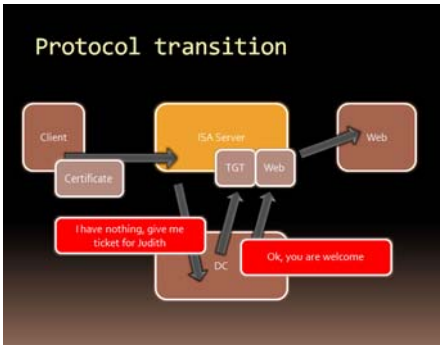


Obr. 7

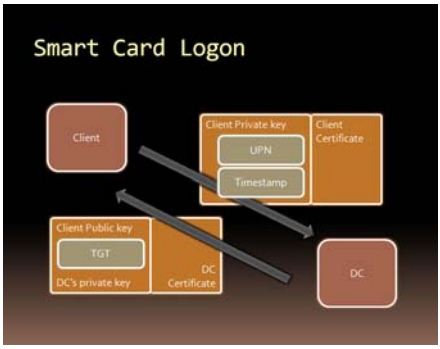
jen, pokud je to povoleno v politice služby). Vydání tiketu je ale možné i úplně bez jakéhokoliv „důkazu“, tedy naprosto „zadarmo“, pokud si o něho požádá důvěryhodná služba. Tato metoda se nazývá protocol transition.

Protocol Transition

V předchozím případě delegace se čekalo, že klient má přihlašovací údaje, díky kterým si může od autentizačního serveru vyžádat tiket. Toto ale není vždy možné. Autentizační databáze obsahuje pouze MD4 heše hesel. Pokud by bylo potřeba ověřovat uživatele jiným způsobem, například certifikátem, otiskem prstu, sítnicí, jiným typem heše apod., nebylo by možné použít tikety a tuto Kerberos implementaci vůbec. To by snižovalo možnosti aplikací a omezovalo SSO. Od Windows 2003 je však i na toto řešení v podobě protocol transition. Jedná se přesně o převod jedné autentizační metody na Kerberos.



Obr. 8



Obr. 9

Pokud je nějaká služba správcem zdůvěryhodněna, může si vyžádat tikety úplně bez jakékoliv před-autentizace (viz obr. 8).

Přihlašování čipovými kartami

Autentizační databáze obsahuje pouze MD4 heše hesel. Pro Windows infrastrukturu bylo však potřeba zajistit možnost přihlašování certifikáty a public-private klíči. V případě předchozí protocol transition je ale potřeba, aby služba byla zdůvěryhodněna. Problém je, že si potom může vyžádat tiket pro libovolného uživatele. Důvěru tedy správce vyslovuje jen ve velmi speciálních případech, protože nebezpečnost takového nastavení je jistě veliká.

Naopak každý uživatel a stanice/server by měla být schopna přihlašovat uživatele na základě PKI certifikátů, aniž by tato informace byla vůbec uložena v Active Directory, a i když vůbec nemá nic společného s heslem uživatele.

Kerberos tedy byl rozšířen o protokol PKINIT umožňující před-autentizaci pomocí PKI certifikátu uživatele. Autentizační server potom vydává tikety pouze na základě důvěry certifikační autoritě, která takový certifikát vydala (viz obr. 9).

Závěr

Microsoft implementace Kerberos v5 protokolu je velmi zajímavá z pohledu veliké integrace do Windows prostředí. Většina podnikového autentizačního provozu je obvykle realizována právě tímto protokolem. Kerberos tedy zásadně ovlivňuje infrastrukturní bezpečností východiska a každý správce i designer sítí by si jich měl být dobře vědom, protože jejich pochybení a neporozumění vytváří velmi bezpečnostně citlivé díry do infrastruktury.

Otevřené mikroplatební schéma pro rozsáhlé infrastruktury

Roman Žilka, Pavel Tuček, Václav Matyáš, Andriy Stetsko

E-mail: zilka@fi.muni.cz, tucek@ics.muni.cz, matyas@fi.muni.cz, stetsko@fi.muni.cz

1 Terminologie úvodem

Mikroplatby jsou slovem, které máme v prostředí českého internetu možnost okrajově vídat zhruba od konce devadesátých let. Jak slovo samo napovídá, jedná se o způsob plateb, a to za služby i hmotné a digitální zboží. Ačkoli u nás mikroplatby zatím nedoznaly valného rozmachu, zejména pro nehostinnou legislativu [6], ve světě tento progresivní obchodní model kvete. Lze doufat, že se tak v dohledném čase stane i u nás; mj. i proto, že mikroplatební způsob účtování ladí do not průkopníkům moderních paradigmat jako cloud computing či SaaS (Software as a Service, software jako služba) [4]. Tento příspěvek představí nejen mikroplatby jako takové, ale i právě vznikající otevřený mikroplatební systém, nevyhne se diskuzi některých úskalí takového počínání a načrtne klíčové použité algoritmy.

Výklad otevřeme zavedením klíčového pojmu ze zkoumané oblasti, a to (*elektronického*) *platebního schématu* ((E)PS), které lze chápat jako množinu pravidel a protokolů, které umožní provozovat prodej digitálního i hmotného zboží. Různá PS mohou nabídnout různé vlastnosti jako např. bezpečnost, výpočetní efektivitu, jednoduchost či anonymitu účastníků. Ti jsou v PS obvykle rozvrženi do následujících tří rolí:

- *prodejce*;
- *zákazník*/*uživatel*/*kupující*;
- *broker*: důvěryhodná třetí strana schopná převést reálné peníze od zákazníka k prodejci.

V typickém PS zákazník posílá prodejci jakési digitální platidlo – virtuální *mince* – výměnou za zboží. Broker je naproti tomu schopen prodejci tyto mince proměnit za reálné peníze, které odebere z majetku zákazníka. O brokerovi lze

bez podstatné újmy na obecnosti uvažovat jako o bance či firmě a o zákazníkovi jako klientu této banky, resp. zaměstnanci firmy. Majetek zákazníka, kterým pak broker disponuje, je v prvém případě zůstatek na účtu, v druhém případě výplata pro nadcházející měsíc.

Transformace virtuálních mincí na reálné peníze si však žádá jistý prováděcí výdaj. Ten je ve světě bank a běžných bankovních převodů obvykle ve srovnání s převáděnou částkou zanedbatelný. Představme si však, že virtuální mince mají přiřazenu nominální hodnotu menší než 1 Kč a že prodej zboží o takové malé hodnotě či hodnotě jednotek Kč není nic neobvyklého. Když k tomu přidáme ještě vysokou frekvenci (několikrát za minutu na jednoho reálného zákazníka) takových miniaturních transakcí, ukáží se konvenční bankovní převody jako zcela nevhodné. Na Internetu (a nejen tam) lze skutečně spatřovat příležitosti pro takovéto frekventované a nízkohodnotové nákupy – tedy *mikroplatby*: účtování sledování internetové televize po minutě či účtování za každé jedno kliknutí při pohybu návštěvníka na webu a další. PS, jehož účelem je vypořádat se se specifiky mikroplateb a učinit je realizovatelnými, se pak nazývá (*elektronické*) *mikroplatební schéma*. Dlužno dodat, že předchozí větu skutečně nelze brát jako definici, neboť není jednotně stanoveno, kde končí konvenční PS a začínají mikroplatební schémata. Zatímco v akademické sféře se pojmem „mikroplatební" zpravidla rozumí „disponující dostatečnými zjednodušeními jednotlivých transakcí, takže režijní manipulační náklady na převody reálných peněz jsou únosné navzdory nízkým hodnotám prodávaného zboží", zákon vymezuje mikroplatby (i když ne nutně doslova pod tímto jménem) jemněji a spíše za jejich charakteristiky považuje omezenou hodnotu a způsob zpracování (pro stav v ČR viz zákon č. 124/2002 Sb.).

Jednotlivá PS se výrazně liší tím, jak často probíhá transfer reálných peněz mezi prodejci a jejich zákazníky. Aby k převodu mohlo vůbec dojít, příslušné virtuální mince se musí napřed dostat do rukou brokerovi. U tzv. *on-line PS* se broker po síti aktivně účastní každé jednotlivé prodejní transakce mezi prodejcem a zákazníkem. Mince se tak k němu dostávají hned při nákupu zboží. U *off-line PS* broker není účasten a mince mu jednotliví prodejci zasílají po dávkách dle dohody. Zmíněný rozdíl mezi on-line a off-line schématy má další podstatné důsledky; tabulka 1 uvádí některé ze zajímavých.

2 Projekt pod vlajkou FI MU a Y Soft s. r. o.

Jak bylo naznačeno, projekt, o kterém bude řeč, se týká mj. vývoje původního PS (ať už do jakékoli míry mikroplatebního); konkrétně ve spolupráci Fakulty informatiky Masarykovy univerzity (http://www.fi.muni.cz) a firmy Y Soft s. r. o. (http://www.ysoft.cz), průmyslového partnera fakulty. Firma Y Soft zde hraje roli zadavatele a sponzora a několik pracovníků Laboratoře bezpečnosti a aplikované

Tab. 1: On-line versus off-line PS

	off-line PS	lev **on-line PS**
autentizace nákupčího prodejci a naopak	bývá žádoucí alespoň na úrovni pseudonymu (ID)	není nezbytně nutná: pouze broker se dozví, kdo nakupuje a přidělí prodejci reálné peníze od původce, který zůstává pro prodejce anonymním
autorizace transakce brokerem	není realizovatelná – broker se o proběhlém prodeji dozví až později	je realizovatelná – broker tak může okamžitě blokovat uživatele, stanovovat peněžní limity na utrácení apod.
stálé, zajištěné a zabezpečené síťové spojení mezi brokerem a prodejci	není nutné	je nutné
časová náročnost jedné transakce	v mezích vhodných pro použití bezkontaktní čipové karty nákupčím	nevhodná pro bezkontaktní čipovou kartu

kryptografie fakulty roli řešitele. Budiž znovu zdůrazněno, že vývoj PS není zdaleka u konce a představen zde bude ve své aktuální pracovní verzi.

Cílené PS má být schopné obsloužit uživatelské základny čítající desítky až desetitisíce uživatelů s řadou prodejců nabízejících zboží jako použití počítače na jistou dobu, parkovné, tisk na místní tiskárně, potraviny z prodejních automatů a další drobné předměty a služby, které spotřebovávají lidé v prostředí firemních budov. Nejedná se tedy v tomto případě jen o prodej po Internetu. Nákupčími zde mohou být jak zaměstnanci firem, kteří nakupují u svých zaměstnavatelů i jinde, tak nezávislí jednotlivci. Nákupčí disponují bezkontaktní čipovou kartou, mobilním telefonem se speciálním SW a vhodným komunikačním rozhraním či podobným zařízením, které za ně vykonává nákup formou protokolů předepsaných platebním schématem. Prodejce pak zastupují jednotlivá PC, automaty a další zařízení, která uživateli něco nabízejí a která jsou pro tento účel vybavena bezdrátovým komunikačním rozhraním.

Pro malý kus HW, kterým je zmíněná uživatelská „čipová karta“, je předem nutné stanovit pevné požadavky, neboť PS se nesmí svou výpočetní a komuni-

kační složitostí vymknout dosahu podobných malých zařízení a zároveň je třeba vystačit s cenově průměrným zařízením. Stanovujeme proto, že *čipová karta*, jak bude zařízení nadále pro jednoduchost označováno,

- neumí sama měřit čas;
- má výpočetní výkon a paměťovou kapacitu běžné programovatelné čipové karty;
- nemá vlastní zdroj energie;
- neumí samovolně detekovat, kdy jí byl odebrán (vnitřní stav karty je zmražen) a navrácen (karta pokračuje tam, kde skončila) přísun energie;
- komunikuje bezdrátově s parametry signálu podobnými zařízením pro RFID;
- nemá žádné vstupní rozhraní pro komunikaci s člověkem přímo;
- může mít malý displej (držitelé karet bez displeje mohou být v některých situacích znevýhodněni).

Co se týče bezpečnostních vlastností čipové karty, nebyly stanoveny žádné exaktní nároky. Akceptujeme tu možnost, že dostatečně schopný a majetný útočník bude moci například přečíst paměť karty, naklonovat ji a eventuálně poté dokonce provést cílené modifikace extrahovaného obsahu karty. Zabránit zcela všem možným útokům tohoto charakteru čistě v rámci PS je nemožné, implementujeme proto v mezích možností více či méně silná opatření v různých bodech návrhu PS dle citu a kolektivní zkušenosti (podrobněji ke klasifikaci útoků na čipové karty a podobná zařízení viz [9]). Reálným cílem je učinit možné útoky na kartu jako takovou cenově nevýhodné a dostatečně rychle detekovatelné. Přesto nezastíráme skutečnost, že broker (případně spolu s prodejcem) čelí možnosti jisté (únosné) finanční ztráty.

Následují některé další vstupní axiomy návrhu netýkající se přímo čipové karty.

- Off-line PS: a priori byla vyřazena on-line schémata, neboť jejich vrozené nároky na síťovou infrastrukturu se zdály přílišným a zároveň ne nezbytným břemenem. Zároveň je umožněno nasazení pohodlných bezkontaktních čipových karet.
- Izolace zákazníků navzájem a prodejců navzájem: nikdy spolu nekomunikují prodejci navzájem a stejně tak uživatelé navzájem.
- Každý prodejce je v kontaktu s každým relevantním (viz kap. 4) brokerem alespoň 1× za předdefinovanou, konstantní dobu – *poločas životnosti mince* (C_P). Spadá do rozsahu 1–3 dny a broker s prodejcem si během každého tohoto pravidelného setkání (zpravidla formou elektronické komunikace) vymění nashromážděné mince, informace o zablokovaných uživatelích apod.

Do mezí návrhu PS nespadají mj. následující problémy.

- Převody reálných peněz: schéma nestanovuje, zda si budou subjekty předávat reálné finance bankovními převody mezi účty, strháváním z platu, formou SMS s navýšenou cenou, hotovostně či nějak jinak. Očekává se, že každý broker bude smluvně svázán s každým prodejcem, od něhož přijímá mince k proplacení a s každým uživatelem, jehož reálným majetkem disponuje, aby byla zajištěna potřebná důvěra mezi některými stranami, vymahatelnost škod, postižitelnost krádeží atd. Formulace těchto smluv rovněž nespadá do hranic schématu.
- Pravidla pro udržení a vyřazení uživatelů, prodejců či brokerů ze smluvních vztahů: tato budou ošetřena ve smlouvách mezi relevantními subjekty.
- Řízení přístupu zúčastněných osob k jednotlivým kusům hardwaru a dalšího fyzického vybavení, které se nějak týká prodejní infrastruktury: toto téma nabývá na důležitosti například u zpoplatnění tiskových úloh produkovaných na společných tiskárnách a vyloučení situace, kdy si pro vytištěný materiál přijde dříve někdo jiný než původce a plátce tiskové úlohy.
- Volba kryptografických algoritmů: PS používá hašování a symetrické a asymetrické šifrování a nestanovuje, jaký algoritmus konkrétně bude nasazen. V plánu vývoje PS je však rozvaha kroků, které je třeba učinit v krizovém bodě prolomení některého z algoritmů v aktuálním užívání.

Konečně, cílené PS má být škálovatelné a konfigurovatelné, aby tak mohlo nabídnout různou úroveň komplexnosti, bezpečnosti a dalších parametrů na míru různým scénářům nasazení. Celá specifikace PS má být neproprietární. To je jednou z motivací pro projekt jako t akový: ku znalosti všech osob zúčastněných na projektu v současné době existují buď schémata, která jsou otevřená, ale chybí jim některé klíčové vlastnosti (zabezpečení, konfigurovatelnost a obecnost, ...), anebo schémata, která sice dosahují těch kvalit, kterých bychom rádi dosáhli i my, ale jsou uzavřená.

3 PayWord a slepé cesty vývoje

Započetí úvah o tvaru PS předcházelo zkoumání existujících schémat. Vyplynuly z něj klíčové požadavky na cílové schéma – např. absence síťového spojení mezi prodejcem a brokerem. Schéma, které se zdálo nejvhodnější coby startovní bod vývoje, byl PayWord [7] R. Rivesta a A. Shamira. Argumenty proti kandidátům budou následovat za přehledovým popisem PayWordu samotného.

Iniciálním bodem v PayWordu je registrace uživatele u brokera (např. otevření účtu v bance). Broker uživateli vystaví *certifikát*, který obsahuje

- identifikátor uživatele;

- identifikátor brokera;
- veřejný klíč uživatele;
- dobu platnosti certifikátu.

Certifikát je digitálně podepsaný brokerem a podává informaci o tom, že uvedený broker je schopen proplatit virtuální mince, které budou podepsané soukromým klíčem příslušným k uvedenému veřejnému klíči.

Držitel certifikátu – zákazník – je schopen vystavovat mince, které prodejce přijme. Ten tak učiní, neboť existuje broker, který mu je proplatí za reálné peníze. Při nákupu zákazník nejprve pošle prodejci tzv. *závazek* (*commitment*), který je podepsaný zákazníkovým soukromým klíčem a obsahuje

- aktuální datum a čas;
- identifikátor prodejce;
- certifikát zákazníka;
- kořen hashového řetězce (viz dále).

Závazek je první částí pomyslné mince, která věrohodně specifikuje, kdo od koho kupuje. Nespecifikuje ovšem částku, kterou zákazník zaplatil – ta je reprezentována tzv. *hashovým řetězcem*. Jde o uspořádanou množinu $(n+1)$ binárních řetězců (*článků*) $W_0, W_1, \ldots, W_n$ vypočtených dle následujícího algoritmu:

- nechť W_n je náhodné;
- pro každé $(n-1) \geq n \geq 0$ nechť $W_{n-1} = hash(W_n)$.

Klíčovou charakteristikou je, že za znalosti pouze W_0 je

- snadné ověřit, zda libovolný daný binární řetězec je či není článkem hashového řetězce (jisté rozumně omezené délky) a pokud ano, tak jak vzdálený je od kořene;
- nemožné odvodit předchozí články hashového řetězce.

W_0 je nazván *kořenem* řetězce a je přítomen v závazku, který zákazník zašle prodejci. Pouze zákazník zná předem ostatní články řetězce, protože jej sám sestrojil. Tím, že prodejci zašle článek W_x, zákazník zaplatí x jednotek reálné měny (cena jedné jednotky je v systému pevně stanovena předem). Prodejce může ověřit zopakováním x hashovacích cyklů na W_x, že tento má skutečně hodnotu x jednotek. Z vlastností hashového řetězce však vyplývá, že prodejce sám není schopen vypočítat W_x pro žádné $n \geq x \geq 1$, a tedy není schopen účtovat zákazníkovi více než kolik zákazník skutečně zaplatil. Závazek spolu s W_x pak tvoří celek virtuální mince. Broker při jejím proplácení provádí hashovací cyklus na W_x tak dlouho, než se dostane ke kořeni uloženém v závazku. Odpovídající počet jednotek měny pak v podobě reálných financí převede do majetku prodejce vyznačeného v závazku.

Schéma dále přirozeně podporuje jistou formu „přihazování“ k již zaplacené částce: má-li zákazník předpočítaný řetězec délky n, ale zaplatil článkem W_x, kde $x < n$, tak může v rámci jedné otevřené transakce připlatit ke stejnému závazku další peníze tím, že vyjeví prodejci W_y, kde $n \geq y \geq x$. Tím zákazník připlatil $(y - x)$ jednotek měny. Prodejce pak může W_x zahodit a minci pak tvoří závazek a W_y. Tato vlastnost hashových řetězců a PayWordu – agregace plateb – usnadňuje práci brokerovi, který může zpracovat prodej několika položek jako jedinou transakci.

Toliko k základní struktuře protokolů PayWordu bez jakýchkoli volitelných drobných modifikací, které autoři navrhli pro demonstraci adaptability schématu. Proti jiným existujícím off-line schématům mluvil jeden či více následujících argumentů:

- počítají s použitelností časových razítek (např. [8]);
- neimplementují samy o sobě absolutní limit na utrácení v součtu přes všechny prodejce;
- jsou v nějakém smyslu neohrabané, výpočetně náročné či neefektivní (např. [7, 5, 3]);
- vyžadují infrastrukturu nerealizovatelnou v našem reálném kontextu ([2] a jiná).

Pro PayWord pak hovoří především výpočetní efektivita, (z hlediska praktické realizovatelnosti) nemožnost vytvářet mince „načerno“ (neoriginální, ale platné) a nemožnost dvojího zaplacení stejným platebním prvkem (zde pravděpodobně nikoli mincí) u jednoho prodejce. Má samozřejmě i nedostatky, z nichž některé budou podrobně vysvětleny v druhé půli aktuální kapitoly a o některých se vyslovují i jiní autoři. Např. [1] identifikuje následující neduhy PayWordu rozšířeného o některé vlastnosti, které jsou esenciální pro praktickou užitelnost schématu a které navrhli již autoři v [7].

- Předpokládejme, že součástí certifikátu zákazníka je i údaj o limitu na utrácení u jednoho prodejce za dobu platnosti certifikátu. Dále předpokládejme, že broker disponuje uživatelovým majetkem, který o mnoho nepřesahuje zmíněný limit. Pak zákazník může utratit víc peněz, než kolik je z jeho majetku posléze schopný vydat broker prodejcům, a to tím, že u několika prodejců utratí částku blízkou limitu.
- Předpokládejme, že je v existenci seznam zablokovaných uživatelů (blacklist), který je rozšiřován od brokera k prodejcům jednou denně. I potom lze však vykonat útok uvedený v předchozím bodě, a to po dobu jednoho dne.
- Politika může být nastavena tak, že přestože není uživatelův absolutní limit na utrácení u všech prodejců v součtu uveden na jeho certifikátu,

a sdělen tak prodejcům, broker začne odmítat proplácet prodejcům mince v momentě, kdy vyčerpá (prodejcům neznámý) absolutní limit pro dané období. Znamená to, že právě prodejci budou tratní, aniž by tomu mohli zabránit, pokud uživatel provede útok popsaný v prvním bodě. Broker pak může na prodejce zaútočit tím, že vytvoří fiktivního uživatele, na jehož účet nakoupí u různých prodejců zboží v celkové hodnotě XY a při sběru mincí pak odmítne proplatit z uživatelova účtu částku celkově vyšší než AB, kde $XY \gg AB$.

Vývoj našeho PS formou modifikování PayWordu se velmi dlouho nesl v duchu udržování zajímavé ideje reprezentace ceny hashovým řetězcem v centru pozornosti. Nicméně představa, že sám zákazník generuje mince, se jevila jako nepřijatelná dle obecných bezpečnostních zásad – hovoříme nyní o době, kdy se při vývoji mnohem více dbalo o předcházení a detekci možných následků hackování uživatelských čipových karet. Úloha generování mincí byla proto delegována na brokera. Ten měl vytvořit pro každého zákazníka řetězec (či řetězce) takové délky, která vždy bude dostačující a nahrávat řetězce na čipové karty při pravidelných návštěvách uživatelů u brokera – shodně s expirací uživatelských certifikátů, které měly mít platnost v řádu jednotek týdnů.

K implementaci limitu na utrácení uživatele u všech prodejců v součtu za dané období, tj. další důležité vlastnosti, kterou PayWord postrádal, byla změněna sémantika hashového řetězce: čipová karta by nepoužívala jeden řetězec pro každou transakci (či, v krajním případě, pro každého prodejce) zvlášť, ale jediný dlouhý řetězec pro veškeré transakce u všech prodejců. Čipová karta by formou výše popsaného paywordovského „přihazování“ jednoduše postupovala po tomto řetězci směrem od kořene tak, jak by uživatel nakupoval střídavě u různých prodejců. Každá mince měla obsahovat mimo jiné

- poslední článek, který karta použila v bezprostředně předcházející transakci – ten se tím stává pomyslným kořenem pro podřetězec aktuální nadcházející transakce;
- článek vzdálenější od skutečného kořene, než je výše zmíněný pomyslný aktuální kořen; tento vzdálenější článek pak reprezentuje částku zaplacenou při aktuální transakci.

Důsledkem úprav popsaných v předchozích dvou odstavcích bylo, že broker znal všechny platné hashové řetězce v oběhu, a to v celé jejich délce, a mohl tak nejen detekovat hacknutou čipovou kartu, která se pokusila o utracení jedné subsekvence z řetězce dvakrát (u různých prodejců), ale mohl rovněž chronologicky uspořádat transakce každého uživatele: stačilo by jednotlivé subsekvence, které brokerovi vrací prodejci, uspořádat do úplného hashového řetězce. Na druhou stranu by ovšem prodejce, kterého zákazník navštíví alespoň dvakrát, mohl v omezené míře zjistit, zda zákazník navštěvuje i jiné prodejce a kolik u nich utrácí.

Uvedený systém s jedním řetězcem, který by umožnil uživateli zaplatit jakékoli povolené množství peněz po dobu až několika málo měsíců, však narážel na přílišnou výpočetní složitost. Kdyby byl hashový řetězec, který broker předkládal uživateli, reprezentován trojicí (W_0, W_n, nominální hodnota jednoho kroku v řetězci), uživatel by musel při prvním nákupu vykonat $(n - x)$ hashovacích cyklů nad W_n, kde x je počet členů řetězce potřebných k zaplacení. Takové množství operací by čipové kartě zabralo nesmírné množství času, pokud uvážíme, že nominální hodnota jednoho kroku v řetězci musela být stanovena dost nízko na to, aby bylo možné prodávat mikrohodnotové položky. Téměř stejné množství hashovacích cyklů by bylo nutné provést i při druhé operaci, o něco menší množství při třetí atd.

Tento problém byl vyřešen předpočítáním mezilehlých článků v řetězci brokerem a uložením těchto mezičlánků uživateli na kartu spolu s celým novým řetězcem. Jediný dlouhý řetězec byl dále nahrazen čtyřmi řetězci, z nichž měl každý jinou nominální hodnotu jednoho kroku: 0,05, 0,5, 5, resp. 20 Kč. Klíčové datové struktury PS pak vypadaly dle následujícího přehledu:

- mince = (závazek, start_1, konec_1, start_2, konec_2, start_3, konec_3, start_4, $\text{konec}_4)_{[Z]}$;
- závazek = ((kořen řetězce$_1$, kořen řetězce$_2$, kořen řetězce$_3$, kořen řetězce$_4$, identifikátor$_P$, identifikátor$_Z)_{[P]})_{[Z]}$;
- kořen řetězce$_1$ = (identifikátor$_B$, 0,05 Kč, $W_{0,1})_{[B]}$;
- kořen řetězce$_2$ = (identifikátor$_B$, 0,5 Kč, $W_{0,2})_{[B]}$;
- kořen řetězce$_3$ = (identifikátor$_B$, 5 Kč, $W_{0,3})_{[B]}$;
- kořen řetězce$_4$ = (identifikátor$_B$, 20 Kč, $W_{0,4})_{[B]}$;
- řetězec$_1$ = (kořen řetězce$_1$, ukazatel na posledně použitý článek, $W_{50,1}$, $W_{100,1}$, $W_{150,1}$, …, $W_{n,1}$);
- řetězec$_2$ = (kořen řetězce$_2$, ukazatel na posledně použitý článek, $W_{50,2}$, $W_{100,2}$, $W_{150,2}$, …, $W_{n,2}$);
- řetězec$_3$ = (kořen řetězce$_3$, ukazatel na posledně použitý článek, $W_{50,3}$, $W_{100,3}$, $W_{150,3}$, …, $W_{n,3}$);
- řetězec$_4$ = (kořen řetězce$_4$, ukazatel na posledně použitý článek, $W_{50,4}$, $W_{100,4}$, $W_{150,4}$, …, $W_{n,4}$).

Kde

- $(\dots)_{[X]}$: struktura digitálně podepsaná entitou X (Z – zákazník, P – prodejce, B – broker);
- mince: struktura nejvyšší úrovně reprezentující platbu;

- start$_x$: článek řetězce$_x$, který byl při předchozí transakci použit jako poslední;
- konec$_x$: článek řetězce$_x$ vzdálenější od jeho kořene než start$_x$; reprezentuje cenu zaplacenou prodejci;
- ukazatel na posledně použitý článek: je nastaven na aktuální konec$_x$ bezprostředně po skončení každé transakce; iniciálně ukazuje na kořen řetězce;
- žádný z řetězců$_i$ není sdělen prodejci.

Zamýšlený efekt nasazení několika řetězců, tj. redukce časové složitosti nákupní operace, se však nedostavil. Vyplývá to z faktu, že v krajním případě může zákazník chtít nakoupit mnoho předmětů či služeb o ceně 0,45 Kč, a tak délka řetězce$_1$ musela být alespoň $(L/0,05)$, kde L je uživatelův limit na utrácení v sumě přes všechny prodejce. Takový řetězec sám by stačil na pokrytí jakýchkoli myslitelných transakcí, které by uživatel v mezích limitu mohl chtít provést. Uvedené schéma by mohlo trpět i nepříjemně dlouhým trváním jedné transakce: to by mohlo překročit časový limit, který by uživatel bezkontaktní čipové karty považoval za pohodlný.

Schéma bylo po dalších úpravách angažujících méně hashování, ale více asymetrické kryptografie, opuštěno jako příliš složité a v druhé polovině loňského roku došlo k dramatické proměně PS tím, že z něj byly hashové řetězce vytlačeny zcela. Doposud popisovaná anabáze spějící do tohoto mrtvého bodu zde má za cíl demonstrovat, jakým způsobem probíhá vývoj podobného systému a naznačit, že slepých cest bylo mnoho – pokud je vůbec možné je v běhu myšlenek izolovat a jednotlivě uchopit – a vývoj proto trvá déle, než se může na první pohled zdát očekávatelné.

Od čistého PayWordu k PS, jak bylo prozatímně specifikováno v čase psaní tohoto příspěvku a jak bude načrtnuto v následující kapitole, spěje však relativně dobře definovatelný most stojící na dvou fundamentálních myšlenkách, které stojí za zvážení, kdykoli se bude jednat o návrh off-line PS s relativně bohatou množinou vlastností.

Tou první je optimalizace na nežádoucím místě: silným bodem efektivity PayWordu je ona agregace mnoha malých plateb do jedné: dostane-li broker k proplacení dvojici (kořen, článek řetězce), neví, zda si zákazník u daného prodejce pořídil jednu nákladnější věc naznačené ceny XY, anebo několik levnějších věcí v celkové ceně XY. Toto je vlastnost, která z PayWordu dělá schéma vskutku mikroplatební – agregace redukuje složitost zpracování transakce na straně kupujícího, prodejce, ale hlavně brokera. Pokud by naopak broker zpracovával v extrému každou jednu prodanou položku jako oddělenou transakci, mohl by být výsledný systém roven mnoha a mnoha nízkohodnotovým bankovním převodům (položíme-li brokera rovného bance) se standardní režijní cenou, která se na jeden takový bankovní převod peněz váže.

Nicméně umět rozlišit i na straně brokera jednotlivé prodané předměty či služby, je vlastnost, která se ukázala jako nepostradatelná. Tento závěr nevyplývá z diskuzí o tvaru schématu samotného, jakožto spíše z průběžného ujasňování scénářů z reálného světa, ve kterých se subjekty používající PS mohou ocitnout. Efektivita v tomto směru tedy dala přednost rozšíření jiné množiny deviz PS. Důsledkem je níže prezentované schéma, které už hashový řetězec vůbec nepoužívá a elektronické mince se podobají spíše šekům v tom, že

- jsou vázané na konkrétního vydavatele;
- jsou vázané na konkrétního příjemce;
- mají vyznačenu pevnou, ale libovolnou cenu.

Druhou klíčovou myšlenkou přemosťující PayWord a vlastní schéma je autentizace prodejce nakupujícímu. PayWord chápe záležitost tak, že nezáleží na tom, kdo se vydaného závazku a příslušného článku řetězce zmocní a přinese je brokerovi – broker vždy dá peníze prodejci vyznačenému na závazku. A z toho vyplývá, že každý prodejce je při prodeji motivován kontrolovat, že je na závazku správně identifikován právě on.

Ve skutečnosti se tato volná identifikace prodejce však nesnáší s rozšířením základní kostry PayWordu o brokerem uvalené limity na utrácení daného uživatele za časovou jednotku u jednoho každého prodejce. Minimálně uživatelova čipová karta si bude muset interně udržovat povědomí o tom, kolik peněz držitel už utratil v daném časovém okně u jednotlivých prodejců, aby měl dotyčný vždy přehled o tom, kolik ještě může utratit a aby karta mohla případné limit přesahující transakce zamítnout (v případě, že je tato zodpovědnost delegována mj. na kartu). Prodejce ABC pak může znemožnit držiteli karty nakupovat u prodejce XYZ tím, že se za XYZ bude lživě vydávat. Čipová karta se bude domnívat dříve, než to odpovídá realitě, že držitel už u XYZ utratil limitní množství peněz. Tomuto útoku se naše PS brání řádnou autentizací prodejců pomocí certifikátu při každé transakci.

4 Aktuální znění protokolů PS

V následujícím přehledu protokolů PS bude opět abstrahováno na úroveň virtuálního zákazníka (Z), prodejce (P) a brokera (B); je tedy upuštěno od ohledu na jejich počet. Ani zde nebude specifikováno, jakým protokolem nižší úrovně konkrétně bude probíhat komunikace mezi zúčastněnými stranami, jak přesně budou na binární úrovni formátovány předávané zprávy atp.

Každý prodejce se zařadí do systému registrací u brokera(ů):

1. $P \to B$: $(\mathrm{id}_P, \mathrm{vklíč}_P)_{[P]}$
2. $B \to P$: $\mathrm{cert}_P = (\mathrm{id}_P, \mathrm{vklíč}_P, \mathrm{id}_B, \mathrm{expirace})_{[B]}$

Kde

- id_P: identifikátor prodejce pochopitelný pro člověka;
- $\mathrm{vklíč}_P$: veřejný klíč prodejce (soukromý klíč si prodejce ponechá v tajnosti);
- cert_P: certifikát prodejce, kterým broker stvrzuje, že jeho držitel má dané jméno a klíč a je oprávněn účastnit se prodeje po vyznačenou dobu platnosti;
- id_B: brokerův identifikátor, který v prostředí více brokerů umožní zákazníkovi vybrat správný klíč k ověření podpisu na cert_P;
- expirace: doba, do níž je certifikát v platnosti; je rovna času vydání certifikátu plus C_P (poločas životnosti mince).

Zákazník iniciálně podstoupí analogickou proceduru:

1. $Z \to B$: $(\mathrm{id}_Z, \mathrm{vklíč}_Z)_{[Z]}$
2. $B \to Z$: $\mathrm{cert}_Z = (\mathrm{id}_Z, \mathrm{vklíč}_Z, \mathrm{id}_B, \text{expirace}, \text{limit})_{[B]}$

Kde

- cert_Z: certifikát zákazníka, kterým broker stvrzuje, že disponuje jeho reálnými financemi v dostatečném množství, a je tedy schopen proplácet mince vydané tímto zákazníkem;
- expirace: limit platnosti cert_Z; je na prodejci, aby zkontroloval platnost při prodeji, neboť broker bude schopen rozpoznat mince vydané uživatelem s prošlým cert_Z a odmítne prodejci tyto mince proplatit;
- limit: množství peněz, které může zákazník utratit u každého z prodejců (zvlášť), a to v součtu během jedné periody C_P; je rovněž na prodejci, aby neumožnil utratit žádnému ze zákazníků více, neboť broker by přebytečné mince neproplatil;
- id_Z: identifikátor zákazníka; jeho forma nebyla doposud stanovena a není vyloučeno, že bude ze všech zde zmíněných datových struktur odstraněn, neboť dotyčného jednotlivce lze (pokud to nevyloučí jiné požadavky na systém) jednoznačně identifikovat i $\mathrm{vklíčem}_Z$ a zároveň není důvod žádné další informace o držiteli sdělovat prodejcům, kterým bude v běžném provozu držitel certifikát předkládat.

Expirace zde má podružnější roli: očekává se, že bude nastavena na dobu v řádu roků, po níž bude držitel certifikátu nucen navštívit brokera a sjednat prodloužení platnosti certifikátu, čímž vyjádří svůj trvající zájem o svou existenci v systému. Broker dostane zároveň pohodlnou příležitost přehodnotit, jakou expirační dobu a limit na utrácení dotyčnému stanovit pro další certifikát,

příp. zda jej ze systému vyloučit. Když se o prodloužení platnosti držitel dlouhou dobu nepřihlásí, může to broker brát jako příležitost k jeho odstranění ze systému a pročištění databází.

Naproti tomu limit na utrácení stanovuje buď sám broker, nebo broker ve spolupráci s držitelem certifikátu jako zamezení příliš vysoké finanční ztrátě v případě, že je ukradena a zneužita (k nákupům) čipová karta, na níž je certifikát spolu s privátním klíčem držitele uložen. V diskuzi je i přidání dalších limitů do cert_Z, např. limit na utrácení daného uživatele v součtu u všech prodejců.

Vlastní nákup zboží v praxi zahajuje zákazníkova čipová karta zkontaktováním prodejního automatu či jiného zařízení, které prodej uskutečňuje a reprezentuje prodejce. Poté, co je vybráno zboží ke koupi (jedna položka), vygeneruje čipová karta minci, kterou zaplatí:

1. $Z \to P$: výzva
2. $P \to Z$: cert_P, $(\text{výzva, cena})_{[P]}$
3. $Z \to P$: mince $= (\mathrm{cert}_Z, \mathrm{cert}_P, \text{cena}, \mathrm{id}_M)_{[Z]}$

Kde

- výzva: náhodný řetězec unikátní pro každou transakci; je-li schopen jej prodejce korektně digitálně podepsat, což čipová karta zákazníka kontroluje před krokem 3, je prodejce autentizován;
- id_M: unikátní identifikátor mince.

Účelem výměny výzvy, resp. jejího podepsaného tvaru (kroky 1 a 2), je zabránit prodejci vydávat se za konkurenci, jak bylo popsáno výše. Mince je pak klíčovou strukturou v celém PS. Broker z ní extrahuje, že to byl zákazník identifikovaný cert_Z, kdo nakupoval a že utratil vyznačené množství peněz u prodejce identifikovaného cert_P. Tím je známo, kolik peněz odkud kam převést. Podpis na minci znemožňuje zákazníkovi popřít platbu a požadovat převedené peníze zpět. Unikátní identifikátor naopak znemožňuje prodejci nechat si proplatit minci dvakrát a tvrdit, že stejný zákazník u něj vícekrát zakoupil zboží ve stejné ceně. Přestože je tedy id_M generován zákazníkem, je prodejce motivován kontrolovat, že již dříve minci s tímto identifikátorem nepřijal, neboť broker proplatí jednomu prodejci jen jednu minci se stejným id_M. Nestačí ani, když se nová mince liší v ceně a cert_P (s jiným obdobím platnosti) – viz dále.

Pro naplnění pouhého cíle rozlišení dvou mincí by stačilo, aby bylo id_M bráno postupně z prosté sekvence přirozených čísel, anebo vždy náhodně vygenerováno. První přístup přináší brokerovi lehkou auditní informaci o tom, v jakém pořadí uživatel platby prováděl, a z tohoto důvodu by mohlo být žádoucí, aby čipová karta skutečně svědomitě generovala id_M z plynulé sekvence, a to nehledě na to, že žádná dříve vydaná mince nemá stejnou naznačenou cenu a cert_P. Druhý přístup naopak dává zákazníkovi vyšší úroveň anonymity vůči prodejcům: žádný

z prodejců není schopen ze všech mincí od daného zákazníka vydedukovat, zda dotyčný navštěvuje i konkurenci a kolik položek tam nakupuje.

Spojení výhod obou přístupů lze získat jejich zkombinováním za použití šifry: nechť je při každé transakci vypočítáno id_M takto:

1. vygeneruj náhodný řetězec;
2. připoj na jeho konec pořadové číslo transakce (bráno z prosté sekvence přirozených čísel);
3. výsledek zašifruj veřejným klíčem brokera a použij jako id_M.

5 Budoucnost projektu závěrem

Jak lze vytušit z dosavadního výkladu, vývoj PS probíhá doposud „in vitro“; bez modelové implementace. Nové nápady a zásadnější modifikace se stále objevují relativně často. Širší pozornost bude věnována mj. následujícím oblastem:

- popis reakce na situaci, kdy je prolomen některý z kryptografických algoritmů použitých ve PS v živém systému;
- specifikace protokolů kolem blokování (blacklistingu) certifikátů uživatelů a prodejců;
- úprava protokolů a datových struktur tak, aby obsáhly situaci, kdy je jeden zákazník svázán s více brokery;
- reakce na stav, kdy uživatel vydá u dvou různých prodejců minci se stejným id_M; bude se však pravděpodobně jednat pouze o zásah vně protokolů PS (uživatel nebude a priori podezříván z úmyslné neoprávněné modifikace čipové karty a broker mu nabídne či vnutí náhradní kartu);
- reklamace zboží (bezprostředně po koupi);
- formálnější specifikace protokolů v pseudojazyce identifikující důsledně chybové a nestandardní stavy;
- specifikace protokolů PS i v souladu s alternativní sadou konfigurací rolí.

Naposled zmíněné konfigurace rolí jsou problematikou, která nebyla z prostorových důvodů do tohoto příspěvku zahrnuta. Jedná se o přehled všech reálně smysluplných mapování tří rolí v PS (prodejce, zákazník, broker) na skutečné subjekty (osoby, firmy, ...) a tato mapování – konfigurace – jsou seskupena do dvou sad. V příspěvku byla automaticky brána v úvahu pouze ta sada, kde je cert_Z každého jednotlivého uživatele vybaven jiným, unikátním klíčem, takže čipová karta jednoznačně reprezentuje svého držitele. V alternativní sadě konfigurací je klíč identický pro všechny členy nějaké skupiny (např. zaměstnance jedné firmy, studenty jedné školy). V případě zaměstnanců stejné firmy tak lze

řešit např. volné platební karty s limitem na utrácení, které rozdá zaměstnavatel v rámci zaměstnaneckých výhod. Jiným důvodem pro nasazení může být anonymita uživatelů vůči prodejcům: ti se dozví, že u nich nakupoval zaměstnanec dané firmy, ale nebude možné říci, který to byl. Tato informace, která bude uložená jako dodatečné pole v zašifrovaném tvaru v mincích, bude ovšem dekódovatelná brokerem.

Vývoj PS jako takového je jednou ze součástí širšího projektu, který dále obnáší:

- bližší rozvahu a implementaci zabezpečeného tiskového serveru a SW pro samotnou tiskárnu, která bude prodávat uživatelům tiskové úlohy (právě jejich prodej byl motivačním impulzem pro start projektu);
- způsoby autentizace osob veškerým zařízením, které ji mohou vyžadovat;
- právní aspekty provozu celého systému (ve spolupráci s Právnickou fakultou Masarykovy univerzity);
- obhájené i budoucí diplomové a bakalářské práce týkající se převážně tiskového subsystému a autentizace uživatelů.

V současné době, souběžně s pokračujícím vývojem PS, začínáme zvažovat i jeho implementaci. Veškeré protokoly, které byly doposud řešeny pouze obecně, jsou zpětně analyzovány a probíhá výběr konkrétních algoritmů a technologií. Pro ilustraci si ukážeme pár problémů, kterými se v tomto procesu zabýváme, na protokolu nákupu:

1. $Z \to P$: výzva
2. $P \to Z$: cert_P, (výzva, cena)$_{[P]}$
3. $Z \to P$: mince = (cert_Z, cert_P, cena, id_M)$_{[Z]}$

Hned zpočátku je třeba si uvědomit, že veškerá komunikace mezi zákazníkem (Z) a prodejcem (P) probíhá bezdrátově. Není tedy problém tuto komunikaci odposlouchávat a následně si ji v lepším případě pouze přečíst nebo v horším případě ji použít k útoku na zákazníka (odposlechnutí výzvy a následná odpověď v kratším čase než odpoví prodejce). Je tedy třeba vyřešit otázku zabezpečení komunikace, což je záležitost, kterou se PS samotné nezabývá. Dalším problémem je implementace certifikátů všech zúčastněných stran a výběr algoritmu pro podpisy, které PS ve svém návrhu také nediskutuje. V neposlední řadě bude důležitým úkolem návrh zařízení, které bude reprezentovat prodejce a brokera.

Stejně jako je v rámci PS zákazník zastoupen čipovou kartou, budou broker a prodejce reprezentováni zařízeními označovanými *terminály*. Terminály brokera a prodejce se od sebe budou lišit na základě požadované funkčnosti a zabezpečení. Terminál prodejce budeme nazývat *prodejní terminál*. Požadavky na něj budou následující:

- připojen do sítě, ale po většinu času bude off-line nebo pouze ve spojení sáinfrastrukturou prodejce;
- výpočetním výkonem a paměťovou kapacitou bude ekvivalentní levnému počítači;
- schopen bezdrátové komunikace s čipovou kartou zákazníka;
- disponuje vstupním zařízením a displejem pro přímou komunikaci se zákazníkem;
- osazen čipovou kartou či podobným zařízením pro bezpečné uchování certifikátů a základní kryptografické operace.

Terminál, který bude zastupovat brokera, budeme nazývat *zabezpečený terminál*. Požadavky na něj budou velmi podobné jako na terminál, pouze výkon bude srovnatelný s běžným počítačem a vstupní zařízení a displej budou primárně pro obsluhu terminálu. Ta se bude účastnit některých úkonů jako například registrace nového zákazníka nebo vydání certifikátu. Zabezpečený terminál bude zajišťovat následující úkony:

- registrace zákazníka;
- vydání certifikátu zákazníkovi;
- odvolání certifikátu zákazníka.

Zabezpečené terminály budou běžně k nalezení v trafikách (podobně jako například loterijní terminály) či přímo na pobočce prodejce, který se na tomto s brokerem dohodne.

Dále bude třeba implementovat také funkce týkající se správy prodejců. V tuto chvíli není zřejmé, zda tyto budou řešeny pouze organizačními opatřeními (návštěva prodejce na pobočce brokera, apod.) a zabezpečeným spojením s brokerem, nebo zda tyto funkce bude zajišťovat další typ terminálu. Funkce, o které se jedná, jsou tyto:

- registrace prodejce;
- vydání certifikátu prodejci;
- výkup mincí od prodejce;
- odvolání certifikátu prodejce.

Ať už budou funkce implementovány jako nový typ terminálu nebo pouze organizačními pokyny, dostupné budou pouze na pobočkách brokera.

Předpokládá se, že vývoj PS ještě pár let potrvá a že jeho implementace bude prováděna postupně v symbióze s vývojem; stejně tak i jeho následné zavádění do praxe. Posláním tohoto příspěvku bylo nejen představit projekt, který má očekávanou hodnotu pro budoucnost, ale i požádat publikum o komentáře a návrhy, které mohou přispět ke zdokonalení platebního systému, jehož specifikace bude po dokončení veřejně využitelná a publikovaná.

Poděkování

Ze strany firmy Y Soft s. r. o. se o projekt stará Ondřej Krajíček, který nejen že plní roli zákazníka-konzultanta a dohližitele nad tím, co vzniká za firemní finance, ale nad tento rámec své povinnosti na některých brainstormingových sezeních rovněž přispívá radami i samotnému vývoji PS. Děkujeme Ondrovi za jeho čas.

Literatura

[1] Adachi, N., Komano, Y., Aoki, S., Ohta, K. *The Security Problems of Rivest and Shamir's PayWord Scheme.* 2003 IEEE International Conference on E-Commerce Technology, 2003. URL http://www.computer.org/portal/web/csdl/abs/proceedings/cec/2003/1969/00/19690020abs.htm (březen 2010).

[2] Catalano, D., Ruffo, G. *A Fair Micro-Payment Scheme for Profit Sharing in P2P Networks.* In Proceedings of the 2004 International Workshop on Hot Topics in Peer-to-Peer Systems, 2004. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1376613&isnumber=30046 (březen 2010).

[3] Glassman, S., Manasse, M., Abadi, M., Gauthier, P., Sobalvarro, P. *The Millicent Protocol for Inexpensive Electronic Commerce.* Fourth International World Wide Web Conference, 1995.
URL http://www.w3.org/Conferences/WWW4/Papers/246 (březen 2010).

[4] Knorr, E., Gruman, G. *What cloud computing really means.* InfoWorld, 2009.
URL http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031 (duben 2010).

[5] Lee, T. O., Yip, Y. L., Tsang, C. M., Ng, K. W. *An Agent-Based Micropayment Scheme for E-Commerce.* Department of Computer Science & Engineering, The Chinese University of Hong Kong.
URL http://www.springerlink.com/content/f3fhbe7tkc72kul2/fulltext.pdf (březen 2010).

[6] Piják, M. *Mikroplatby – jsou nezbytné?* Měšec.cz, 2003.
URL http://www.mesec.cz/clanky/mikroplatby-jsou-nezbytne (březen 2010).

[7] Rivest, R. L., Shamir, A. *PayWord and MicroMint: Two simple micropayment schemes.* 2001.
URL http://people.csail.mit.edu/rivest/RivestShamir-mpay.pdf (březen 2010).

[8] Shao, X., Nguyen, A. T., Muthukkumarasamy, V. *WebCoin: A Conceptual Model of a Micropayment System.* AusWeb04, 2004.

URL http://ausweb.scu.edu.au/aw04/papers/refereed/nguyen2/paper.html (březen 2010).

[9] Skorobogatov, S. P. *Semi-invasive attacks – A new approach to hardware security analysis.* Computer laboratory, University of Cambridge, 2005. URL http://issuu.com/flavio58/docs/ucam-cl-tr-630 (březen 2010).

Migrace na IPv6 (s firewallem Kernun)

Peter Pecho

E-mail: peter.pecho@tns.cz

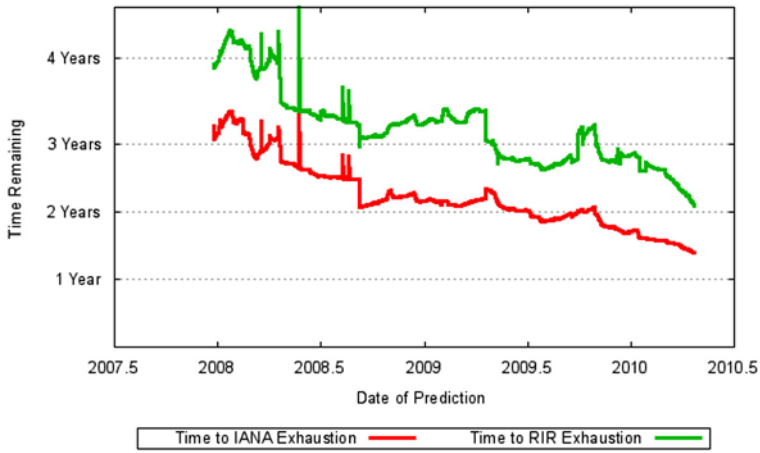
Abstrakt

Problém vyčerpávání volných IPv4 adres urychluje potřebu migrace současných sítí na protokol IPv6. I přes světové prvenství ČR v počtu přidělených IPv6 autonomních systémů k celkovému počtu IPv4 autonomních systémů je zatím pouze malé procento sítí dostupných přes tento protokol. Cílem článku je objasnit možná úskalí a zjednodušit společnostem přechod na IPv6. Článek se dále zaměřuje na problematiku implementace IPv6 do uživatelských a serverových operačních systémů, síťových prvků, běžných síťových aplikací, běžných uživatelských aplikací a mobilních zařízení. Dále jsou popsány odlišnosti síťového provozu důležité z hlediska bezpečnostní politiky a provozu v síti. Na závěr se článek věnuje metodám vhodným pro postupný přechod na nový protokol a zpřístupnění služeb klientům připojeným přes odlišný protokol.

1 Úvod

Dramatické rozšíření Internetu v posledních 20 letech způsobilo řadu problému s nimiž se setkáváme také v současnosti. Část řešení byla vyřešena novými portacemi vlastností z protokolu IPv6 do IPv4, zejména jde o IPsec, mobilitu, atd. Nejvíce zmiňovaný problém, nedostatek IP adres, se pomocí dodatečných technologií (NAT, CIDR) stejně tak podařilo oddálit o řadu let. V současnosti ovšem nastává doba, kdy všechny dočasné „malé“ řešení přestávají být účinné a je důležité vykonat velké rozhodnutí.

Jedním z důvodů současného stavu je neefektivní přidělování IP adres v počátcích Internetu. Velkým společnostem byli přidělované neúměrně velké adresní rozsahy, které neměli možnost využít. Jako příklad je možné uvést univerzitu ve Stanfordu, univerzitu MIT, nebo společnost Xerox, jež měli od začátku přidělené IPv4/8 adresy. Tyto organizace mohly ovšem pouze s velkými problémy využít tyto miliony veřejných IP adres. Opačný problém měla např. Čína, která musela zpočátku vystačit s adresami třídy B a které byl stejný rozsah IP adres, jako



Obr. 1 Model vyčerpávání adresného prostoru IPv4

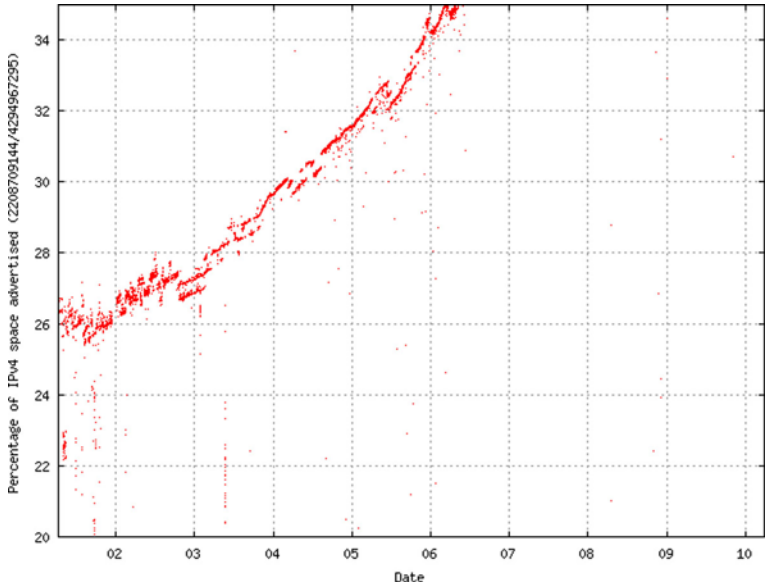
zmíněným organizacím, přidělený až v roce 2000. V následujícím období některé z organizací tyto bloky IP adres vrátily a ty byly opětovně využity. Mnohé ze společností těmito rozsahy adres disponují dodnes (např. Xerox a MIT) [3, 12].

V současnosti je již zřejmé, že změnám ve struktuře a fungování Internetu se nevyhneme a nastanou poměrně brzo. Organizace IANA, jako nejvyšší autorita přidělování adres, očekává vyčerpání volných rozsahů v průběhu následujícího roku 2011 a regionální registrátoři o rok později [8]. Po vyčerpání adres z IPv4 rozsahu nebudou mít nové zařízení a veřejně poskytované služby jinou možnost připojení než přes IPv6. Pro zbytek Internetu, jenž bude požadovat komunikaci s těmito zařízeními, bude nevyhnutné implementovat taktéž IPv6 nebo použít specializované služby pro překlad adres.

2 Využití adresného prostoru IPv4

Globální koordinátor přidělování IP adres, organizace IANA, pravidelně zveřejňuje statistiky přidělování IPv4 adres, na jejichž základě je možné předpovědět datum vyčerpání celého adresného prostoru. Na základě současných statistik nastane vyčerpání volných adres přidělovaných organizací IANA lokálním registrátorům dne 27. 9. 2011. Lokální registrátoři přidělí poslední volné adresy přibližně o rok později, viz obrázek 1 [8].

Přehled o využití adresného prostoru IPv4 je možné získat z páteřních směrovačů. Na základě oznamovaných adresných rozsahů a úplných adresných rozsahů, které mají směrovat, je možné spolehlivě zjistit využívání IPv4 adres. Jako pří-



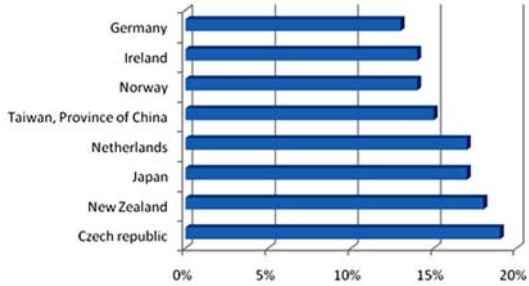
Obr. 2 Statistika využití IPv4 adres na páteřním směrovači APNIC

klad statistiky uvádím informace ze směrovače AS65000 regionálního registrátora APNIC. Poměr oznamovaných adres k přiděleným adresám je v tomto čase psaní článku 51,4%. Dlouhodobě je tento trend lineárně rostoucí (viz obrázek 2) [7].

3 Obchodování s IP adresami

I po vyčerpání IPv4 adres budou stejně tak některé adresné rozsahy nevyužité. Jde zejména o adresy, které byly přiděleny na počátcích Internetu a nebyly příslušnými organizacemi plně využity, případně již tyto rozsahy byly úplně opuštěny. Jejich opětovné využití bude možné pomocí změn v topologii, změnám ve směřování a vytvoření trhu s adresami.

Původní směrnice organizace IANA, jež zamezovaly nakládání s IP adresami, jako s komoditou jsou již částečně historií. Evropa zrušila zákaz obchodování s adresami v roce 2008 a Spojené státy k tomuto kroku přistoupily v červnu 2009. Důvodem povolení trhu s IP adresami je snaha o dostání volných rozsahů adres do oběhu. Regionální registrátoři již v současnosti diskutují o obchodování s adresami a navrhují modely tohoto nového trhu. Některé z nich (např. JPNIC) umožní přímý prodej adres prostřednictvím poskytovatelů Internetu a příbuzných společností a sama se nebude podílet na nákupu a prodeji. Již v současnosti



Obr. 3 Poměr IPv6 autonomních systémů k IP4 autonomním systémům

je možné očekávat urychlování migrace na IPv6 vzhledem k nepředvídatelnému vývoji cen IPv4 adres. Mnohé organizace se obávají stejného scénáře jako nastal před pár lety se skupováním doménových jmen. Zde ovšem hrozí také technické komplikace. Prodej IPv4 adres mezi oblastními registrátory bude nemožný, nebo alespoň bude problematický, vzhledem k hrozícímu nárůstu složitosti směřování na páteřních směrovačích [2, 5, 13].

4 Poskytování IPv6 u místních ISP

IPv6 peering, nebo-li propojení ISP v ČR, je na vysoké úrovni. Z celkového počtu 97 organizací registrovaných u NIX.CZ disponuje již 39 z nich IPv6 peeringem [10]. Nezávislý report organizace BGPMON přináší ještě pozitivnější výsledky. Na základě porovnání počtu IPv6 autonomních systémů vůči všem IPv4 autonomním systémům porovnávala zmíněná organizace připravenost jednotlivých zemí na přechod na IPv6. Jelikož se v první desítce ocitly pouze malé země s jednotkami autonomních systémů, byly v upravené statistice ponechány pouze ty, které měly nejméně 100 autonomních systémů. V této statistice se objevila Česká republika (19 %) na prvním místě, následovaná Novým Zélandem (18 %), Japonskem (17 %) a Nizozemskem (17 %) – viz obrázek 3 [4].

Pro představu o současných možnostech připojení k IPv6 jsme udělali vlastní průzkum, ve kterém jsme kontaktovali místní ISP s žádostí o připojení firemní pobočky do IPv6 sítě. Z našeho průzkumu jsme vyloučili společnosti sdružující poskytovatele, správce domén, zahraniční poskytovatele a další organizace, které běžně neposkytují služby koncovým zákazníkům. Část poskytovatelů měla na svých stránkách uvedeny nesprávné kontaktní údaje, příp. jejich stránky již byly nefunkční. Tyto poskytovatele jsme již dále nekontaktovali. Výsledky z našeho průzkumu najdete v tabulce 1.

Z průzkumu vyplývá, že koncoví zákazníci mají pouze omezené možnosti připojení vlastních sítí do Internetu přes IPv6. Pouze 7 ISP odpovědělo kladně na

Tab. 1 Možnosti připojení přes IPv6 u místních ISP

Celkem oslovených ISP	25
Nesprávné kontaktní údaje	2
Bez odpovědi	7
Snaha o získání obchodního kontaktu	3
Testovací provoz IPv6	2
Pouze IPv6 hosting / server hosting	4
Ano	7

možnost tohoto připojení firemní pobočky, dva ISP provozovali IPv6 v testovacím režimu a další tři ISP chtěli pokračovat v obchodních jednáních (což nutně neznamená, že IPv6 skutečně podporují). Pouze dva z těchto poskytovatelů zmiňovali podporu IPv6 na svých webových stránkách. Je tedy otázkou, nakolik jsou dané informace u dalších ISP založené na pravdě.

Další možností připojení vlastních serverů je jejich umístěním do vhodných datových center. Čtyři z oslovených společností poskytují IPv6, avšak některé z nich tento provoz stalé považují za testovací. Při výběru vhodného server hostingu se stejně tak ujistěte, zda IPv6 konektivita je dostupná pouze v Praze nebo i v jiných lokalitách.

Ze zpětné vazby od oslovených organizací taktéž vyplývá, že s požadavky o připojení pomocí IPv6 se setkávají pouze výjimečně., jelikož mnozí z nich museli oslovit své technické oddělení, díky čemuž jsme na odpověď museli čekat několik dní.

5 Úskalí přechodu na IPv6

Jak bude níže popsané, přechod jednotlivých systémů na IPv6 bude jednoduchý. Horší to bude s přechodem celých sítí a komplexních síťových aplikací. Zařízení, jež jsou součástí těchto systémů, nemusí podporovat všechny technologie a protokoly pro jejich jednotné a plošné nahrazení novějšími technologiemi.

5.1 Operační systémy

Přechod koncových stanic na IPv6 bude jednoduchý. Současné verze operačních systémů společnosti Microsoft, stejně jako Linux, BSD a další běžné unixové systémy jsou na IPv6 připraveny. Většina moderních operačních systémů poskytuje několik způsobů připojení k IPv6 včetně přímých připojení a připojení pomocí tunelů.

Operační systémy Windows Vista a Windows 7 jsou tzv. dual-stack operační systémy, které implicitně podporují IPv4 i IPv6, přičemž preferují nový protokol. Podpora protokolu IPv6 přes IPv4 je vyřešená pomocí technologií 6to4, ISATAP a Teredo tunelů. Podporu IPv6 ve Vistách a novějších systémech není možné odinstalovat, je možné pouze zablokování přes editaci registrů. Microsoft si tímto přístupem přímo „vynucuje“ používaní nového protokolu v jeho operačních systémech [14]. Pokud již využíváte IPv6, ať už vědomě nebo nevědomě, je důležité myslet na bezpečnost všech způsobů síťové komunikace. Automatická konfigurace síťových zařízení, jež je součástí IPv6, může způsobit nevědomé připojení uživatele k Internetu, které bude v rozporu z bezpečnostní politikou. Na to je potřebné obzvláště myslet u serverů, zejména když poskytují citlivé služby.

Linux, jakož to nejrozšířenější unixový systém, přišel v roce 1996 jako první s experimentální podporou IPv6 a jako poslední přestal svou implementaci označovat jako experimentální. Současná produkční jádra 2.6 a 2.4 obsahují stabilní implementaci IPv6, doporučuje se ovšem používání nové větve, jelikož větev 2.4 již není opravována podle nejnovějších RFC. Všechny rozšířené distribuce mají podporu IPv6 implicitně zapnutou a není třeba doinstalovat nic dalšího. Linux standardně podporuje 6to4 tunely, podporu jiných tunelů je ovšem stejně možné doplnit. Jako příklad uvádím pouze open source implementaci Teredo tunelu označenou jako Miredo. Kvalitu implementace dokazuje certifikát IPv6 Ready fáze 2 pro koncový stroj i směrovač [11].

BSD systémy poskytují jednu z nejkvalitnějších implementací IPv6. Podpora automatické konfigurace je na rozdíl od výše zmíněných systémů automaticky vypnutá. Důvodem je pravděpodobně časté použití BSD systémů jako směrovačů a serverů poskytujících síťové služby. Přechodové systémy jsou shodné s Linuxem – 6to4, ISATAP a Teredo tunely. BSD systémy stejně jako Linux získali certifikát IPv6 Ready fáze 2 pro koncový stroj i směřovač.

5.2 Serverové aplikace

Základní síťové aplikace (webové servery, mailové servery, atd.) jsou stejně tak připravené na IPv6 nebo budou v blízké době. Vývojáři těchto nástrojů úzce spolupracují s autory operačních systémů, a proto integrace podpory IPv6, příp. souběžná podpora obou protokolů, je samozřejmá a spolehlivá. Jako příklad je možné uvést webový server Apache, jenž podporuje dual-stack a nerozlišuje mezi IPv4 klienty a IPv6 klienty. Stejně tak SMTP server sendmail poskytuje podporu IPv6 již řadu let.

5.3 Směrovače a další síťové prvky

Směrovače a další síťové prvky bude možné stejně tak poměrně jednoduše upgradovat. FreeBSD poskytuje referenční implementaci IPv6 směrovače, podpora

v jiných unixových systémech bude stejně tak na vysoké úrovni. Komerční řešení (např. Cisco) již poskytují upgrade na IPv6 zdarma (a ne za poplatek jak tomu bylo v minulosti).

5.4 Uživatelské aplikace

Největší problém nastane ovšem při úpravě aplikačního softwaru, zejména softwaru na zakázku. Malé a střední softwarové firmy budou mít největší problémy s podporou IPv6 ve svých produktech. Nehledě na vývoj nového softwaru s touto podporou bude potřebné upravit (nebo nahradit) všechny současné aplikace, aby byli schopny komunikace přes IPv6.

5.5 Mobilní aplikace

Při migraci na IPv6 je také potřebné myslet na specifické zařízení, jež neposkytují podporu IPv6 a ani ji poskytovat nebudou. Typicky jde o mobilní zařízení jako mobilní telefony, multimediální přehrávače apod. Tyto zařízení budou v počátku přechodu na IPv6 schované za NAT-em a až časem morálně zastarají a budou nahrazeny. Novější zařízení disponující větším výkonem a prostředky budou pravděpodobně upgradovány pomocí novějšího firmwaru. Čtvrtá generace mobilních telefonních sítí, označovaná též jako 4G, již vyžaduje podporu IPv6 v koncových zařízeních [6].

6 Bezpečnost a provoz na síti

Bezpečnost současných počítačových sítí je řešená pomocí kombinace specializovaných nástrojů: centrálních firewallů, firewallů pro koncové stanice, IDS/IPS systémů, aplikačních proxy, antivirů, antispamů, NATů, honeypotů, atd. Současná řešení poskytují pouze částečnou podporu IPv6 – lépe jsou na tom firewally, hůře IDS/IPS systémy. Z netechnických prostředků je potřebné zmínit také bezpečnostní pravidla. Ty jsou v současných produkčních sítích až na výjimky zaměřené pouze na IPv4 a vzhledem k odlišnostem mezi oběma protokoly nemůžou poskytovat stejnou úroveň ochrany. Typickým příkladem takové konfigurace je filtrace ICMPv4 paketů na rozhraní vnitřní a vnější sítě. Pakety ICMPv6 nesmí být na tomto místě zahazovány, jinak by nebyly zajištěny základní síťové služby: automatické konfigurace, resolvování, správa multicastových skupin a jiné. Funkčnost z IPv4, původně provozována na protokolech TCP/UDP, se totiž přesunula do protokolu ICMPv6. Vhodné nastavení bezpečnostních politik na hraničních a transportních směrovačích je možné najít v různých oficiálních i neoficiálních doporučeních (viz RFC 4890).

Hlavním přínosem IPv6 je rozšíření adresného rozsahu, což přináší kromě dostatku volných adres také zvýšení odolnosti vůči skenování sítě. Implicitní podsítě v IPv6 mají 264 adres, což neumožňuje skenování celého adresného prostoru. Důsledkem je, že viry a obecně libovolný malware založený na skenování sítě nebude mít šanci zjistit strukturu sítě v „rozumné“ době (při vhodném adresování koncových stanic). Vývojáři škodlivého kódu se proto „budou muset“ časem přizpůsobit technologii IPv6. Pokud by útočník hádal pouze posledních 24 bitů rychlostí 1 000 testů za vteřinu, proskenování sítě by mu trvalo více než 4 hodiny.

IPv6 dále poskytuje tzv. „privacy extensions“ (tj. krátkodobé adresy podle RFC 3041), jenž omezují dobu, po kterou je počítač na dané adrese dostupný. Toto rozšíření je vhodné k znehodnocení výsledků skenování pro zjištění struktury sítě. Tyto pseudonáhodné adresy, jež působí jako ideální prostředek pro zajištění soukromí ovšem nejsou vhodné pro komunikaci v rámci interní sítě. Pokud by v rámci této sítě nastaly problémy v komunikací s koncovými stanicemi, těžko by se chyba v takovém prostředí lokalizovala. Krátkodobé adresy jsou ovšem vhodné pro navazování odchozích spojení. Pro příchozí spojení daný počítač využívá trvalou adresu zavedenou v DNS.

Další technologie, IPsec, která byla prvně navržena pro IPv6 a později importována také do IPv4, poskytuje dvě základní bezpečnostní funkce – integritu dat a důvěrnost. Kvůli návrhu, jenž byl těsně svázán s novou generací IP protokolu, se někdy mluví o vyšší bezpečnosti. Toto tvrzení může být ovšem pravdivé pouze v ideálním světě s dokonale navrženými a implementovanými aplikacemi a efektivní správou klíčů. Vybudovat jednotné uložiště klíčů ovšem stále není reálné. Problém je zde však spíše politický než technický.

Překlad síťových adres, známý též jako NAT, který byl široce používán pro efektivní využití adresného prostoru, v IPv6 již není potřebný a původně s ním tento protokol ani nepočítal. Časem se situace změnila a byl zahrnut také do nového protokolu. Důvodem byla snaha o maximální zjednodušení migrace sítí a také jeho pozitivní vliv na bezpečnost sítě. I když není možné považovat NAT za standardní bezpečnostní prvek, stejně tak mu nemůžeme odepřít vliv na utajení struktury interní sítě.

Ponechání původních technologií zjednodušuje migraci sítí, u velkých sítích ovšem není možné očekávat okamžitý přesun všech služeb na IPv6. Je mnohem pravděpodobnější, že podpora IPv4 bude udržována v průběhu migrace a několik let poté. Bezpečnost v daném období bude proto potřebné řešit komplexněji s ohledem na oba protokoly. Stejně tak bude přechod úzce spojen s dual-stack aplikacemi, jenž mohou být náchylné na útoky zaměřené vůči IPv4 i IPv6. Chyby se mohou vyskytovat nejen v IPv6 stackech, ale také v aplikacích, jenž byly portovány na nový protokol. Zejména zpočátku portování je nutné proto myslet na patchování kritických prvků v infrastruktuře.

7 Postupná migrace na IPv6

Podívejme se na několik typických scénářů migrace na IPv6.

7.1 Scenář 1: IPv4 uvnitř, IPv6 venku

Jde pravděpodobně o nejběžnější scénář kdy je IPv4 síť připojena do IPv6 sítě. Cílem je poskytnout vnějším uživatelům služby z interní sítě, stejně jako vnitřním uživatelům, veřejné služby IPv6 Internetu. Jelikož jsou IPv4 a IPv6 navzájem nekompatibilní, není možné přímé spojení dvou entit z různých „světů“. Řešení je založeno na propojení sítí pomocí vhodného prvku schopného překladu adres mezi těmito protokoly. Jedním z možných řešení je použití proxy, ať už generické TCP/UDP nebo proxy pro specifické aplikace. Jelikož jsou proxy charakteristické rozkládáním spojení, není pro ně problém rozložit komunikaci přenášenou jedním protokolem a její složení do jiného protokolu. Je nutné upozornit, že takové řešení je netransparentní, jelikož zařízení v odlišných sítích se navzájem nedokážou přímo adresovat.

Nejprve si ovšem ověřte, zda vámi vybraná proxy překlad mezi IPv4 a IPv6 umí. Vhodným řešením je například Kernun Net Access.

7.2 Scénář 2: IPv6 uvnitř, IPv4 venku

Pokud byste chtěli experimentovat s IPv6 již dnes a chtěli být zároveň připojeni do Internetu, není problém využít existující IPv4 připojení. Pro propojení sítí je opět možné využít tzv. dual-stack proxy. Tyto proxy umožňují komunikaci na obou protokolech a jak již bylo dříve zmíněno, dokážou mezi protokoly také překládat. Výhodou může být také možnost automatického přechodu na IPv6 hned jak ho bude váš ISP poskytovat.

7.3 Scénář 3: Propojení poboček přes IPv6

Oddělené sítě, jež stále běží na IPv4 bude možné přepojit přes VPN tunel postavený nad IPv6. Takto bude možné udržovat současnou infrastrukturu intranetů v počátcích migrace Internetu.

8 Závěr

Vyčerpání adresního prostoru IPv4 je přede dveřmi, k migraci se ovšem nikomu nechce. Důvodem je nejen málo zkušeností s protokolem IPv6, ale také potřeba poskytování služeb obou protokolů. S tím jsou spojeny mnohé komplikace: podpora obou protokolů v síťových prvcích, serverových a uživatelských aplikacích, rozšíření bezpečnostních politik k IPv6, atd. Pro zjednodušení byly do nového

protokolu zahrnuty techniky, se kterými se původně nepočítalo (např. překlad síťových adres). Stejně tak je ovšem možné použít současné technologie pro překlad paketů mezi těmito sítěmi. Příkladem takové technologie jsou proxy (např. Kernun Net Access), nehledě na navržení k bezpečnostním, nebo jiným účelům. Netransparentní překlad paketů na rozhraní proxy umožní komunikaci mezi sítěmi IPv4 a IPv6, i když jsou navzájem nekompatibilní.

Pokud byste již dnes chtěli zkusit IPv6 ve vlastní sití, máte možnost využít připojení nabízené místními ISP. Tyto možnosti jsou sice omezeny, s velkou pravděpodobností byste však měli najít ve svém okolí poskytovatele disponujícího IPv6 peeringem přidělujícího IPv6 adresy také koncovým zákazníkům. Stejně tak je možné využít možnost připojení serverů v datacentrech k IPv6 síti, které již není ničím výjimečným. Umístění České republiky na první příčce v počtu IPv6 autonomních systémů k IPv4 systémům je proto možné považovat za opodstatněné.

Literatura

[1] APNIC Homepage, URL: http://www.apnic.net, citováno 23. 4. 2010.

[2] ARIN: *Policy Proposal: IPv4 Transfer Policy Proposal*, URL: http://lists.arin.net/pipermail/arin-ppml/2008-February/009976.html, poslední aktualizace 11. 2. 2008, citováno 23. 4. 2010.

[3] ARIN WHOIS Database, URL: http://ws.arin.net/whois/, citováno 23. 4. 2010.

[4] BGPmon: *The Vatican taking the lead in IPv6 rollout?*, URL: http://bgpmon.net/blog/?p=228, poslední aktualizace 26. 10. 2009, citováno 23. 4. 2010.

[5] Carolyn Duffy Marsan: *Will there be an IP address black market?*, URL: http://www.networkworld.com/news/2008/021408-ipv4-iana-conrad.html, poslední aktualizace 18. 2. 2008, citováno 23. 4. 2010.

[6] Derek Morr: *Verizon mandates IPv6 support for next-gen cell phones*, URL: http://www.personal.psu.edu/dvm105/blogs/ipv6/2009/06/verizon-mandates-ipv6-support.html, poslední aktualizace 8. 6. 2009, citováno 23. 4. 2010.

[7] Geoff Huston: *AS65000 BGP Routing Table Analysis Report*, URL: http://bgp.potaroo.net/as2.0/bgp-active.html, poslední aktualizace 23. 4. 2009, citováno 23. 4. 2010.

[8] Geoff Huston: *IPv4 Address Report*,
URL: http://www.potaroo.net/tools/ipv4/index.html, poslední aktualizace 23. 4. 2010, citováno 23. 4. 2010.

[9] Iljitsch van Beijnum: *Can an IPv4 stock market stave off address depletion, IPv6?*, URL: http://arstechnica.com/old/content/2008/02/can-an-ipv4-stock-market-stave-off-address-depletion-ipv6, poslední aktualizace: 18. 2. 2008, citováno 23. 4. 2010.

[10] NIX.CZ: *IPv6 Peering*, URL: http://www.nix.cz/cz/peering_ipv6, citováno 23. 4. 2010.

[11] Pavel Satrapa: *IPv6*, CZ.NIC, z. s. p. o.

[12] Sean Siler: *Mythbusters #2: Stanford has more IP addresses than China*,
URL: http://blogs.technet.com/ipv6/archive/2007/05/14/mythbusters-2-stanford-has-more-ip-addresses-than-china.aspx, poslední aktualizace 14. 5. 2007, citováno 23. 4. 2010.

[13] The Yomiuri Shimbun: *IP address trade may start in '10*,
URL: http://m2m.tmcnet.com/news/2009/12/22/4546069.htm, poslední aktualizace 22. 12. 2009, citováno 23. 4. 2010.

[14] *Windows Vista: Check your IPv6 configuration*,
URL: http://www.ipv6day.org/action.php?n=En.Configuration-WindowsVista, poslední aktualizace 8. 3. 2007, citováno 23. 4. 2010.

Automatic source code transformations for strengthening practical security of smart card applications

Vašek Lorenc, Tobiáš Smolka, Petr Švenda

E-mail: valor@ics.muni.cz, xsmolka@fi.muni.cz, svenda@fi.muni.cz

Abstract

The availability of programmable cryptographic smart cards provides possibility to run application in significantly more secured environment then ordinary personal computer. Smart card platforms like Java Card or .NET allow to implement portable applications that can be run on different smart card hardware. Barriers for a skilled Java developer switching to the Java Card platform are relatively small – working applets can be written quickly. Unfortunately, the resulting overall security of the applet is strongly dependent on the implementation of the smart card operating system, related libraries, as well as physical resistance and information leakage of the underlaying hardware. Same Java Card applet may run securely on one smart card hardware platform, but be vulnerable on an other. Defenses implementable on the source code level for later case might exist, but such a situation is unfavorable for applet developer as multiple versions of applet must be maintained to support a wider range of smart cards (although all providing Java Card platform).

In this paper we describe several practical attacks on modern smart cards, discuss possible defenses and propose a general framework for automatic replacement of vulnerable operations by safe equivalents. A code strengthening constructions can be also automatically inserted. Only one version of the applet is maintained for multiple different smart card hardware and personalization of source code is performed in in automated fashion. Practical implementation and examples of usage are presented and discussed.

1 Introduction

The cryptographic smart cards are currently in ubiquitous usage in many areas of daily life starting from mobile phones SIM modules over banking cards up to

document signing. A common cryptographic smart card consists from several main components. The main processor (8–32 bits) is capable of execution of ordinary code either in form of native assembler code or more directly bytecode for Java Card smart cards. The cryptographic co-processor is designed to significantly speed up execution of time-consuming cryptographic algorithms (e.g., RSA or DES) and to provide additional protection for cryptographic material in use.

Although significantly more secure then ordinary the PC platform, cryptographic smart cards cannot be assumed as a completely secure environment. Several classes of attacks were developed in recent years, targeting everything on smart card platform from insufficient physical security over side channel leakage to logical errors in implementation. Epoxied packaging is etched-out or small holes are drilled so that micro-probes can be inserted and used to read out the memory content. Power consumption or electromagnetic emanations of the chip is measured with high precision and processed key material is revealed. Faults are induced into the memory so computation is corrupted and may result in unwanted behavior of the applet. Incorrect or incomplete implementation of programming interface can be misused to reveal sensitive data.

A defense can be usually developed and implemented against the particular attack. However, originally simple code may become complicated by that and non-trivial knowledge from developer is also required. Moreover, manual implementation of defense can introduce new coding errors. Additionally, different smart card hardware or operating system implementation may require different defenses. Original idea of "Implement once, run everywhere" is then hard to fulfill.

Our contribution here is a design and prototype implementation of system, which allows to parse applet source code and produce transformed version with incorporated protections. Such automatic transformation provides possibility to maintain only single version of the main applet with all defenses (possibly specific for particular smart card hardware) consistently added later. Transformation rules can be provided by independent security laboratories and therefore decreasing requirements on developer when defenses and best practices are implemented.

Our work is motivated by following targets:

- To enable clear code development with minimization of the logical errors while defensive code constructions can be added automatically later.
- To incorporate software-level defenses implemented by others and support transparent code reuse without introduction of unintentional coding errors.
- To provide software-level defense against smart card vulnerabilities when different smart card hardware platform without such vulnerabilities cannot be used.

- To support quick reactions on newly discovered vulnerable operations, when immediate switch to other hardware platform is not possible and applet rewriting may introduce new implementation errors.
- To detect potential problems or unintentional consequences of operations used in source code.

The rest of the paper is organized as follows: Section two gives description of selected problems of smart card security, describes testing setup used by us and discuss relevant work in area. In Section three, general framework for automatic replacement of vulnerable operations is proposed. Our prototype implementation is described in section four, together with four practical examples and limitations of the approach. Section five describes potential areas of the future research and concludes the paper.

2 Selected problems of cryptographic smart cards

Besides the programming language, API and libraries, a Java Card programmer has to deal with many obstacles in order to produce robust, functional and secure applet. In order to illustrate some difficulties and attacks, we are going to describe selected problems of the platform.

We will focus on Java Card platform, but described problems generally hold for other smart card platform as well – namely .NET or MULTOS smart cards. Underlaying hardware is usually the same or very similar (main CPU, EEPROM memories, cryptographic co-processor) and basic principles of software environment as well (applet isolation, memory management, available libraries). Therefore, described attacks are usually relevant also for other platforms.

2.1 Java Card platform security problems

With the complexity of Java Card programming topics, more than just programming language related problems should be taken into account – issues coming from the long and sometimes unclear platform specification, multi applet environment and inconsistencies on the platform implementation level might lead to hidden security risks, investigated more in [3, 12, 2].

To an accidental observer, Java Card platform looks like a full featured Java environment – there are plenty of properties shared by both the platforms. E.g. Java language and bytecode is used in Java Card, although the second one implements only a subset of the whole ("desktop") Java specification, mainly due to the limited resources of the platform. Unfortunately, there are some

substantial differences that can become security benefits or be source of security problems:

```
public class OwnerPIN implements PIN {
  byte[] triesLeft = new byte[1];
  byte[] temps = JCSystem.makeTransientByteArray(1,
      JCSystem.CLEAR_ON_RESET); // temporary array

  boolean check(...) {
    ...
    // compute new value of the counter:
    temps[0] = triesLeft[0] - 1;
    // update the try counter non-atomically:
    Util.arrayCopyNonAtomic(temps, 0, triesLeft, 0, 1);
   ...
  }
}
```

Figure 1: PIN verification function resistant to transaction rollback [3]

No dynamic class loading and threading can reduce the complexity of reasoning about security properties of an applet. Limited support for threading was introduced recently in Java Card 3.0 Connected Edition specification.

Applet firewall mechanism is a key element in controlling and restricting data flow between different applets installed on one smart card. Improper implementation of this system compoment could lead to private data leakage and/or modification.

Atomic operations and transactions may help the programmer to keep the applet data consistent regardless of the card tears.

Despite its useful properties, transactions could arrange hardly predictable results when improperly combined with a code that was not designed with transactions on author's mind. Moreover, even the Java Card specification prior to version 2.0 had a reference code of PIN verification function prone to transaction rollback. If a less careful programmer would use it within a running transaction, it would leave to an attacker an unlimited number of PIN guesses. A better version (Figure 1, explained in [3]) uses non-atomic functions to modify a PIN counter value.

No on-card bytecode verifier is a critical missing component when preventing mistyped code to access memory beyond outside. Although there are some cards with on-card verifier, the verifier itself can be limited and incomplete, making relying on such verification unsafe [3].

Bytecode verification is supposed to prevent attackers from viewing and modification of different memory locations in Java environment, where types and memory accesses are usually strictly controlled. Problem with Java Card platform lies in very limited resources, so that on-card verification is usually unfeasible and thus *off-card verification* has to take place instead. Despite the fact that the off-card verification might prevent some problems, it is clearly inefficient against attacks focused on malicious bytecode modifications after the verification process and faults induced during the applet execution.

Applet/data persistence might be a source of an unexpected problems when programmer does not use a proper variable initialization – variable initialized only in one applet run might be misused when dealing with card tears and consequent runs.

Absence of garbage collector could make memory (de)allocations more error-prone, especially when combined with transaction mechanism. Even a code perfectly correct from the verifier perspective may eventually lead to an invalid memory access, causing either negative impact for the applet environment or allowing the attacker to read out all of the available memory of the smart card [4].

Many Java Card Runtime Environment (JCRE) issues arising from small but subtle differences between Java Card and Java environments might illustrate the complexity of programming Java Card applets. This is the situation when high-level approach might hide important differences causing the security problems for resulting Java Card applet.

2.2 Multi Applet Programming

Developing of a single applet for Java Card might force a programmer to overcome some difficulties, either technical or security ones. However, design and development of an applet that has to share the same card and some private data with other applets might be even more challenging.

Difficulties arise from the fact that other applets developed typically by third party will occupy the same card and could share some data in some instances. Malicious applets installed on the card might be used to build a power-analysis profile of the card, overcome Java Card runtime checks and/or read out all available memory in order to retrieve private data.

Besides these, difficulties with *Shareable Interface Object* became another source of possible security issues.

On Java Card platform, every applet and system runtime has its *security context* that has to be checked by the runtime environment when objects and applets try to communicate and share data. Applet firewall uses object-oriented approach to control behaviour and data visibility between different applets.

In [12] authors demonstrated that some implementations of Java Card firewall are vulnerable to inappropriate manipulation with *Shareable Interface Object*, revealing sensitive information about system objects with *isinstance()* or *equals()* functions and in the worst case leaving a chance to manipulate system AID, making some of the protection mechanisms in Java Card inefficient.

2.3 Power analysis

A significant thread to smart card hardware comes from the family of side channel attacks. The attention to the power analysis was first drawn by Kocher et al. [5]. They presented a powerful power analysis attack, differential power analysis, which led to the extraction of the DES keys used by the smart card. Since then, many researchers has focused on the power analysis attacks and many new techniques have been developed. Lot of effort has been spent on the power analysis of ciphers, such as DES and AES. An exhaustive overview of the power analysis techniques and countermeasures can be found in [7]. Although original attack is more than ten years old, defense mechanisms implemented by smart card manufacturers are still far from perfect.

The power analysis enables an attacker to obtain potentially sensitive information on the operations executed by the smart card as well as on the data processed. It is based on measurements of the current drawn by the card from the reader. The measurements can be done with low cost devices using a small resistor (e.g. 30 Ω) connected in series to the smart card ground line and a digital oscilloscope. The oscilloscope is attached across the resistor and measures the fluctuation of current. The fluctuation forms a power trace (see Figure 4 for example), which displays the current consumption in time. The typical smart card power consumption varies in the order of single milliampere to tens of milliamperes.

The main focus of existing countermeasures is on the protection of the cryptographic material during execution of cryptographic algorithm – e.g., DES, AES or RSA computation. Such algorithms are executed in special cryptographic co-processors with increased protection. Protection is partially based on the obscurity and the exact protection mechanisms are usually not published by smart card manufacturer. Obtaining cryptographic material via power analysis of cryptographic algorithm from cryptographic co-processor is difficult for an attacker today. However, other parts of the code execution on smart card re-

Figure 2: SCSAT04 power measurement device for cryptographic smart card analysis

ceived much smaller amount of attention. Not only cryptographic co-processor, but also execution of instructions in the main processor should be protected, as potentially sensitive data might be processed also on this level. In the seminal paper of this field [9] the authors exploited the power analysis techniques to reconstruct the bytecode of the Java Card applet running on the smart card.

Our power analysis platform, SCSAT04 (see Figure 2), was developed in collaboration with VUT Brno. It integrates multiple side channel tools on a single board. The core is an embedded linux running on the etrax CPU, which provides communication with PC via ethernet network and with a smart card via an integrated smart card reader. The board also contains 200MHz oscilloscope and 12bit A/D converter for measuring the smart card power consumption. Beside power analysis, the SCSAT04 is capable of fault induction via peaks on smart card power supply, data and clock bus. At the current setup, we are able to sample the card power consumption with the digitalization frequency up to 100 Mhz and with the resolution of 12 bits per sample. The SCSAT04 board is equipped with 48 MB of fast RAM, thus it can store up to 24 millions samples. The sampling is triggered on a specific pattern of bytes on the I/O wire. The SCSAT04 board was designed to introduce as little noise as possible into the measurement process.

2.4 Constructions vulnerable to power analysis

The sensitive data processed by the smart card might be revealed by power analysis in three different ways. First, the data can be revealed by statistical

means directly while they are processed by a vulnerable instruction. Second, if different instructions are executed depending on the sensitive data, an attacker can analyze the program flow and thus reveal the data. And third, if single instruction execution differs depending on the data, the attacker again is able reveal the data.

The first way is the most powerful but also the most difficult one. The vulnerability stems from the fact, that the amount of power consumed by the card during an execution of an instruction depends on the data (usually on its Hamming weight) manipulated by the instruction. This vulnerability has been well studied and number of attacks has been proposed including Kocher's differential power analysis. However this kind of attack is extremely difficult in real conditions with commercial cryptographic smart cards. With our current setup, we have not been able to successfully reveal any information on processed data this way.

The other two ways are more usable. We have shown, that it is possible to identify particular bytecode instructions and reconstruct the bytecode executed [6]. So if the program flow is controlled by the sensitive data, the attacker might be able to reveal them by bytecode reconstruction. Consider an *if-then-else* construction. The attacker is not able to obtain values involved in the condition directly from power trace, however if she can tell the difference between execution of *then* and *else* branch (because sequence of instructions is different), she reveals the result of the condition anyway. Thus if the condition is based lets say on the key bit, she can easily reveal its value. Simple countermeasure is to execute similar bytecode in the both branches. This counters also the time analysis (information leakage about processed data based on time needed to finish the operation), because the time of execution of both branches is same and constant.

Similar problem arises in the cases of *for* and *while* cycles. The attacker is able to determine number of loops executed. But also this can be easily countered. Cycles depending on sensitive values should always perform constant number of similarly looking loops, even though some of them are not necessary. This approach again effectively counters the time analysis as the time of execution is constant. Note that using random number of loops could partially solve the problem, but the attacker might be able to estimate the original value by averaging over multiple runs of the application.

Example of suchlike vulnerability can be found in comparison of arrays. Typical comparison is done sequentially and finishes just after a different element is found. Suppose the application is comparing arrays with PIN digits in plaintext. Then if the comparison finishes just after the second digit was compared, attacker knows that first digit was correct but the second not. Therefore she can guess the digits separately one at a time and thus rapidly increase her chances. Secure comparison should be constant in time and ideally non-deterministic and

not sequential. It means that elements are compared in random order, which is changed on each run of the applet. This can effectively counter also statistical attacks on the processed data.

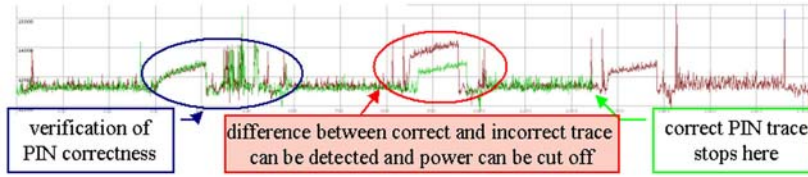


Figure 3: An example of the power trace of PIN verification procedure. Difference between correct and incorrect PIN is clearly visible

Sensitive information can be also leaked during an evaluation of the boolean expressions. If the lazy evaluation strategy is implemented then an attacker can reveal values of variables used in the expression. Consider a boolean expression $A\&B$. In the case of very quick evaluation, attacker knows that A was *false*, otherwise A was *true*. This expression can be easily fixed by transformation into its equivalent $!(!A\|\|!B)$. Note that non-optimizing compiler should be used to prevent the compilation into vulnerable expression different from non-vulnerable one written in the source code. Result of the transformation should be also verified in the compiled bytecode to confirm, that compiler didn't changed intended representation of the bytecode.

The above examples demonstrate how sensitive values can be extracted just by the analysis of the bytecode executed. We have also presented the defense mechanisms, which effectively counters them. However we have encountered another weakness, which enables the third power analysis technique that might lead to sensitive data leakage. We have found out [6], that appearance of instructions for conditional jump, like *ifeq*[1] or *ifne*[2], differ depending whether the jump was taken or not. The reason probably comes from the way how Instruction Pointer (IP, pointing to the instruction to be executed) is manipulated. If no jump in code is performed, IP is incremented by one via processor native *inc* micro-instruction[3]. If the conditional jump is performed, IP is incremented by the value usually bigger than one (offset to jump target) and therefore *add* micro-instruction is used. As *add* micro-instruction is more complicated than *inc*, its duration is longer and results in significantly different power trace for the parent bytecode instruction. Thus even if the programmer follows secure programming patterns and both branches of the *if-then-else* construction exe-

[1]Instruction jump if equal.
[2]Instruction jump if not equal.
[3]One bytecode instruction comprises from several micro-instructions.

cute similar bytecode, attacker might be still able to reveal the result of the condition.

Fortunately, this *if-then-else* construction can be avoided and replaced by semantically equivalent *switch* construction, which uses *stableswitch* bytecode instruction. The *stableswitch* instruction still leaks some information when first branch from all *case* statements is taken (on some hardware platforms), but not for the other branches. The reason is probably the same as for the *if* instruction – the first branch increments IP only by one (next instruction after *switch*) where other branches require bigger change of IP via *add* micro-instruction. The secure replacement of the *if-then-else* by the *switch* construction is not completely straightforward and is described in the greater details in the section 4.

2.5 Bytecode reverse engineering

Power trace might leak some information on the operations performed by the smart card. In particular, execution of single instructions of the processor can be detected and sometimes even type of the instructions can be identified accurately. Accordingly, we might be able to partially reconstruct (reverse-engineer) source code of an application running on a smart card and obtain potentially sensitive information about processed data. We have tested our approach on fourteen different types of commercially available Java Card smart cards from the leading smart card manufacturers. It turned out, that only a third of them was resistant to this kind of power analysis technique with our measurement setup.

Some cards were easier to attack than others. These cards execute a special pre-instruction before each bytecode instruction. This pre-instruction is clearly visible in the power traces, see the parts marked as JF in the Figure 4). We have called it a "separator", in figures marked as JF, because it effectively separates subsequent instructions. We consider the presence of separators as a vulnerability as it makes the whole process of reverse engineering much easier. The separators are usually easy to find and once you have the instructions isolated, you can directly compare them with the templates from the database.

The problem of the vulnerable operations described in the previous section can be solved by switching to a different non-vulnerable smart card hardware. Such solution might be appropriate when only small amount of the smart cards was purchased and other suitable hardware platform exists. Yet platform switching might not be an option when particular platform is already in wide use with the significant resources invested or when simply no other suitable platform with required functionality (memory, speed, supported cryptographic algorithms) is available.

During the power analysis of bytecode instructions we observed that not every instruction leaks an information about its operands. Some vulnerable instructions might have semantically equivalent replacement instructions that is

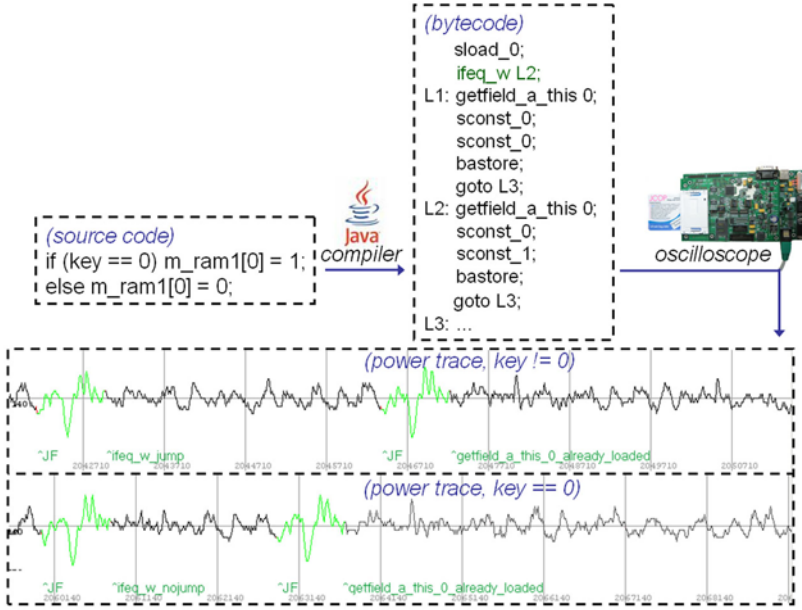


Figure 4: Example of process of reverse engineering of Java Card applet. Bytecode instructions are visible in power trace and allow to distinguish whether then or else branch was taken based only on *if_eq* operation

not vulnerable to attacks presented above. If such replacement instruction exists, source code of Java Card applet can be modified to use different programmatic structure that will compile into sequence of non-vulnerable instructions instead of vulnerable ones.

3 Automatic replacement framework

General patterns for secure programming in the presence of side channel attacks like [11] were developed and are usually followed by the developers to some extend. It introduces the best practices for the developers of security critical devices with side channel leakages or fault induction vulnerabilities. However, consistent incorporation of best practices remains and issue. Proposed defensive constructions often decrease code clarity, make it less readable and increase probability of introduction of unintentional error. Demonstration of this fact is an example of the pathway from the naive to the robust implementation of the PIN verification procedure described in [10], increasing code length more then five-

fold. Some protections are relevant only for some smart card hardware, posing only unnecessary overhead for other. Some constructions cannot be used at all at particular smart card as hardware lacks the support for required operations.

In contrast, our solution tries to remove the burden of the secure programming patterns from the programmers to some extend.

Several practical security-related requirements should be also fulfilled:

1. Java Card application must be available to functionality and security audit after vulnerable instructions were replaced – as code audit purely on bytecode level is significantly more difficult and lot of additional information (e.g., programmers comments) are lost at this moment, replacement of vulnerable instructions should be done on the source code level.

2. Transformed code should clearly show the new version of the code, the old code (that was replaced) as well as description of the replacement rule that was used for it.

3. Code replacement should be easy to adapt to different smart card hardware with different set of vulnerable instructions. The goal is to write applet source code only once in direct and clean way without any restrictions on potential vulnerability of used instructions. The source code of actually deployed applet is then generated automatically based on proposed replacement framework, personalized for particular smart card hardware, taking into account vulnerable instructions existing on this hardware platform.

3.1 Vulnerable instructions replacement framework

Equipped with previously described problems and requirements for defenses, we can now describe details of proposed *automatic replacement framework*. The whole process requires both manual human analysis and software automated steps. However, once manual steps are performed (e.g., by independent security laboratory), main part of the remaining work can be done in anautomated fashion. The overview of whole process is shown on Figure 5.

The whole proposed process consists from the following steps:

1. Identification of vulnerable constructions – identification of problems with specific smart card (with testing tools like SCSAT04 or software testing suite [3]) or using well known generic vulnerabilities (e.g., defense against fault induction attack). This step is usually performed once for given smart card hardware and results in list of vulnerable instructions and constructions;

2. Identification of generally vulnerable constructions taken e.g. from the best practices;

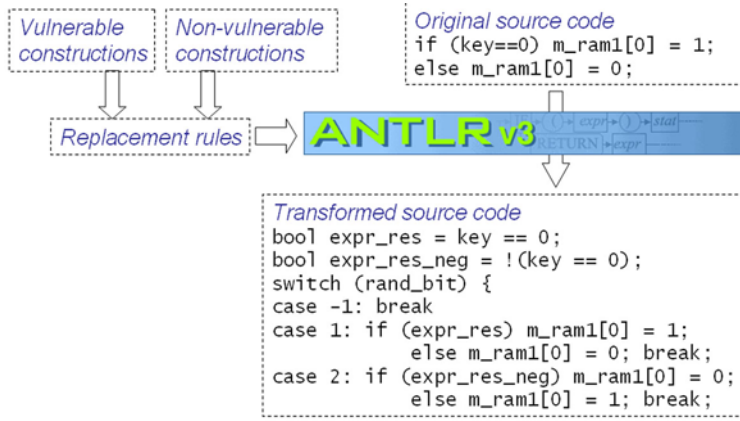


Figure 5: Automatic replacement framework process. Templates of vulnerable constructions and their secure secure replacement are combined in the replacement rules. ANTLR parser is used to automatically transform source code of the applet

3. Identification of the non-vulnerable replacement constructions – results in list of non-vulnerable instructions and constructions that can be used to replace vulnerable ones;
4. Creation of the replacement rules – based on the list of vulnerable and non-vulnerable instruction, semantically equivalent replacement rules are created. See Section 4.1 for example;
5. Processing of the application source code:
   (a) Source code is parsed with syntactic analyzer like ANTLR and syntactic tree is created;
   (b) Code statements to be replaced (vulnerable) are identified, based on the list of vulnerable constructions;
   (c) Replacement rule is found for the vulnerable statement;
   (d) Automatic replacement of vulnerable statement is done on the level of parsed syntactic tree – replaced code is commented out;
   (e) Modified source code is generated from modified syntactic tree.
6. Modified parts of the code can be used as an input for manual audit – transformed code is still in human readable form and contains commented parts of replaced code with identification of used replacement rule;
7. Compilation and analysis of resulting power trace to verify robustness of replaced code.

Whole programmatic constructions can be targeted. The replacement rules can be created to target not only the power analysis leakage (e.g., branch taken in *if_eq*), but also fault induction resiliency (e.g., introduction of shadow variable containing copy of inverted variable value), additional strengthening when correctness of exact implementation of smart card hardware is not known (e.g., additional counter of allowed attempts to PIN verification) or introduction of best practices techniques (e.g., robust clearance of key material before deletion). Examples can be found in Section 4.

Special attention should be payed to behavior of compilers as vulnerable sequence of instructions can be still produced due to compiler optimization even when source code was carefully written. Manual analysis of resulting bytecode is vital addition to source code analysis with respect to existence of vulnerable constructions.

4 Prototype implementation

The prototype implementation of described framework was performed. We used freely available yet very powerful parser ANTLR (ANother Tool for Language Recognition) [1] as a tool for parsing Java Card source code (grammar for Java 1.5 was used), manipulating vulnerable statements and producing output source code. ANTLR grammar is used for description of vulnerable statements as well as its non-vulnerable replacement.

4.1 Example: Vulnerable bytecode replacement

As was discussed in Section 2.4, instructions of conditional jump (*if* family – *if_eg, if_neg, ...*) may leak information about the branch selected for program continuation. Such leakage is clearly unwanted as it reveals result of condition evaluation even when both branches are identical on the instruction level (prepared in such way as a defense against timing attack), thus leaking information about variable used in expression and rendering such defense ineffective. Partially non-vulnerable *switch* instruction was identified for the same platform that is not leaking information about the branch taken (see Section 2.4 for discussion).

Equipped with such a knowledge, we can design replacement rule that will automatically replace occurrences of vulnerable *if-then-else* statement to semantically equivalent and secure statement based on *switch* instruction and randomization. The transformation is not completely straightforward. At first, bogus code of branch that will never be used must be added at position just after *switch* instruction in compiled bytecode. This bogus branch will occupy vulnerable position accessible with *inc* micro-instruction (see Section 2.4 for rationale). Additionally, *switch* operates over integer operand where *if* operates

over boolean expression. Simple trick to typecast operand using conditional statement *expr*?1 : 0 (where *expr* is original boolean expression from *if-then-else* statement) cannot be used as it keeps vulnerability in the conditional expression – an attacker can observe the evaluation of the statement *expr*?1 : 0.

Therefore, randomization needs to must be introduced. Result of expression is evaluated in advance (expr_res variable) as well as its negation (*expr_res_neg*). The actual *switch* branch taken is selected in runtime based on the random variable with two possible values, 0 and 1. If random variable is equal to 0, *switch* branch (*case*) containing original ordering of *if-then-else* branches and expr_res is taken. Otherwise branch with inverted result of boolean expression (*expr_res_neg*) and swapped *then* and *else* branch is taken. This replacement rule is more formally described on Figure 6.

```
// grammar snippet describing IF statement, vulnerable on certain platforms
^(if parenthesizedExpression ifTrue=statement ifFalse=statement?) - >
ifTransformation(
    expr={$parenthesizedExpression.text},
    ifTrue={$ifTrue.text},
    ifFalse={$ifFalse.text}
)

// grammar snipped describing replacement SWITCH construction
ifTransformation(expr,ifTrue,ifFalse,random_bit) ::= <<
boolean exp_res = <exp>;        // result of original expression
boolean exp_res_neq = ! <exp>; // inverted result of expression
switch (random_bit){
        case -1:
                // never executed
                break;
        case 1:
                if (exp_res) ¡ifTrue¿ else ¡ifFalse¿
                break;
        case 0:
                if (exp_res_neq) ¡ifFalse¿ else ¡ifTrue¿
                break;
        default:
                throw new Exception();
}
>>
```

Figure 6: Example replacement rule for (vulnerable) *if-then-else* statement by non-vulnerable construction using *switch* statement with randomized execution of original and inverted command. Slightly simplified for the clarity reasons

The described replacement will results into non-vulnerable construction on given platform according to the following analysis. An attack should not be able obtain knowledge about expression operand as he cannot:

- Distinguish which *switch* branch (*case*) was taken – the value of random variable is unknown to an attacker and *switch* instruction itself is not leaking information about branch taken. Therefore attacker has no knowledge whether $if(exp\_res) < ifTrue > else < ifFalse >$ or $if(exp\_res\_neq) < ifFalse > else < ifTrue >$ statement will be executed even when source code is known to him.
- Distinguish which *if-then-else* branch was taken – an attacker can still observe execution of *if* instruction and decide whether first (*then*) branch or second (*else*) branch was taken. But as he cannot distinguish whether jump decision was based on *exp_res* or *exp_res_neg* nor he knows *switch* branch, both possibilities are equally probable for him.
- Infer information about operand evaluated in *if-then-else* expression – as branch taken cannot be distinguished, an attacker cannot distinguished whether *ifTrue* or *ifFalse* statement was executed[4]. Ultimately, an attacker cannot infer the information about the operand evaluated in *expr* expression as both results (*expr* is true/false) are equally possible.

Although such a replacement can done manually in principle, resulting code is significantly less readable than original *if-the-else* instruction and is hot candidate for the coding mistakes. Usage of automatic replacement framework will help here to develop straightforward source code and add complicated non-vulnerable construction later.

4.2 Example: Protection against memory fault induction

Another practical example, usable as a defense against wide range of fault induction attacks, is introduction of the shadow variables or shadow arrays to detect faults induced by an attacker in memory. Shadow variable usually contains inverse value of original variable and consistency is checked (resp. updated) every time and original variable is used (resp. changed). Such protection is example of more general protection – is independent from particular hardware and should be widely used. Yet, implementation of such defense obscures significantly source code as variable consistency against shadow counterpart must be ensured.

Replacement rule for this protection was implemented within proposed framework. Developer implements source code without mentioned defense. Source code is then parsed, variables in code are identified and shadow counterparts are automatically created. Special “getter” and “setter” functions are created for

[4] Assuming that both statements consists from same sequence of instructions.

every used primitive data type and inserted into original code. “Getter” takes care for verification of variable consistency against shadow variable, “setter” provide variable update functionality. The example of such transformation is shown on Figure 7.

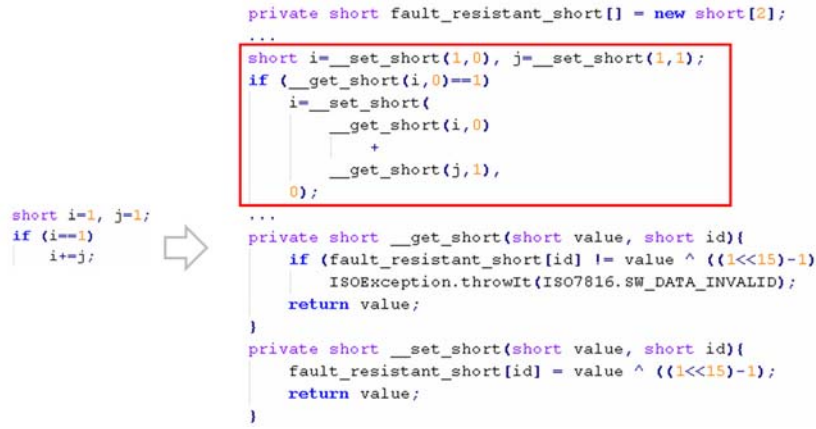


Figure 7: Example of source code transformation adding robust protection against fault induction by creation of shadow variables with an inverse value. Array *fault_resistant_short* contains shadow variables for *short* type. Variable in original code is replaced by “setter” or “getter” call performing lookup into *fault_resistant_short* array and checking consistency

4.3 Example: Robust state checking and transition with visual modeling

Typical Java Card applet is typically operating in several logical states with different levels of authorization. Some information like applet identification number might be accessible to anybody. Possibility to use to use on-card signature key is available only after user was authenticated by PIN. Blocked user PIN can be unblocked only when administrator authenticated with his own secret key. Usual solution is to check fulfillment of security preconditions before operation is executed (e.g., OwnerPIN::isVerified() before signature key is used) or special variable holding current logical state state (e.g., “user authenticated” or “admin authenticated”) is introduced and checked. With the exception of very simple applets, maintaining a correct checks of logical states, especially when sudden card removal from reader may occur and functions are later added to source code of existing applet.

Two main categories of problems with state may occur:

1. Bug in code – improperly verified transition between states or unintended function call without fulfillment of all security preconditions. Function call may be accessible to outside attacker by sequence of operations unintended by developer.

2. Fault attack – Applet security state is changed despite the correct code implementation. An attacker may be able to make (usually random and untargeted) change in smart card memory or instruction sequence. If variable holding applet state is corrupted, an attacker may be able to execute sensitive operation without prior authentication. Result of comparison against expected state may be corrupted as well during execution of comparison instruction.

Proposed solution robustly enforcing state checking and transition is based on following steps:

Model and visualization of allowed state transitions and function calls – developer can easily define, change and inspect visually state transition model as well as functions callable in each security state. Model is automatically and consistently transformed and integrated into existing source code. For model description and visualization, we used GraphViz grammar[5] as it can be conveniently created, visualized and inspected. ANTLR grammar was created, allowing to parse GraphViz grammar and transform it into Java Card source code.

State transition enforcement – state transition is always moderated over specialized function and new state can be only one from well defined set of consequent states. Developer only adds SetStateTransition(newState) function call every time an applet likes to change its security state (e.g., after user PIN is verified). Transition is checked against set of allowed transitions and permitted only when transition between *currentState* (hold in special variable) to *newState* exists in the model.

Function independently verifies if can be called – function itself tests, whether can be called in present state and context before continues to execute code in its body. Function VerifyAllowedFunction() is added at beginning of every function. If current function is not allowed in the model to be called in the present security state, applet execution is aborted with appropriate response (e.g., logging or even card blocking).

[5] http://www.graphviz.org/


```
private void SetStateTransition(short newState) throws Exception {
  // CHECK IF TRANSITION IS ALLOWED
  switch (m_currentState) {
    case STATE_UPLOADED: {
        if (newState == STATE_INSTALLED) {m_currentState = STATE_INSTALLED; break;}
        throw new Exception();
    }
    case STATE_INSTALLED: {
        if (newState == STATE_SELECTED) {m_currentState = STATE_SELECTED; break;}
        if (newState == STATE_BLOCKED) {m_currentState = STATE_BLOCKED; break;}
        throw new Exception();
    }

    case STATE_SELECTED: {
        if (newState == STATE_SELECTED) {m_currentState = STATE_SELECTED; break;}
        if (newState == STATE_USER_AUTH) {m_currentState = STATE_USER_AUTH; break;}
        if (newState == STATE_ADMIN_AUTH) {m_currentState = STATE_ADMIN_AUTH; break;}
        if (newState == STATE_BLOCKED) {m_currentState = STATE_BLOCKED; break;}
        if (newState == STATE_INSTALLED) {m_currentState = STATE_INSTALLED; break;}
        throw new Exception();
    }
```

Figure 8: Example of applet transition model visualization and resulting source code for SetStateTransition() function. Similar code is generated also for VerifyAllowedFunction()

4.4 Example: Java Card Firewall Implementation Issues

Designing an applet for smart card where other applets of third-party providers will be installed may put higher demands on proper Java Card firewall implementation.

Altough it might be unfeasible to overcome all non-complient behavior of the firewall of a specific Java Card mentioned earlier in this paper, it still can be useful to know what kind of inconsistencies and problems might appear on the given smart card.

In order to simplify this process, the Java Card Firewall Tester [8] made by Wojcich Mostowski project has been released to public use, consisting of set of applets and host application. Altough some cards cannot be tested due to their limitations (either memory limitations or too restrictive applet verification), the result of this tester is still very useful in the rest cases.

Even if the firewall tester does not detect a real security problem, it still might reveal too restrictive (or too permissive) behavior of the JCRE that could help the author of the applet to change the design in order to make it run or

```
Non-external ciphers should not be usable externally.
INS: 0x13 Test#: 13
```

Figure 9: JC Firewall Tester reports non-compliant handling of ciphers declared as non-external

that some specific Java Card properties are not checked properly by the runtime (figure 9).

Another notification for a programmer could be helpful in case of inappropriate use of transaction and non-atomic features of Java Card. In order to illustrate possible problems with (non-)atomic operations, see a code snippet in figure 10. There are obviously two different results for almost identical set of operations, depending on the order of the operations within the transaction – a variable is backed-up just before it is updated conditionally for the first time. Although this is logical behavior, this fact might prepare hard times for an applet designer.

```
a[0] = 0;                          a[0] = 0;
beginTransaction();                beginTransaction();
  a[0] = 1;                          arrayFillNonAtomic(
  arrayFillNonAtomic(                  a,0,1,2); // a[0] = 2;
    a,0,1,2); // a[0] = 2;           a[0] = 1;
abortTransaction();                abortTransaction();
// a[0] == 0;                      // a[0] == 2;
```

Figure 10: Atomic and non-atomic update of a variable with different results [3]

Such a situation cannot be automatically solved by the transformation tool. As a possible result of applet rewrite, even the author seems to be unsure whether the variable should be a subject of transaction rollback or not. However, programmers can only be informed that both conditional and non-conditional update of a persistent variable is being requested inside a transaction and that this may lead to unpredictable results and variable states and that more developer's attention is needed.

These symptoms can be detected by our transformation tool, since the combination of started transaction, a local variable updated with assignment statement and functions `arrayCopyNonAtomic()` or `arrayFillNonAtomic()` called on the same variable can be analyzed on the source level.

4.5 Limitation of approach

There are several practical limitation to the described approach that may affect applicability to some extend, depending on the area where automatic code transformation or replacement is used.

At first, non-vulnerable replacement construction needs to be found. This requires careful inspection of power trace of candidate replacement constructions for possible vulnerabilities. E.g., *switch* instruction was used in one of our examples, but if first branch is utilized, replacement construction will be still vulnerable. Therefore, no proof of vulnerability resistance of resulting bytecode is provided (proofs can be provided in theory, if accurate model of power leakage for particular device is known – but that is frequently not the case).

Readability of the transformed code is usually impacted and possibility of introduction of unintentional bugs to code by transformation is opened. Still, it is usually better option to have simpler code written by the developer itself with complex replacement constructions added later. But transformation needs to be tested and audited carefully.

Depending on properties of replacement construction, computation and memory overhead might be introduced. E.g., described *if-then-else* replacement requires more then three-times more instructions to begin execution of the proper branch. If the transformation requires significant amount of duplicated variables, restricted memory available to the applet might be exhausted. The introduction of shadow variables as a protection against fault induction doubles the required memory and requires function call every time a variable is accessed or modified. Nevertheless, current smart cards usually provide enought power and memory to facilitate additional overhead for a typical applet.

Device-dependent transformations (e.g., vulnerable *if-then-else* instruction transformed by *switch*) need to be kept up to date for particular smart card hardware (if different processor is used, vulnerability assessment needs to be redone). More general transformation (e.g., additional shadow variable) requires less maintenance as are independent from particular hardware.

And finally, replacement construction simply not exists sometimes or is too complicated to create generally applicable replacement rule for ANTLR.

5 Conclusions and future work

This paper described practical vulnerabilities of current smart cards against both logical and physical errors. General replacement framework is described as a way how to automatically, consistently and in error-free way introduce protections on source code level against them. The framework takes the source code of Java Card applet and transform it automatically into the more secure version.

Applicability of the concept is broad – replacement of vulnerable constructions by non-vulnerable ones based on predefined set of replacement rules can be done, as well as consistent insertion of strengthening code (e.g., fault induction protections). Prototype tool was implemented based on ANTLR parser with several replacement rules to verify practicality of the described concept.

The main idea is that source code is developed only once and in clean way, so number of the programming errors is minimized and a focus on logical correctness can be made. Automated framework then add more complex security constructions and personalize code to particular smart card hardware, taking into account its vulnerabilities. These manipulations can be used to add protection against fault induction attacks (e.g., shadow variables), introduce techniques from best practices (e.g., additional check for PIN verification retry counter) or add techniques hardening power analysis (e.g., introduction non-determinism into code execution). Replacement is done in a way that still supports manual code audit of the final source code.

So far, we focused on smart cards with Java Card platform. However, the proposed transformation framework with ANTLR is generally applicable to other platforms like MULTOS or .NET as well as inspected vulnerabilities steams mainly from generally valid attack threads rather then from particular programming platform.

References

[1] Another tool for language recognition (antlr). http://www.antlr.org, accessed 27. 4. 2010.

[2] Poll, E., Hubbers, E. *Transactions and non-atomic api methods in java card: specification ambiguity and strange implementation behaviours.* Available at http://www.cs.ru.nl/~erikpoll/papers/acna_new.pdf, accessed 27. 4. 2010.

[3] Poll, E., Hubbers, E., Mostowski, W. Tearing java cards. In *In Proceedings, e-Smart 2006, Sophia-Antipolis*, 2006.

[4] Hogenboom, J., Mostowski, W. Full memory attack on a Java Card. In *4th Benelux Workshop on Information and System Security, Proceedings*, Louvain-la-Neuve, Belgium, November 2009.
Available at http://www.dice.ucl.ac.be/crypto/wissec2009/static/13.pdf, accessed 27. 4. 2010.

[5] Kocher, P., Jaffe, J., Jun, B. Differential power analysis. In *Advances in Cryptology – Crypto 99, LNCS 1666*, 1999.

[6] Kůr, J., Smolka, T., Švenda, P. Improving resiliency of javacard code against power analysis. In *Proceedings of SantaCrypt '09, Prague*, 2009.

[7] Mangard, S., Oswald, E., Popp, T. *Power analysis attacks.* 2007. ISBN 978-0-387-30857-9.

[8] Mostowski, W. *Java card firewall tester (jcft).* Available at http://www.cs.ru.nl/~woj/firewalltester/, accessed 27. 4. 2010.

[9] Vermoen, D., Witteman, M., Gaydadjiev, G. N. Reverse engineering java card applets using power analysis. In *WISTP 2007, LNCS 4462*, 2007, p. 138–149.

[10] Vétillard, E. *Jc101-12c: Defending against attacks.* 2008. http://javacard.vetilles.com/2008/05/15/jc101-12c-defending-against-attacks/, accessed 27. 4. 2010.

[11] Witteman, M., Oostdijk, M. Secure application programming in the presence of side channel attacks. In *RSA Conference 2008*, 2008.

[12] Poll, E., Mostowski, W. *Testing the java card applet firewall.* Technical report, Radboud University Nijmegen, 2007.

Architektura Windows 7 – od jádra systému po bezpečnostní prvky

Ondřej Výšek

E-mail: ondrej.vysek@microsoft.com

S příchodem operačního systému Windows 7 přišla i celá řada změn a technologií. Pravděpodobně nejzákladnější změnou, která odráží architekturu celého operačního systému je mikro jádro – MinWin. Windows 7 jako první operační systém z dílny Microsoftu odděluje jádro operačního systému od operačního systému jako takového, obdobně jako tomu je u systémů postavených na platformě Unix. Toto minimalistické jádro operačního systému obsahuje opravdu nejzákladnější funkční prvky – cca 150 knihoven, ovladače pro řadiče disků a síťovou vrstvu. Oddělení jádra OS od zbylé části Windows 7 přináší 2 důležité efekty – zvýšení stability celého OS, kde je možné restartovat ovladač při jeho případném pádu a také zvýšení zabezpečení jak operačního systému, tak i dat, která jsou uložena a zpracovávána na počítači.

Další změnou, která musela být provedena s ohledem na mikrojádro je tzv. DLL-refactoring, kde aplikace nemohou volat přímo jádro operačního systému, ale volají knihovnu s původním názvem a tato knihovna předává volání dále do jádra operačního systému. S tímto také souvisí „virtualizace“ některých knihoven. V takovém případě se vývojář nemusí starat, kde je fyzicky implementované konkrétní API, namísto toho volá obecnou systémovou knihovnu, která následně předává volání konkrétní knihovně. Mapování mezi fyzickou knihovnou a „virtuální“ knihovnou je provedeno v souboru apisetchema.dll.

Další velmi významnou změnou je zavedení technologie Time coalescing, která umožňuje sdružovat časovače, které jsou vyžadovány od fyzického hardware. V případě, kdy je spouštěna aplikace, která vyžaduje časovač, operační systém pozdrží IRQ a pokud v systému – hardware již běží časovač o stejné délce, pak jsou tyto časovače vykonávány najednou. Time coalescing je pak nejvíce zřetelný u aplikací, které jsou náročné na výkon, tedy například virtualizace, zpracování multimediálních dat atd.

Dalšími technologiemi, které byly součástí předchozí verze Windows, jsou například User Account Control – UAC. Při výchozím nastavení, i pokud je uživatel administrátorem, uživatel běží v kontextu běžného uživatele, nikoliv

administrátora. Ve chvíli, kdy takový uživatel vyžaduje administrátorský přístup k operačnímu systému, jeho oprávnění jsou povýšena na administrátorská (pokud je uživatel běžným uživatelem, pak samozřejmě k povýšení nedojde). V rámci Windows 7 přibyla další 2 nastavení UAC, a to medium a low. Obě nastavení jsou si velice podobná s tím, že uživatel není dotazován při povýšení oprávnění u programů, které jsou součástí Windows a jsou podepsány certifikátem Windows. Rozdíl mezi těmito nastaveními je pouze v tom, že při nastavení medium dojde k ztmavení obrazovky a dialog s dotazem na povýšení oprávnění není možné nijak ovlivnit. V případě nastavení low je pak dialog pouhým dialogovým oknem, kterého si uživatel nemusí všimnout, ale v případě škodlivého software je možné jej ovlivnit emulací myši či podobnými technikami.

Pokud se uživatel rozhodne vypnout UAC, pak je také automaticky vypnuta i virtualizace souborového systému a registry, která zajišťuje ochranu systémový částí přesměrováním zápisů a čtení běžných programů do uživatelského profilu. Také ale dojde k vypnutí Integrity levels, které jsou využívány i pro chod například Internet Exploreru v tzv. Protected módu, kde IE má právo zápisu pouze do adresářů s cookies a dočasných souborů. Na jakoukoliv změnu mimo tyto oblasti je uživatel dotazová, zdali s nimi souhlasí. Integrity levels jsou rozděleny celkem do 4 úrovní, kde nejvyšší je úroveň systémová a nejnižší je určena pro provoz „nedůvěryhodných aplikací“. Procesy mohou mezi sebou komunikovat pouze z vyšší do nižší úrovně, opačně to není umožněno, resp. Je uživatel upozorněn na nutnost takové akce.

Další technologií, která přispívá ke zvýšení zabezpečení operačního systému je Address Space Layout Randomization (ALSR). Jedná se technologii, která náhodně umisťuje informace spouštěných procesů v RAM, což zabraňuje útočníkovi odhadnout adresu paměti, kde se vyskytují uživatelská data. Technologie ALSR je implementována například i Linux kernel 2.6.12 a vyšší a Mac OS X v10.5 – v obou případech pouze pro vybrané knihovny.

I přes všechny tyto změny v operačním systému se Microsoft snaží zachovat operační systém kompatibilní již pro existující aplikace. Pokud není možné zajistit kompatibilitu aplikací pomocí spolupráce s UAC či prostředky operačního systému, přichází na řadu tzv. shims (compatibility workaround). Jedná se o malé knihovny, které jsou přidány do operačního systému jeho výrobcem, výrobcem software nebo i IT administrátorem, kde při volání specifických API je volání pozměněno a například přesměrováno na jiné umístění v souborovém systému, registry, apod. V současné době Windows 7 obsahují více jak 250 různých možných modifikací API volání a je tak možné zajistit chod i starších, zdánlivě nekompatibilních aplikací, případně zajistit to, že aplikace bude bez problémů fungovat i v případě, že uživatel není administrátorem.